3D Slicer Documentation

Slicer Community

Jul 17, 2026

CONTENTS

For older Slicer documentation (4.10 and earlier), refer to the 3D Slicer wiki.

CHAPTER

ONE

ABOUT 3D SLICER

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, macOS 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 Extensions 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 broadly compatible with the Open Source Definition by The Open Source Initiative and contains no restrictions on legal uses of the software.

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

1.2.1 Historical notes about the license

The Slicer License was drafted in 2005 by lawyers working for Brigham and Women's Hospital (BWH), a teaching affiliate of Harvard Medical School, to be BSD-like but with a few extra provisions related to medical software. It is specific to BWH so it's not directly reusable, but it could serve as a template for projects with similar goals.

It was written in part because BWH was the prime contractor on an NIH-funded development consortium (NA-MIC) and wanted all code contributions to be compatible with ultimate use in real-world medical products (that is, commercial FDA-approved medical devices, which are almost universally closed source even if they build on open software). Compliance with the Slicer License was required for subcontractors, a group that included GE Research, Kitware and several universities (MIT, UNC…) who all reviewed and accepted this license.

The license has been in continuous use since 2005 for the 3D Slicer software package (slicer.org) that as of 2021 has been downloaded more than a million times and has been referenced in about 12,000 academic publications (https://www.slicer.org/wiki/Main_Page/SlicerCommunity). Some of the code is also now being used in several medical products for which this license has been reviewed and accepted by the companies involved.

1.2.2 License terms and reasons

Here are some of the key points that BWH included in addition to BSD terms to make the license suit the case of a large hospital distributing open source medical software.

For using and redistributing 3D Slicer:

- The license states that the code is "designed for research" and "CLINICAL APPLICATIONS ARE NEITHER RECOMMENDED NOR ADVISED" to make it extra clear that any commercial clinical uses of the code are solely the responsibility of the user and not BWH or the other developers. This is a disclaimer rather than a legal restriction.

For making changes or adding any source code or data to 3D Slicer:

- Contributors explicitly grant royalty free rights if they contribute code covered by a patent they control (i.e. to avoid submarine patents).
- No GPL or other copyleft code is allowed because that could make it complicated and risky to mix Slicer code with private intellectual property, which is often present in regulated medical products.
- Contributors affirm that they have de-identified any patient data they contribute to avoid issues with HIPAA or related regulations.

1.2.3 Status compared to other open source licenses

As of June 2021, the Slicer License has been used for over 15 years without incident. In May of 2021, a discourse user suggested submitting the license to the OSI license review process. After some discussion and hearing no objections, the community leadership decided to submit the license for review. Although the OSI process is not legally binding, the discussion could give potential Slicer users perspective on how provisions of the license compare with other commonly used licenses. The discussion concluded that bundling the contribution agreement in the license makes it non-approvable by OSI and the requirement to use the software for legal purposes may not be consistent with the Open Source Definition. Otherwise the license terms appear not to be controversial. Interested parties should review the full discussion for details.

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:

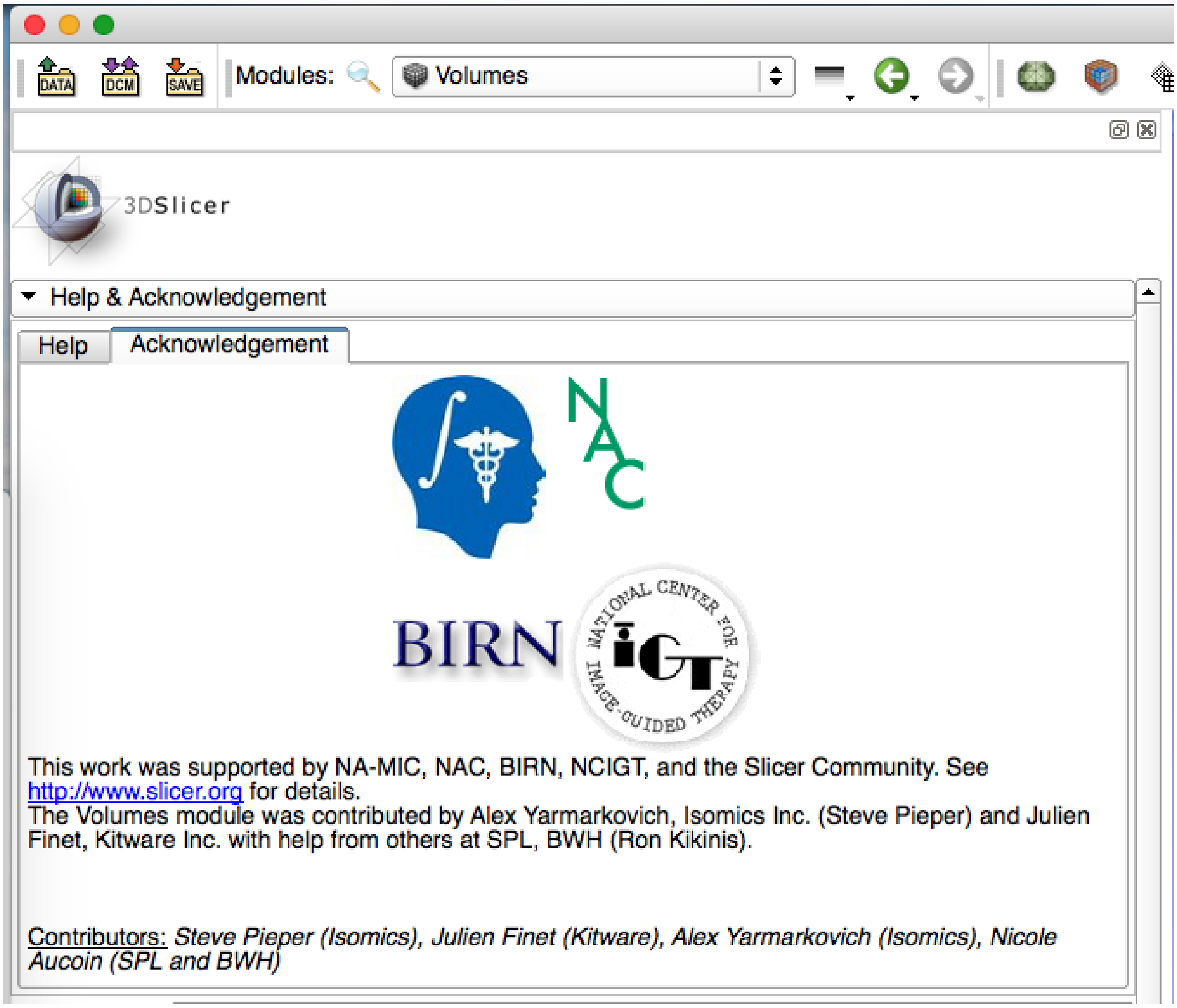
Fedorov A., Beichel R., Kalpathy-Cramer J., Finet J., Fillion-Robin J-C., Pujol S., Bauer C., Jennings D., Fennessy F.M., Sonka M., Buatti J., Aylward S.R., Miller J.V., Pieper S., Kikinis R. 3D Slicer as an Image Computing Platform for the Quantitative Imaging Network. Magnetic Resonance Imaging. 2012 Nov;30(9):1323-41. PMID: 22770690. PMCID: PMC3466397.

1.4 The 3D Slicer name and logo

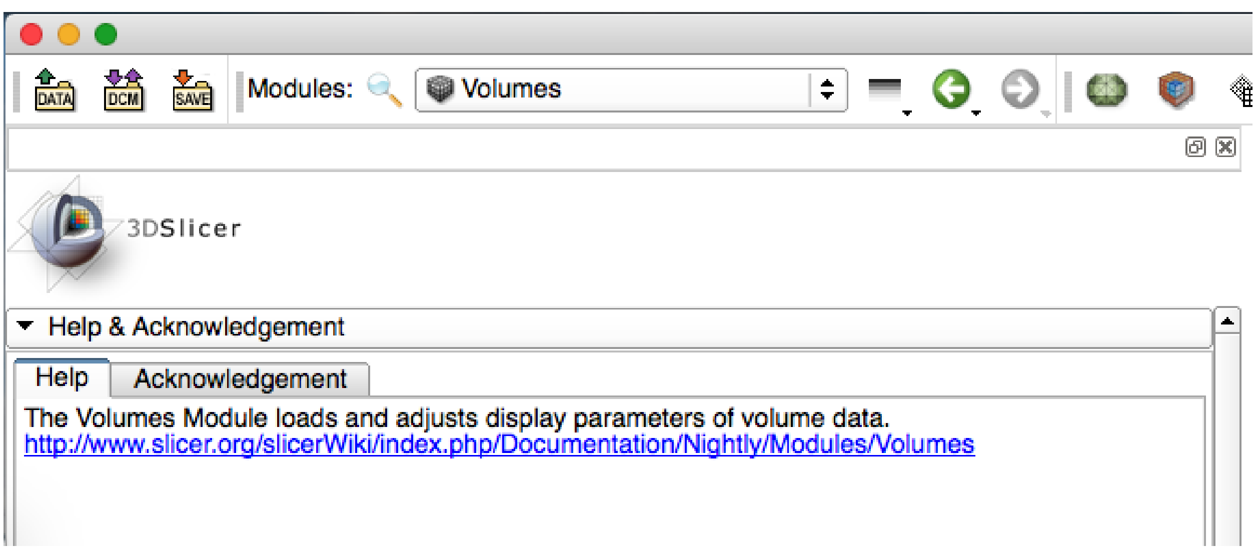
3D Slicer and the logo are trademarks of Brigham and Women's Hospital (BWH) and may not be used without permission. Such permission is broadly granted for academic or commercial uses, such as documenting the use of Slicer in your project or promoting the use of Slicer by others. Please use the original Slicer logo colors and do not alter the shape or text. Using Slicer to imply that BWH or the Slicer community endorses your product or project is not permitted without permission. For other uses please contact Ron Kikinis (kikinis@bwh.harvard.edu) and Steve Pieper (pieper@bwh.harvard.edu).

1.4.1 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:



1.5 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!

1.5.1 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.

1.5.2 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

1.5.3 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. Funding sources that contribute to development of Slicer core or extensions include (ordered by End Date):

1.6 Commercial Use

We invite commercial entities to use 3D Slicer.

1.6.1 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.

1.6.2 Commercial Partners

- Ebatinca SL is an international technology company in Las Palmas, Spain focused on technology for sustainable development. Primary areas: ultrasound navigation and training, collaborative VR, research support, custom Slicer-based solutions.
- 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 focuses on solving the world's most complex scientific challenges through customized software solutions. The company has a long history of contributing to open source platforms that serve as the foundation of many medical visualization and data processing applications. Kitware helps customers develop commercial products based on 3D Slicer and has used the platform to rapidly prototype solutions in nearly every aspect of medical imaging.

Listed in alphabetical order.

1.6.3 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: Allen Institute for Brain Science is developing Cell Locator, a Desktop application for manually aligning specimens to annotated 3D spaces. See more information on this Kitware blog.
- Dent.AI Medical Imaging is developing an AI-powered oral surgery planning system that generates realistic virtual patient models from CBCT and intraoral scan data to support navigated implant placement and bone reconstruction procedures.
- EBATINCA is developing StaRT, an interactive platform for simulation and education in radiation therapy, offering realistic clinical workflows and case-based exercises designed for students and clinical professionals worldwide. See more information on the product website.

- Polarean, Inc.: Polarean's XENOVIEW VDP is an FDA-approved software built on 3D Slicer for visualization and evaluation of lung ventilation. See more information on this Kitware blog.
- 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 developed 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. Their team is now part of Revvity, Inc. See more information on this Kitware blog and their product website.
- Xoran Technologies: Image-guided Platform for Deep Brain Stimulation Surgery. 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.

1.7 Governance

This section introduces the guiding principles and decision-making framework of the Slicer project, including the Code of Conduct and governance model that describe how decisions are made and who is responsible for maintaining the health of the project.

1.7.1 Code of Conduct

The Slicer community is committed to fostering a welcoming, respectful, and inclusive environment. Details are outlined in the Code of Conduct below.

Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment

- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders listed here via private message on the Slicer forum. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

Deviations

For a history of updates and revisions to this document, see https://github.com/Slicer/Slicer/commits/main/CODE_OF_CONDUCT.md.

1.7.2 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.

1.7.3 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.

1.8 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*.

CHAPTER

TWO

GETTING STARTED

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. For example, 3D Slicer + Jupyter notebook in a web browser is available for free via Binder service (no installation needed, the application can run in any web browser).

2.1.1 Operating system versions

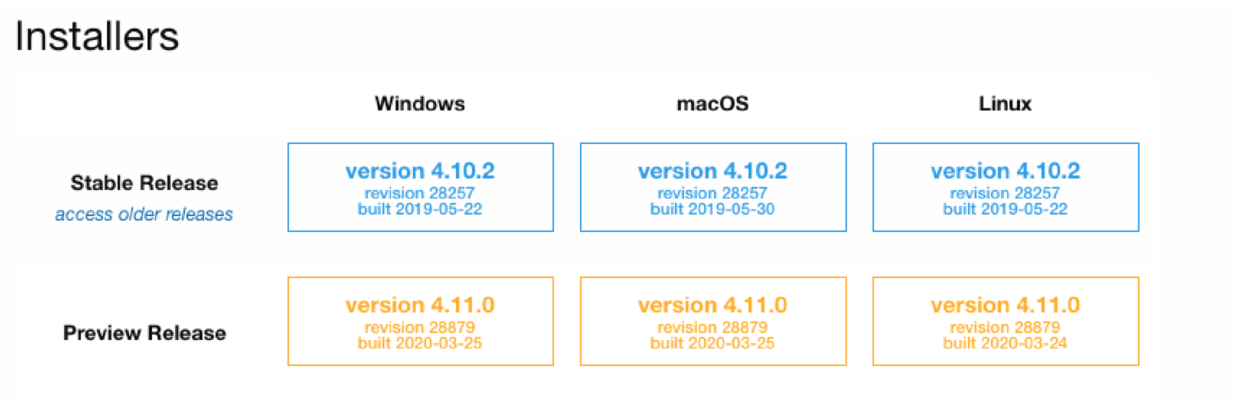
- Windows: Windows 11 with all recommended updates installed. Microsoft does not support Windows 10 (or older versions) anymore and Slicer is not tested on these legacy operating system versions, but may still work.
- macOS: macOS Sonoma (14) or later (both Intel and ARM based systems). Latest public release is recommended.
- Linux: Ubuntu 22.04 or laterDebian 11 or laterFedora 35 or laterAlmaLinux 8 or later. Latest LTS (Long-term-support) version is recommended.

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 1366 by 768 (1920 by 1080 or higher 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.



Notes:

- The "Preview Release" of 3D Slicer (referred to as "Slicer Preview Release") is updated daily (process starts at 11pm ET and takes few hours to complete) and represents the latest development including new features and fixes. *Continuous* updates for extensions are not provided for preview releases: extensions that are available for a preview release are of the latest version that was available at the time when the release was created. To get updated extensions, a new preview release must be installed.
- The "Stable Release" of 3D Slicer (referred to as "Slicer Stable Release") is usually updated a few times a year and is more rigorously tested. Continuous updates for extensions are provided for the *latest* Slicer Stable Release, but not for any earlier releases. For example, if latest Slicer Stable Release is Slicer-5.2.2 and the user has Slicer-5.2.1 installed then the user will no longer receive any extension updates. To check if latest version of application and extensions are installed, you can go to `Welcome` module and in `Updates` section click `Check now` button.
- 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*.
- 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.

Once downloaded, follow the instructions below to complete installation:

2.2.1 Windows

- Run the installer.
  - Current limitation: Installation path must only contain English (ASCII printable) characters because otherwise some Python packages may not load correctly (see this issue for more details).
- Run Slicer from the Windows start menu.
- Use "Apps & features" in Windows settings to remove the application.

2.2.2 Mac

- Open the install package (.dmg file).
- Drag the Slicer application (Slicer.app) to your Applications folder (or other location of your choice).
  - This step is necessary because content of a .dmg file is opened as a read-only volume, and you cannot install extensions or Python packages into a read-only volume.
- Delete the Slicer.app folder to uninstall.

Installing Preview Releases

Slicer 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:

- Try to open the app normally (it'll be blocked, but this step makes application temporarily highlighted in the system settings).
- Go to `System Settings` -> `Privacy & Security`.
- Scroll down - you'll see a message like "Slicer was blocked to protect your Mac" with an `Open Anyway` button.
- Click `Open Anyway`, then enter your password/Touch ID.
- A final confirmation dialog appears - click `Open`.

See more explanation and alternative techniques here.

Installing using Homebrew

Slicer can be installed with a single terminal command using the Homebrew package manager:

```
brew install --cask slicer  # to install
brew upgrade slicer         # to upgrade
brew uninstall slicer       # to uninstall
```

This procedure avoids the typical google-download-mount-drag process to install macOS applications.

Preview releases can be installed using `homebrew-cask-versions`:

```
brew tap homebrew/cask-versions     # needs to be run once
brew install --cask slicer-preview  # to install
brew upgrade slicer-preview         # to upgrade
brew uninstall slicer-preview       # to uninstall
```

2.2.3 Linux

- Open the tar.gz archive and copy directory to the location of your choice.
- Installation of additional packages may be necessary depending on the Linux distribution and version, as described in subsections below.
- Run the `Slicer` executable.
- Remove the directory to uninstall.

Notes:

- 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.

- Getting command-line arguments and process output containing non-ASCII characters requires the system to use a UTF-8 locale. If the system uses a different locale then the `export LANG="C.UTF-8"` command may be used before launching the application to switch to an acceptable locale.

Ubuntu 24.04 (Noble Numbat)

```
sudo apt-get install libglu1-mesa libpulse-mainloop-glib0 libnss3 libasound2t64 qt5dxcb-
↪plugin
```

Ubuntu 22.04 (Jammy Jellyfish), Debian 12 (bookworm), Debian 11 (bullseye)

```
sudo apt-get install libglu1-mesa libpulse-mainloop-glib0 libnss3 libasound2 qt5dxcb-
↪plugin libsm6
```

Warning

Debian 10.12 users may encounter an error when launching Slicer:

```
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to␣
↪run on Wayland anyway.
qt.qpa.plugin: Could not load the Qt platform plugin "xcb" in "" even though it was␣
↪found.
This application failed to start because no Qt platform plugin could be initialized.␣
↪Reinstalling the application may fix this problem.

Available platform plugins are: xcb.
```

The solution is to create symlink to either `libxcb-util.so` or `libxcb-util.so.0.0.0` depending on which library is present. Thus the command should be:

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libxcb-util.so /usr/lib/x86_64-linux-gnu/libxcb-
↪util.so.1
```

or:

```
sudo ln -s /usr/lib/x86_64-linux-gnu/libxcb-util.so.0.0.0 /usr/lib/x86_64-linux-gnu/
↪libxcb-util.so.1
```

ArchLinux

There are user-contributed packages on AUR

- 3dslicer-bin: this package repacks the official binary for convenient. Note the official Slicer is build with `Slicer_STORE_SETTINGS_IN_APPLICATION_HOME_DIR=ON`, is installed to `/opt` where non-root user has no write permission, user will not be able to install extensions or update application settings.
- 3dslicer: you could build the package from the source using this `PKGBUILD` file, or just install it from an unofficial repository: archlinuxcn repo.
- 3dslicer-git: same as 3dslicer but using the latest source.

Fedora 40, 39, 38, 37, 36, 35

Install the dependencies:

```
sudo dnf install mesa-libGLU mesa-libGL libnsl libXrender pulseaudio-libs-glib2 nss␣
↪libXcomposite libXdamage libXrandr ftgl libXcursor libXi libXtst alsa-lib qt5-
↪qtx11extras
```

Warning

The included libcrypto.so.1.1 in the Slicer installation is incompatible with the system libraries used by Fedora 35. The fix, until it is updated, is to move/remove the included libcrypto files:

```
$SLICER_ROOT/lib/Slicer-5.xx/libcrypto.*
```

If double-clicking the app does not launch it, try the following steps in order:

1. Verify execution rights of the app
   - Right-click on the app → *Properties* → make sure **"Executable as a program"** is checked.
   - Or check from a terminal:

     ```
     ls -la
     ```

     You should see something like `-rwxr-xr-x` at the beginning of the line for the executable.

     Note

     Some archive managers (e.g., certain `unrar`/`unzip` tools) may require running extraction with `sudo` to restore the correct file permissions.

     If execution rights are missing, add them with:

     ```
     chmod +x Slicer
     ```

2. Launch from the terminal

   ```
   ./Slicer
   ```

3. If you get the error

   ```
   error: Failed to obtain launcher executable name !
   ```

   try moving the executable into your *Software Home* folder and double-click the app again. It should launch properly from there.

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.

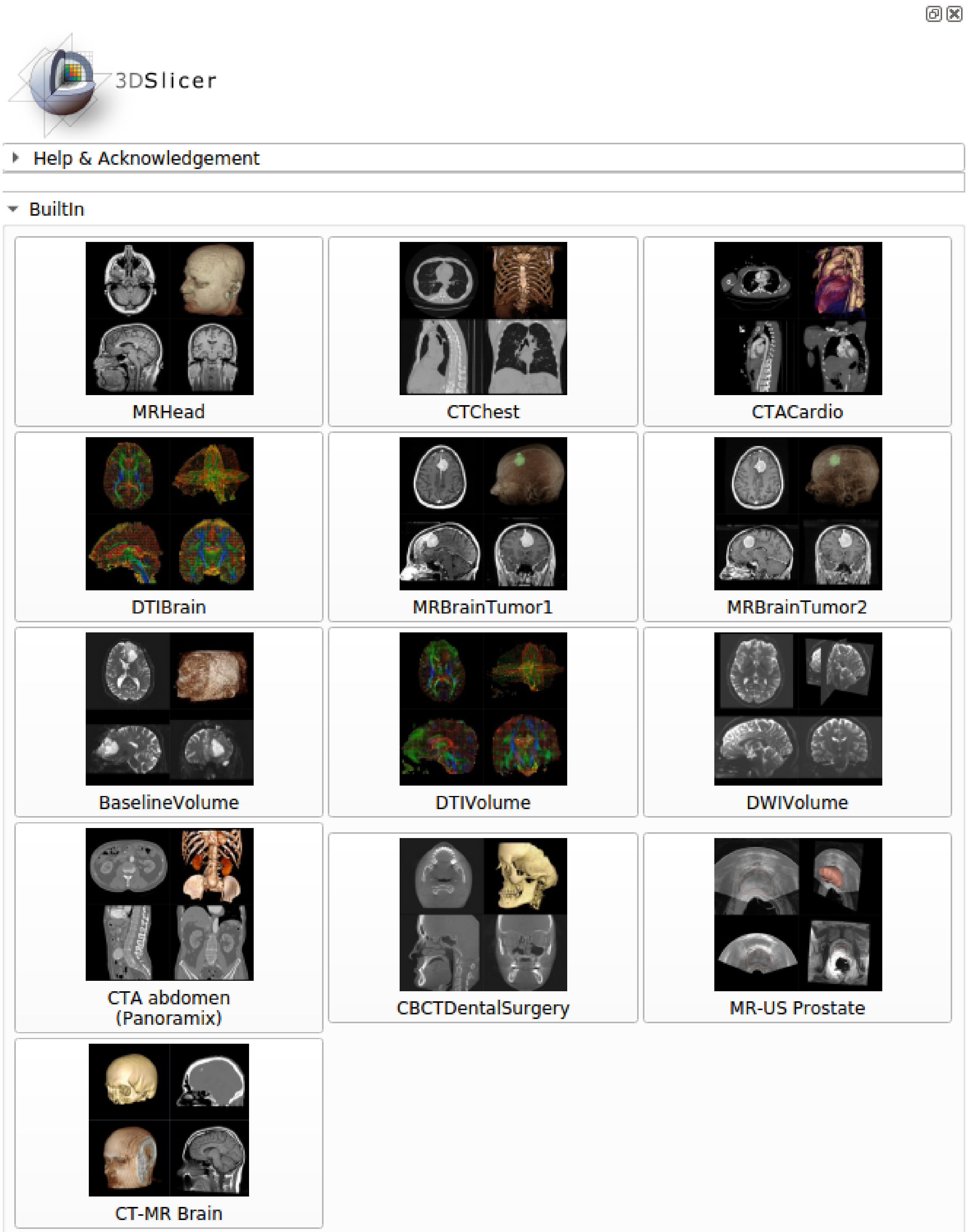
How to learn Slicer?

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

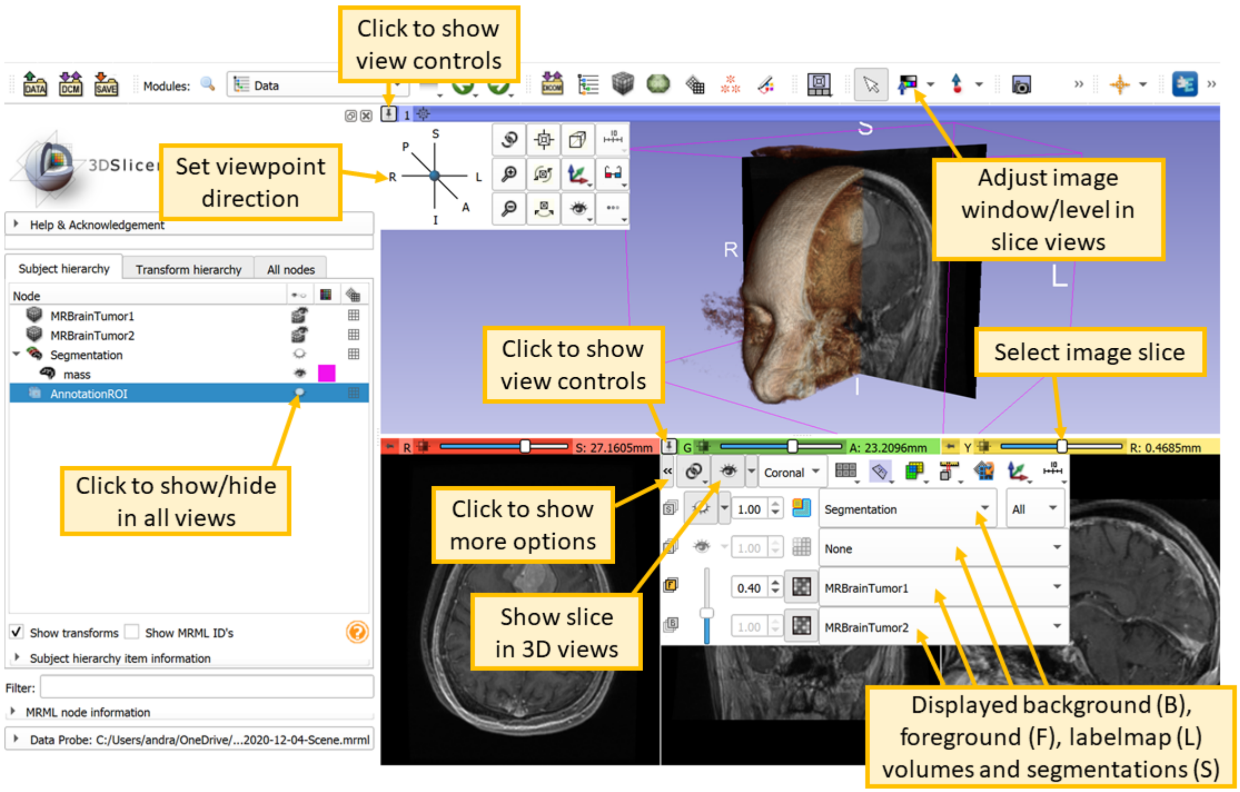
Open 3D Slicer and using the Welcome panel either load your own data or download sample data to explore. Sample data is often useful for trying the features of 3D Slicer if you don't have data of your own.


3DSlicer
Help & Acknowledgement
BuiltIn
MRHead
CTChest
CTACardio
DTIBrain
MRBrainTumor1
MRBrainTumor2
BaselineVolume
DTIVolume
DWIVolume
CTA abdomen
(Panoramix)
CBCTDentalSurgery
MR-US Prostate
CT-MR Brain

View data

Data module's Subject hierarchy tab shows all data sets in the scene. Click the "eye" icon to show/hide an item in all views.

You can customize views (show orientation marker, ruler, change orientation, transparency) 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 following (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.

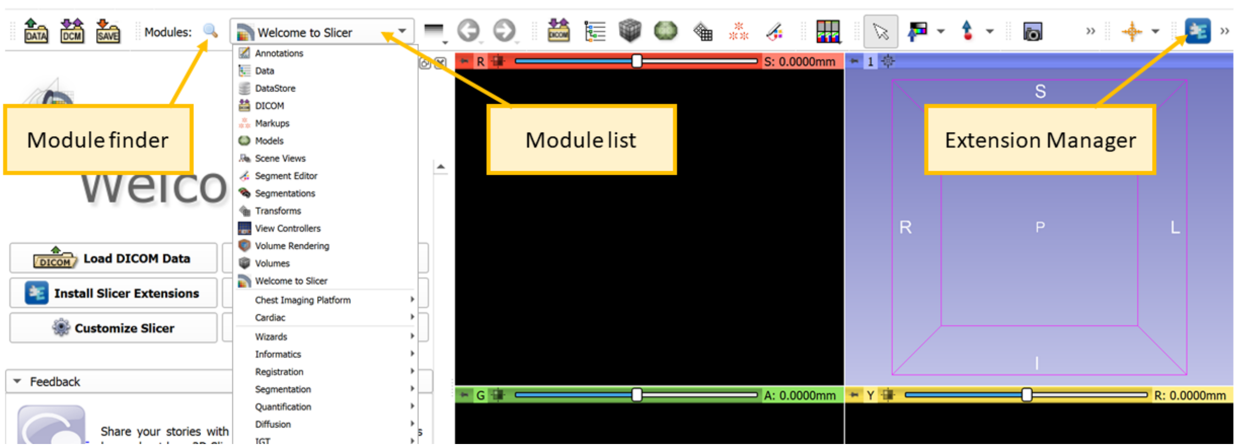
- *Models*: Loads and adjusts display parameters of models. Allows the user to change the appearance of and organize 3D surface models.
- *Transforms*: This module is used for creating and editing transformation matrices. You can establish these relations by moving nodes from the Transformable list to the Transformed list or by dragging the nodes under the Transformation nodes in the Data module.

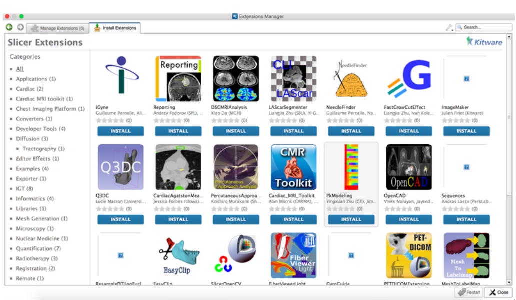
Save data

All data in the scene can be saved at once using `File` menu -> `Save data`, or selected data sets can be exported from the `Data` module by right-clicking and selecting `Export to file...` or `Export to DICOM...` . Details are described in the *Data loading and saving section*.

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 extensions manager to selectively install features that are useful for the end-user.





For details about downloading extensions, see *Extensions 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 Interface* 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!

2.4 Glossary

Terms used in various fields of medical and biomedical image computing and clinical images are not always consistent. This section defines terms that are commonly used in 3D Slicer, especially those that may have different meaning in other contexts.

- **Bounds**: Describes bounding box of a spatial object along 3 axes. Defined in VTK by 6 floating-point values: `X_min`, `X_max`, `Y_min`, `Y_max`, `Z_min`, `Z_max`.
- **Brightness/contrast**: Specifies linear mapping of voxel values to brightness of a displayed pixel. Brightness is the linear offset, contrast is the multiplier. In medical imaging, this linear mapping is more commonly specified by window/level values.
- **Cell**: Data cells are simple topological elements of meshes, such as lines, polygons, tetrahedra, etc.
- **Color legend** (or color bar, scalar bar): a widget overlaid on slice or 3D views that displays a color legend, indicating meaning of colors.
- **Coordinate system** (or coordinate frame, reference frame, space): Specified by position of origin, axis directions, and distance unit. All coordinate systems in 3D Slicer are right-handed.
- **Extension** (or Slicer extension): A collection of modules that is not bundled with the core application but can be downloaded and installed using the Extensions manager.
- ***Extensions manager***: A software component of Slicer that allows browsing, installing, uninstalling extensions in the Extensions catalog (also known as the Slicer app store) directly from the application.
- **Extensions index**: A repository that contains description of each extension that the Extension catalog is built from.
- **Extent**: Range of integer coordinates along 3 axes. Defined in VTK by 6 values, for IJK axes: `I_min`, `I_max`, `J_min`, `J_max`, `K_min`, `K_max`. Both minimum and maximum values are inclusive, therefore size of an array is `(I_max - I_min + 1)` x `(J_max - J_min + 1)` x `(K_max - K_min + 1)`.
- **Fiducial**: Represents a point in 3D space. The term originates from image-guided surgery, where "fiducial markers" are used to mark point positions.
- **Frame**: One time point in a time sequence. To avoid ambiguity, this term is not used to refer to a slice of a volume.
- **Geometry**: Specifies location and shape of an object in 3D space. See "Volume" term for definition of image geometry.

- **Image intensity**: Typically refers to the value of a voxel. Displayed pixel brightness and color is computed from this value based on the chosen window/level and color lookup table.
- **IJK**: Voxel coordinate system axes. Integer coordinate values correspond to voxel center positions. IJK values are often used as coordinate values to designate an element within a 3D array. By VTK convention, and I indexes the column, J indexes the row, K indexes the slice. Note that numpy uses the opposite ordering convention, where `a[K][J][I]`. Sometimes this memory layout is described as I being the fastest moving index and K being the slowest moving.
- **ITK**: Insight Toolkit. Software library that Slicer uses for most image processing operations.
- **Labelmap** (or labelmap volume, labelmap volume node): Volume node that has discrete (integer) voxel values. Typically each value corresponds to a specific structure or region. This allows compact representation of non-overlapping regions in a single 3D array. Most software use a single labelmap to store an image segmentation, but Slicer uses a dedicated segmentation node, which can contain multiple representations (multiple labelmaps to allow storing overlapping segments; closed surface representation for quick 3D visualization, etc.).
- **LPS**: Left-posterior-superior anatomical coordinate system. Most commonly used coordinate system in medical image computing. Slicer stores all data in LPS coordinate system on disk (and converts to/from RAS when writing to or reading from disk).
- **Markups**: Simple geometric objects and measurements that the user can place in viewers. *Markups module* can be used to create such objects. There are several types, such as point list, line, curve, plane, ROI.
- **Source volume**: Voxel values of this volume is used during segmentation by those effects that rely on intensity of an underlying volume.
- **MRML**: Medical Reality Markup Language: Software library for storage, visualization, and processing of information objects that may be used in medical applications. The library is designed to be reusable in various software applications, but 3D Slicer is the only major application that is known to use it.
- **Model** (or model node): MRML node storing surface mesh (consists of triangle, polygon, or other 2D cells) or volumetric mesh (consists of tetrahedral, wedge, or other 3D cells)
- **Module** (or Slicer module): A Slicer module is a software component consisting of a graphical user interface (that is displayed in the module panel when the module is selected), a logic (that implements algorithms that operate on MRML nodes), and may provide new MRML node types, displayable managers (that are responsible for displaying those nodes in views), input/output plugins (that are responsible for load/save MRML nodes in files), and various other plugins. Modules are typically independent and only communicate with each other via modifying MRML nodes, but sometimes a module use features provided by other modules by calling methods in its logic.
- **Node** (or MRML node): One data object in the scene. A node can represent data (such as an image or a mesh), describe how it is displayed (color, opacity, etc.), stored on disk, spatial transformations applied on them, etc. There is a C++ class hierarchy to define the common behaviors of nodes, such as the property of being storable on disk or being geometrically transformable. The structure of this class hierarchy can be inspected in the code or in the API documentation.
- **Orientation marker**: Arrow, box, or human shaped marker to show axis directions in slice views and 3D views.
- **RAS**: Right-anterior-superior anatomical coordinate system. Coordinate system used internally in Slicer. It can be converted to/from LPS coordinate system by inverting the direction of the first two axes.
- **Reference**: Has no specific meaning, but typically refers to a secondary input (data object, coordinate frame, geometry, etc.) for an operation.
- **Region of interest (ROI)**: Specifies a box-shaped region in 3D. Can be used for cropping volumes, clipping models, etc.
- **Registration**: The process of aligning objects in space. Result of the registration is a transform, which transforms the "moving" object to the "fixed" object.

- **Resolution**: Voxel size of a volume, typically specified in mm/pixel. It is rarely used in the user interface because its meaning is slightly misleading: high resolution value means large spacing, which means coarse (low) image resolution.
- **Ruler**: It may refer to: 1. View ruler: The line that is displayed as an overlay in viewers to serve as a size reference. 2. Markups line: distance measurement tool.
- **Scalar component**: One element of a vector. Number of scalar components means the length of the vector.
- **Scalar value**: A simple number. Typically floating-point.
- **Scene** (or MRML scene): This is the data structure that contains all the data that is currently loaded into the application and additional information about how they should be displayed or used. The term originates computer graphics.
- **Segment**: Corresponds to single structure in a segmentation. See more information in *Image segmentation* section.
- **Segmentation** (also known as contouring, annotation; region of interest, structure set, mask): Process of delineating 3D structures in images. Segmentation can also refer to the MRML node that is the result of the segmentation process. A segmentation node typically contains multiple segments (each segment corresponds to one 3D structure). Segmentation nodes are not labelmap nodes or model nodes but they can store multiple representations (binary labelmap, closed surface, etc.). See more information in *Image segmentation* section.
- **Slice**: Intersection of a 3D object with a plane.
- **Slice view annotations**: text in corner of slice views displaying name, and selected DICOM tags of the displayed volumes
- **Spacing**: Voxel size of a volume, typically specified in mm/pixel.
- **Transform** (or transformation): Can transform any 3D object from one coordinate system to another. Most common type is rigid transform, which can change position and orientation of an object. Linear transforms can scale, mirror, shear objects. Non-linear transforms can arbitrarily warp the 3D space. To display a volume in the world coordinate system, the volume has to be resampled, therefore transform *from* the world coordinate system to the volume is needed (it is called the resampling transform). To transform all other node types to the world coordinate system, all points must be transformed *to* the world coordinate system (modeling transform). Since a transform node must be applicable to any nodes, transform nodes can provide both *from* and *to* the parent (store one and compute the other on-the-fly).
- **Volume** (or volume node, scalar volume, image): MRML node storing 3D array of voxels. Indices of the array are typically referred to as IJK. Range of IJK coordinates are called extents. Geometry of the volume is specified by its origin (position of the IJK=(0,0,0) point), spacing (size of a voxel along I, J, K axes), axis directions (direction of I, J, K axes in the reference coordinate system) with respect to a frame of reference. 2D images are single-slice 3D volumes, with their position and orientation specified in 3D space.
- **Voxel**: One element of a 3D volume. It has a rectangular prism shape. Coordinates of a voxel correspond to the position of its center point. Value of a voxel may be a simple scalar value or a vector.
- **VR**: Abbreviation that can refer to volume rendering or virtual reality. To avoid ambiguity it is generally recommended to use the full term instead (or explicitly define the meaning of the abbreviation in the given context).
- **VTK**: Visualization Toolkit. Software library that Slicer uses for to data representation and visualization. Since most Slicer classes are derived from VTK classes and they heavily use other VTK classes, Slicer adopted many conventions of VTK style and application programming interface.
- **Window/level** (or window width/window level): Specifies linear mapping of voxel values to brightness of a displayed pixel. Window is the size of the intensity range that is mapped to the full displayable intensity range. Level is the voxel value that is mapped to the center of the full displayable intensity range.

CHAPTER

THREE

GET HELP

Contact the Slicer community or commercial partners if you have any questions, bug reports, or enhancement requests - following the guidelines described below.

3.1 I need help in using Slicer

- You can start with typing your question into Google web search. There is a good chance that your question has been asked and answered before and all questions ever asked about Slicer are publicly available and indexed by Google. Most up-to-date information sources are the Slicer forum and Slicer documentation on read-the-docs. Google may find older discussions on former Slicer mailing lists and wiki pages, which may or may not be exactly accurate for the current version of Slicer, but may still provide useful hints.
- Try your best to sort out the issue by reading documentation, portfolio of training materials, and checking error logs (in application menu bar: View->Error log).
- If you are still unclear about what to do: ask a question on the Slicer Forum. In addition to describing the specific question, it helps if you describe the context of your question (who you are, what you are working on, why it is important, what is the overall goal of your project). Knowing more about you and your project increases the chance that somebody volunteers to answer the question and you may get a more relevant answer.

3.2 I want to report a problem

We are constantly making fixes and improvements, therefore before you report any issues:

1. Confirm the issue still exists in the current Slicer Stable Release.
2. If it still exists in the Slicer Stable Release, confirm the issue still exists in the latest Slicer Preview Release.

If you find it hard to locate answers using the search feature of the Slicer forum or via internet searches such as Google, try a large language model like Microsoft Copilot, OpenAI ChatGPT, Google Gemini, etc.

If you are not sure if Slicer behaves incorrectly or you are not using it properly then ask about it on the Slicer Forum (in the `Support` category). If you are *sure* that Slicer is not working as intended then submit a bug report in the Slicer issue tracker.

In your question/report provide all the information that is described in the bug reporting template.

Tip

Don't be anonymous: real people trying hard to solve real problems are more likely to get valuable help. If you tell about yourself and your project then it may get more attention and the problem may be resolved sooner.

3.3 I would like to request enhancement or new feature

First search on the Slicer forum and in the Slicer issue tracker to see if someone asked for this feature already. If you find a very similar request, tell us that you are interested in it too by adding a comment and/or adding a "thumbs-up" to the top post.

If you cannot find a similar feature request, then write a post in the Feature request category to discuss it with Slicer developers and community members.

> **Tip**
>
> If you write about yourself and your project then there is a higher chance that your request will be worked on. Describe what assistance you can offer for the implementation (your own time, funding, etc.).

3.4 I would like to let the Slicer community know, how Slicer helped me in my research

Please send us the citation for your paper posting in Community category in Slicer forum.

Background: Funding for Slicer is provided through competitive mechanisms primarily by the United States government and to a lesser extent through funding from other governments. The justification for those resources is that Slicer enables scientific work. Knowing about scientific publications enabled by Slicer is a critical step in this process. Given the international nature of the Slicer community, the nationality of the scientists is not important. Every good paper counts.

3.5 Troubleshooting

3.5.1 Slicer application does not start

- Your computer CPU or graphics capabilities may not meet *minimum system requirements*.
  - Updating your graphics driver may fix some problems, but if that does not help and you have an old computer then you may need to upgrade to a more recently manufactured computer or switch to a software renderer. A software renderer is particularly useful for running Slicer on a headless machine, such as a virtual machine at a cloud computing provider with strong CPU but no GPU, using Remote Desktop Protocol.

> **Note**
>
> **Setting up software renderer on Windows:**
>
> * Download Mesa OpenGL driver from https://github.com/pal1000/mesa-dist-win/releases (MSVC version - mesa3d-X.Y.Z-release-msvc.7z). Last tested with release https://github.com/pal1000/mesa-dist-win/releases/tag/22.2.0
> * Extract the archive and copy files from the x64 folder into the bin subfolder in the Slicer install tree.
> * Configure the rendere by setting environment variables then launch Slicer:
>
> ```
> set GALLIUM_DRIVER=llvmpipe
> set MESA_GL_VERSION_OVERRIDE=3.3COMPAT
> Slicer.exe
> ```
>
> This software renderer has been tested to work well on Windows virtual machines on Microsoft Azure.

- Slicer may not work if it is installed in a folder that has special characters in their name. Try installing Slicer in a path that only contains latin letters and numbers (a-z, 0-9).
- Your Slicer settings might have become corrupted
  - Try launching Slicer using `Slicer.exe --disable-settings` (if it fixes the problem, delete Slicer.ini and Slicer-.ini files from your Slicer settings directory.
  - Rename or remove your Slicer settings directory (for example, `c:\Users\<yourusername>\AppData\Roaming\slicer.org`). See instructions for getting the settings directory *here*. Try to launch Slicer.
- There may be conflicting/incompatible libraries in your system path (most likely caused by installing applications that place libraries in incorrect location on your system). Check your system logs for details and report the problem.
  - On Windows:
    - Start Event Viewer (eventvwr.exe), select Windows Logs / Application, and find the application error. If there is a DLL loading problem a line similar to this will appear: `Faulting module path: <something>.dll`. If you found a line similar to this, then try the following workaround: Start a command window. Enter `set path=` to clear the path variable. Enter Slicer.exe to start Slicer. If Slicer starts successfully then you need to remove unnecessary items from the system path (or delete the libraries installed at incorrect locations).
    - If Slicer still does not work then collect some more information and report the problem:
      - Get DLL dependency information using Dependency Walker tool:
      - Download depends.exe from here
      - Run depends.exe using the Slicer launcher: `Slicer.exe --launch path\to\depends.exe "bin\SlicerApp-real.exe"`
      - In dependency walker: Make sure the full path of DLLs are shown (click View / Full paths if you only see the DLL names). Use File / Save as… => Comma Separated Values (*.csv) to save logs to a file.
      - Enable process loading logging using the sxstrace tool, start Slicer, and save the log file (see instructions here)
  - On Linux:
    - Some linux versions require building your own kerberos and openssl as described and tracked in this issue.

CHAPTER

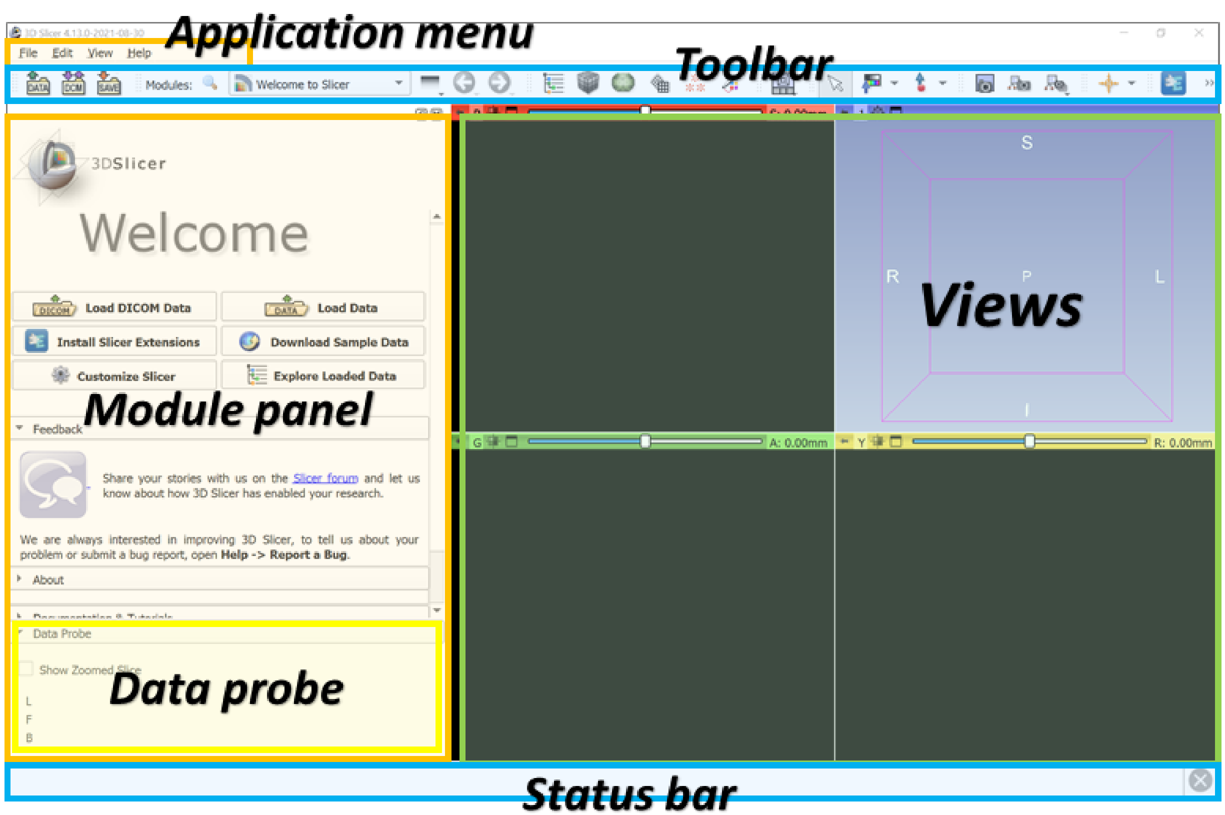
FOUR

USER INTERFACE

4.1 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 Extensions Manager.



4.1.1 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

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

- Slice view information (displayed when the mouse is over a slice view):
  - Slice view name: `Red`, `Green`, `Yellow`, etc.
  - Anatomical position: three coordinate values, prefixed with `R/L` (right/left), `A/P` (anterior/posterior), `S/I` (superior/inferior). The origin - (0,0,0) position - was chosen by the imaging technologist when the image was created. For example `(R 17.6, P 35.3, S 12.1)` for a clinical image means that the current position is 17.6mm to the right from the origin, 35.3mm towards posterior, 12.1mm superior from the origin.
  - View orientation: `Axial`, `Sagittal`, `Coronal` for standard anatomical orientations, and `Reformat` for any other orientation.
  - Slice spacing: distance between slices in this orientation. For a clinical image `Sp:  2.5` means that slices are 2.5mm distance from each other.
- Volume layer information: three lines, one for each volume layer
  - Layer type: `L` (label), `F` (foreground), `B` (background).
  - Volume name, or `None` if no volume is selected for that layer.
  - Volume voxel (IJK) coordinates.
  - Voxel value. For label volumes the label name corresponding to the voxel value is also displayed.
- Segmentation information: for each segmentation visible at that position
  - Layer type: `S` (segmentation)
  - Segmentation name.
  - Segment names. Multiple segment names are listed if multiple segments are displayed at that position (the segments overlap).

4.1.2 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.

4.1.3 Application Menu

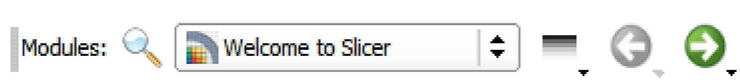
- **File**: Functions for loading a previously saved scene or individual datasets of various types, and for downloading sample datasets from the internet. An option for saving scenes and data is also provided here. **Add Data** allows loading data from files. **DICOM** module is recommended to import 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 **Extensions Manager** for installing extensions from Slicer app store, **Error Log** for checking if the application encountered any potential errors, **Python Console** for getting a Python console to interact with the loaded data or modules, **show/hide toolbars**, or **switch view layout**.
- **Help**: Contains links to documentation, guides, and community sites. **Report a Bug** provides instructions for bug reporting and and convenient access to log messages. There are also links to search for **Slicer Publications** and learn **How to Cite** Slicer.

4.1.4 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 dropdown button** shows the list of recently used modules. **Arrow buttons** can be used for going back to/returning from previously used module.

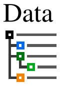


Favorite modules toolbar contains a list of most frequently used modules. The list can be customized using menu: Edit / Application settings / Modules / Favorite Modules. Drag-and-drop modules from the Modules list to the Favorite Modules list to add a module.

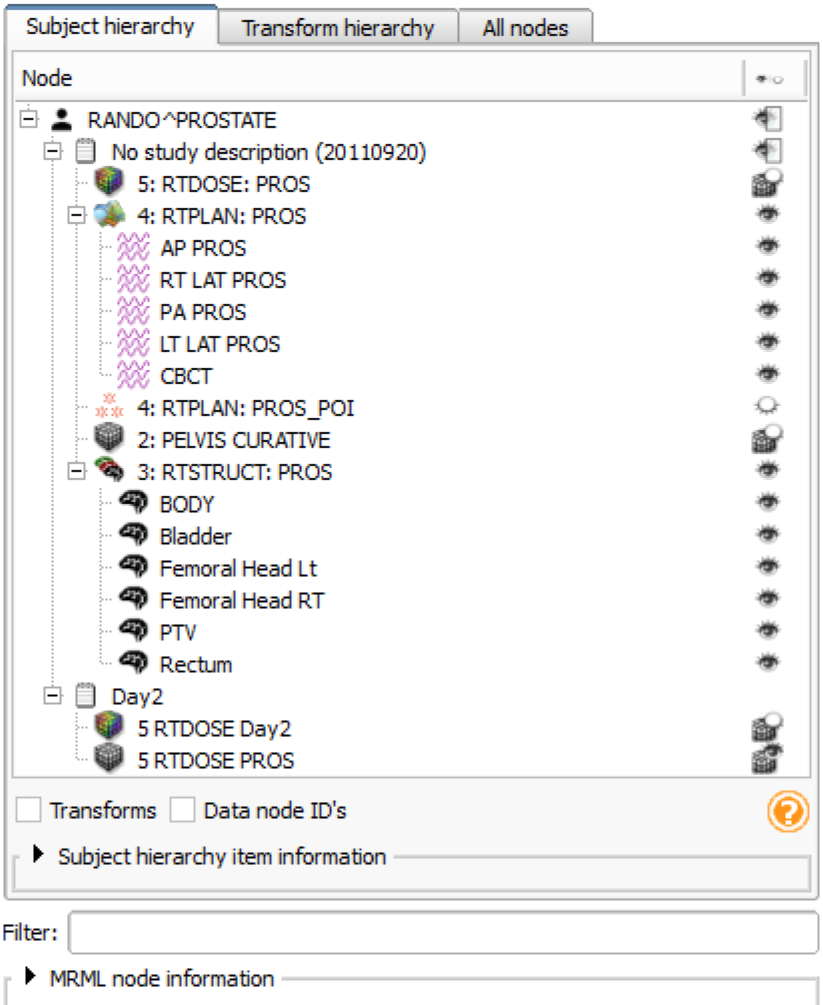
4.1.5 Status bar

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

4.2 Review loaded data

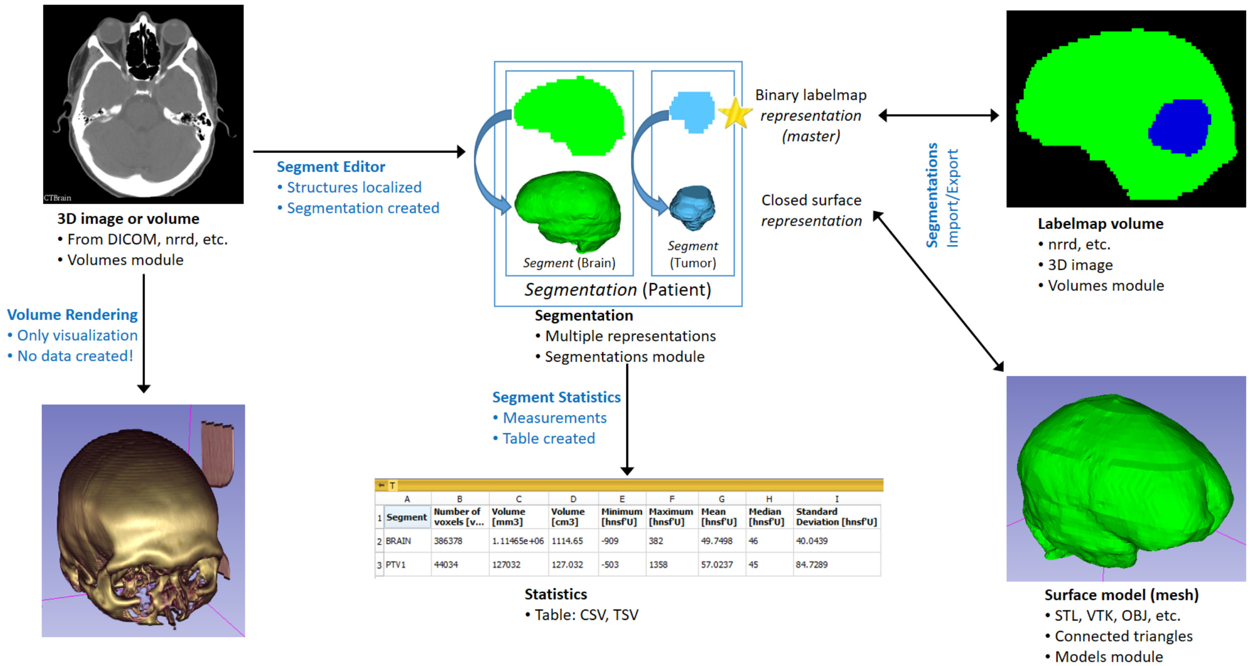
Data available in Slicer can be reviewed in the Data module, which can be found on the toolbar or the modules list . More details about the module can be found on the Slicer wiki.

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.



4.2.1 Selecting displayed data

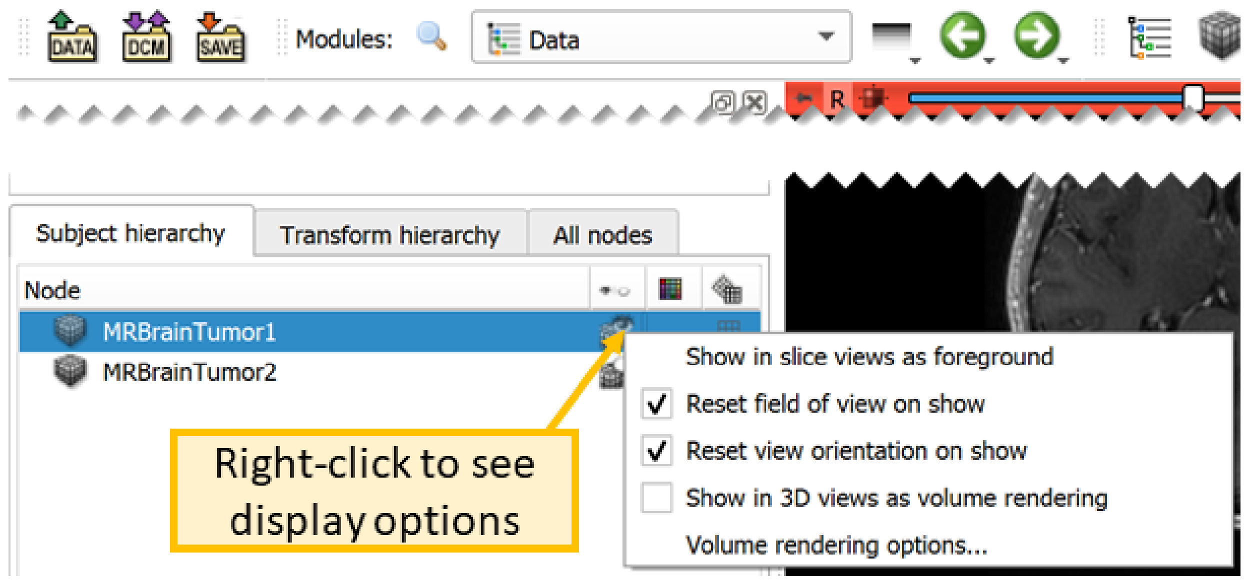
Data module's Subject hierarchy tab shows all data sets in the scene. Click the "eye" icon to show/hide an item in all views. Drag-and-drop an item into a view to show it in that view.

Multiple items can be selected in the subject hierarchy tree using Ctrl-Left-Click or Shift-Left-Click and dragged at once into selected view. If two volumes are dragged into a view at the same time then they will be both shown, blended together.

If a view is displayed only in selected views, you can right-click on the item and select "Show in all views" to quickly show in all views.

If view link is enabled for a slice view then dragging a volume to any of the views will show the volume in all the views in that group.

Display options can be adjusted by right-clicking the eye icon in the display column of the tree. Note that these options are different from options that are offered when right-clicking on the "Node" or "Transform" column in the tree.



For volumes, display options include:

- Reset field of view on show: if enabled, then showing a volume makes adjust views to show the volume in the center, filling the field of view.
- Reset view orientation on show: if enabled, then showing a volume makes the slice views aligned with the volume axes.

4.3 Interacting with views

4.3.1 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.

4.3.2 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 Point List 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.

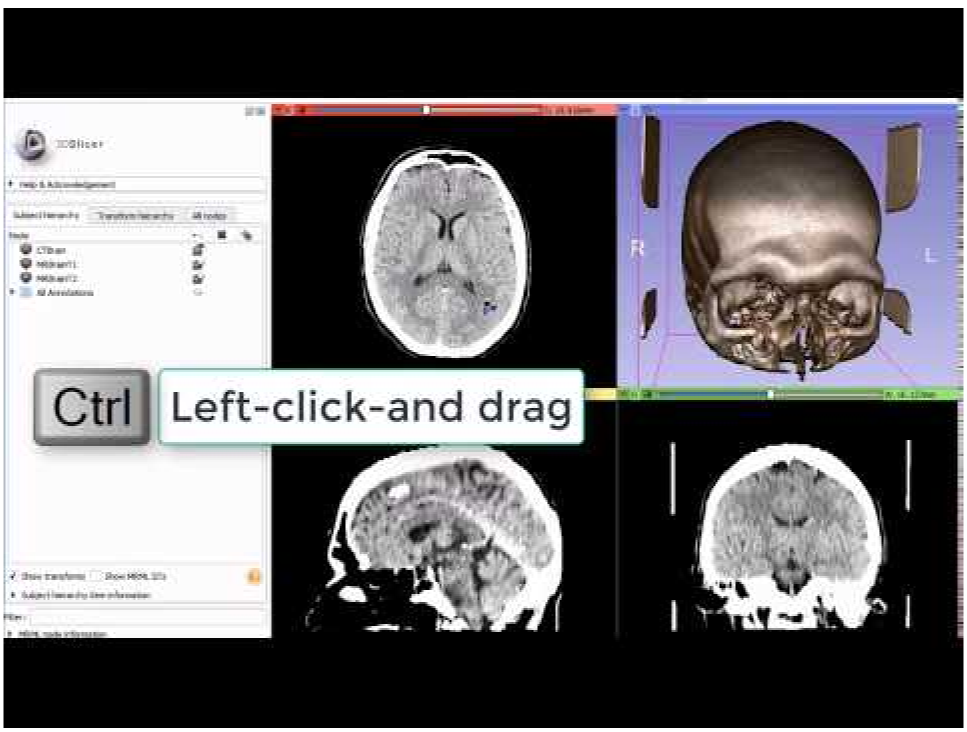
> **Note**
>
> Transform mode is the default interaction mode. By default, Place mode persists for one “place” operation after the Place Mode icon is selected, and then the mode switches back to Transform. Place mode can be made persistent (useful for placing multiple control points) by checking the 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 to contain the entire intensity range of the image (except top/bottom 0.1%, calculated using percentiles, 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 lef-click-and-dragging while holding down Ctrl key.

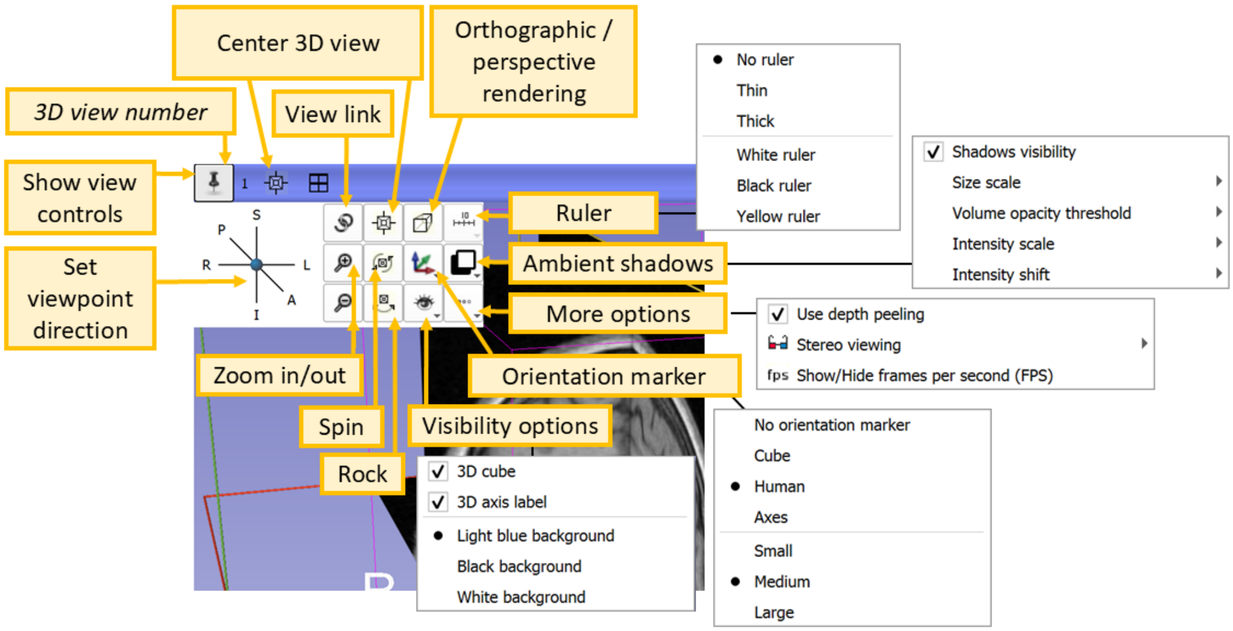


Additional window/level options, presets, intensity histogram, automatic adjustments are available in Display section of *Volumes* module. Presets are also available in the context menu of the Slice views. You can reset to the automatically calculated window/level settings using the context menu or by using `Ctrl` + `left-double-click` on the Slice view.

4.3.3 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.



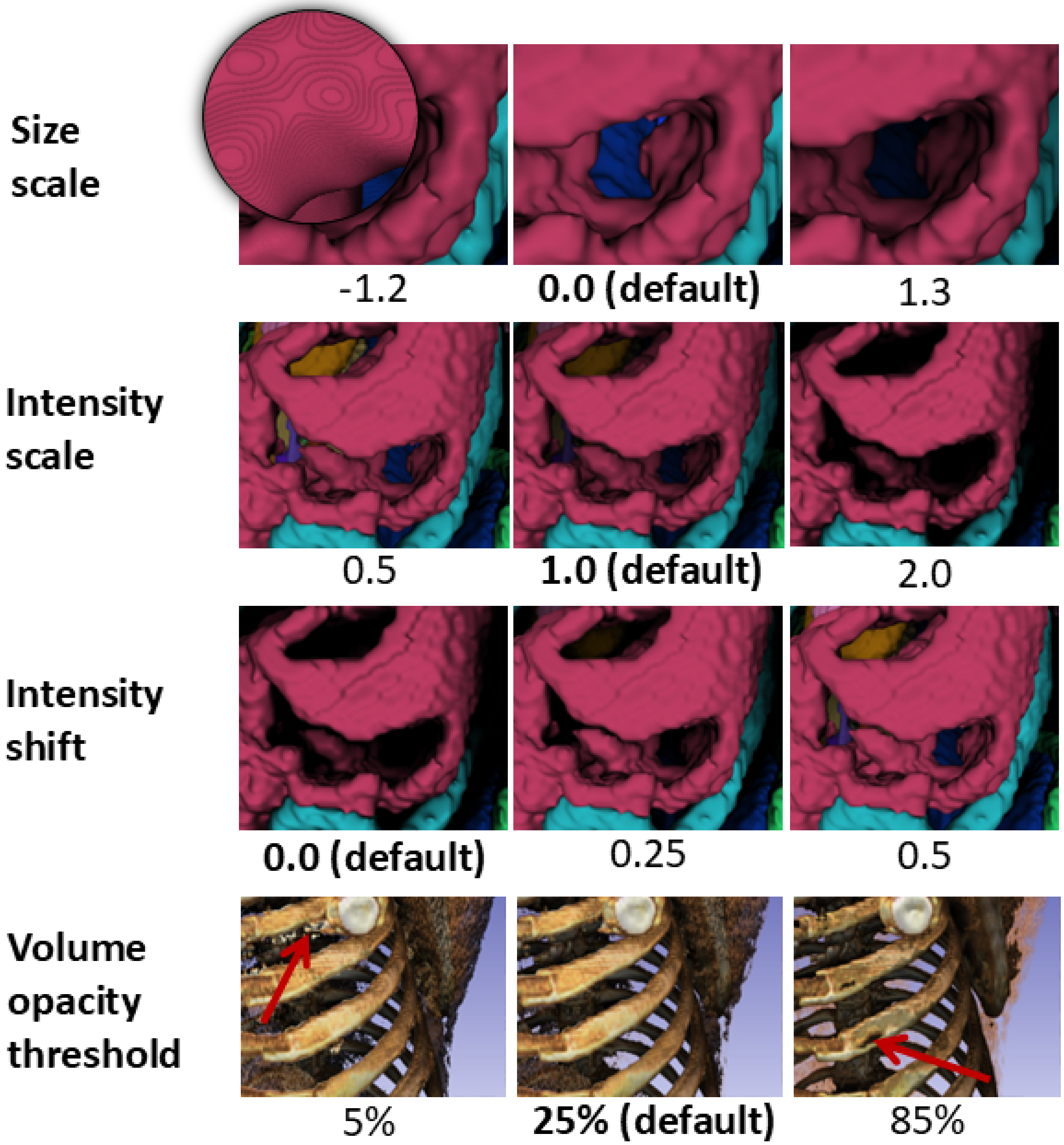
3D View Controls: The blue bar across any 3D View shows a pushpin icon on its left. When the mouse rolls over this icon, a panel for configuring the 3D View is displayed. The panel is hidden when the mouse moves away. For persistent display of this panel, just click the pushpin icon.

- **Center 3D view** (small square) centers the slice on the currently visible 3D view content and all loaded volumes (even if volumes that are not visible). The field of view (zoom factor) is not adjusted, therefore it may be necessary to zoom in/out to see all objects. To reset the center and field of view at the same time, click in the 3D view and hit r key.
- **Maximize view / Restore view layout** temporarily maximizes the selected view / restores the full view layout.
- **Viewpoint direction** switches orientation of the view between standard directions. Clicking on **L**eft, **R**ight, **A**nterior, **P**osterior, **S**uperior, **I**nferior button will make the 3D content viewed from that direction.
- **View link** button synchronizes properties across 3D views (viewpoint position, direction, ruler, orientation marker, etc. settings).
- **Orthographic/perspective rendering** mode toggle. Orthographic mode (parallel projection) is useful for assessing size, because displayed object size does not depend on distance from the viewpoint. Perspective mode provides better depth perception, because objects that are closer appear larger.
- **Ruler** controls display of ruler. Only available in orthographic rendering mode.
- **Stereo viewing** enables stereoscopic display. Red/blue and anaglyph modes just require inexpensive red/blue colored glasses. Other modes require special 3D display hardware. Note that SlicerVirtualReality extension offers superior stereo viewing and interaction experience, with fully immersive 3D visualization by a single click of a button, and rich interaction with objects in the scene using 3D controllers.
- More options (. . .)
  - **Use depth peeling** must be enabled for correct rendering of semi-transparent surfaces (in models, markups, etc). It may make rendering updates slightly slower and artifacts when volume rendering is used in the view.
  - **Show/Hide frames per second (FPS)** displays rendering speed in the corner of the view.
- **Orientation Marker** controls display of human, cube, etc in lower right corner.
- **Visibility options** controls visibility of view background color and displayed components.

- **Spin** continuously spins the view around.
- **Rock** continuously rocks the view left-to-right.
- **Zoom in/out** slightly zooms in/out the view. Convenient buttons for touchscreens.
- **Tilt Lock** can be toggled using `Ctrl + b` keyboard shortcut. In tilt lock mode 3D view rotation is restricted to the azimuth axis (left-right direction) by disabling rotation around elevation axis (up-down direction).

Ambient shadows

- Shadows visibility: if enabled then ambient shadows are displayed to improve depth perception
  - Size scale (default: 0): Size of features to be emphasized by shadows. Lower values emphasize surface unevenness (bumps and depressions). Higher size scale makes larger regions that are much farther much darker. The optimal value can be determined by adjusting the value starting from the highest value gradually until regions that are behind objects appear darker. Very high values (1.5 to 3.0) may make the entire image appear darker, therefore it is important to go below this range, typically down to about 0.5 to 1.5.
  - Intensity scale: (default: 1): Intensity of darkening by shadows. Increase the value to make shadows darker.
  - Intensity shift: (default: 0): Minimum amount of occlusion required for visible darkening by shadows. It is usually only necessary to increase this value if the intensity scale is increased, which makes the entire image somewhat darker. In this case, increasing the intensity shift can compensate the overall darkening.
  - Volume opacity threshold (0% to 100%; default: 25%): This value only affects volume rendering. Voxels that have opacity above this value will cast shadows. Choosing too low value results in dark artifacts in regions that have very low opacity. Choosing too high value results artifacts near opaque regions that have somewhat lower opacity value.

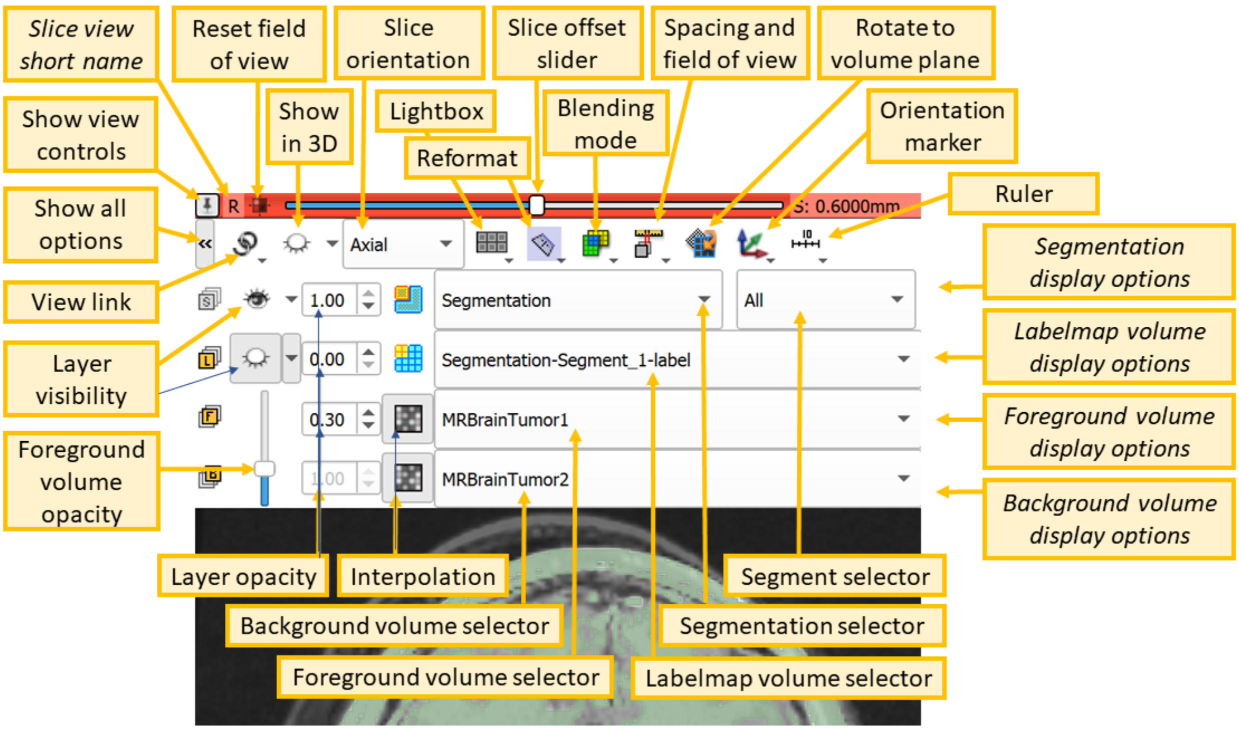

Size
scale
-1.2
0.0 (default)
1.3
Intensity
scale
0.5
1.0 (default)
2.0
Intensity
shift
0.0 (default)
0.25
0.5
Volume
opacity
threshold
5%
25% (default)
85%

Examples:

> **Note**
>
> Default ambient shadows settings can be chosen in the application menu (Edit / Application Settings / Views / 3D viewer defaults).

4.3.4 Slice View

Three default slice views are provided (with Red, Yellow and Green colored bars) in which Axial, Sagittal, Coronal or Oblique 2D slices of volume images can be displayed. Additional generic slice views have a grey colored bar and an identifying number in their upper left corner.



Slice View Controls: The colored bar across any slice view shows a pushpin icon on its left (**Show view controls**). 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 (**Show all options**).

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
- **Show in 3D** "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.
- **Slice orientation** displays allows you to choose the orientation for this slice view.
- **Reformat** allows interactive manipulation of the slice orientation.
- **Slice offset slider** allows slicing through the volume. Step size is set to the background volume's spacing by default but can be modified by clicking on "Spacing and field of view" button. The label next to the offset value (e.g., `S`, `L`, `A`, `IL`, `IRP`) reflects the slice normal direction. If the offset slider moved to the right then the slice

moves in this normal direction. If the slice normal direction is not aligned with an axis then the label contains a combination of directions, with the order of axes reflecting the dominance of the axis. For example, if the plane normal points to anterior and slightly left then the label is `AL`, while if the plane normal mostly left and slightly anterior then the label is `LA`.

- **Blending mode** specifies 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.
- **Orientation Marker** controls display of human, cube, etc in lower right corner.
- **Ruler** controls display of ruler in slice view.
- **View link** button synchronizes properties of views in the same view group, such as foreground/background/label volume selection, foreground/label volume opacity, zoom factor.
  - For parallel views (i.e., that are set to the same orientation,OD such as `axial`), the view center position is synchronized as well.
  - Long-click on the button exposes **hot-linked** option, which controls when properties are synchronized (immediately or when the mouse button is released).
  - A view group typically consists of 3 orthogonal views (e.g., in `Four-Up` view, `R`, `G`, `Y, views are in the same group). In layouts that contain multiple triplets of slice views, each triplet forms a separate group (e.g., in` Three over three`layout there are two view groups, one group is`R, G, Y, the other groups is R+, G+, Y+`).
- **Layer visibility** "eye" buttons and **Layer opacity** spinboxes control visibility of segmentations and volumes in the slice view.
- **Foreground volume opacity** slider allows fading between foreground and background volumes.
- **Interpolation** allows displaying voxel values without interpolation. Recommended to keep interpolation enabled, and only disable it for testing and troubleshooting.
- **Node selectors** are used to choose which background, foreground, and labelmap volumes and segmentations to display in this slice view. Note: multiple segmentations can be displayed in a slice view, but slice view controls only allow adjusting visibility of the currently selected segmentation node.

Removed in version 5.10:

- **Lightbox** feature to select a mosaic (a.k.a. contact sheet) view was removed. Instead the *Lightbox image columns* option provided by the *Screen Capture* module may be used.

4.4 Mouse & Keyboard Shortcuts

4.4.1 Alternate macOS Keybindings

When using shortcuts on macOS, replace the following keys with their corresponding macOS version.

4.4.2 Generic shortcuts

4.4.3 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.

4.4.4 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`

4.4.5 Python console

The following shortcuts are available in the Python console.

Note that when code is pasted into an empty line then all the code in the clipboard is executed *at once*. If the current command line is not empty then the code from the clipboard is pasted into the console and executed *line by line*. When code is executed line by line, the behavior is different in that an empty input line immediately closes the current block, and output is printed after executing each line.

CHAPTER

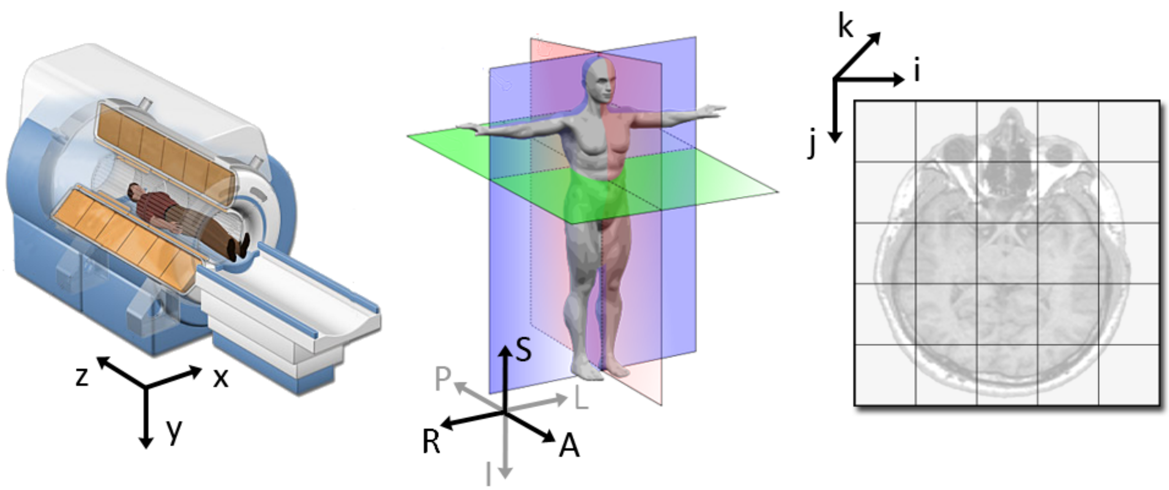
FIVE

COORDINATE SYSTEMS

5.1 Introduction

One of the issues while dealing with medical images and applications are the differences between the coordinate systems. There are three coordinate systems commonly used in imaging applications: a difference can be made between the **world**, **anatomical** and the **image coordinate system**.

The following figure illustrates the three spaces and their corresponding axes:



Each coordinate system serves one purpose and represents its data in a particular way.

Anatomy image based on an image shared by the My MS organization.

Note that Chand John of Stanford created a detailed presentation about the way coordinates are handled in Slicer.

5.1.1 World coordinate system

The world coordinate system is typically a Cartesian coordinate system in which a model (e.g. a MRI scanner or a patient) is positioned. Every model has its own coordinate system but there is only one world coordinate system to define the position and orientation of each model.

5.1.2 Anatomical coordinate system

The most important model coordinate system for medical imaging techniques is the anatomical space (also called patient coordinate system). This space consists of three planes to describe the standard anatomical position of a human:

- the *axial plane* is parallel to the ground and separates the head (Superior) from the feet (Inferior).

- the *coronal plane* is perpendicular to the ground and separates the front (Anterior) from the back (Posterior).
- the *sagittal plane* is perpendicular to the ground and separates the Left from the Right.

From these planes it follows that all axes have their notation in a positive direction (e.g. the negative Superior axis is represented by the Inferior axis).

The anatomical coordinate system is a continuous three-dimensional space in which an image has been sampled. In neuroimaging, it is common to define this space with respect to the human whose brain is being scanned. Hence, the 3D basis is defined along the anatomical axes of anterior-posterior, inferior-superior, and left-right.

However different medical applications use different definitions of this 3D basis. Most common are the following bases:

- LPS (Left, Posterior, Superior) is used in DICOM images

$$LPS = \left\{ \begin{matrix} \text{from right towards left} \\ \text{from anterior towards posterior} \\ \text{from inferior towards superior} \end{matrix} \right\}$$

- RAS (Right, Anterior, Superior) is similar to LPS with the first two axes flipped

$$RAS = \left\{ \begin{matrix} \text{from left towards right} \\ \text{from posterior towards anterior} \\ \text{from inferior towards superior} \end{matrix} \right\}$$

Thus, the only difference between the two conventions is that the sign of the first two coordinates is inverted.

Both bases are equally useful and logical. It is just necessary to know to which basis an image is referenced.

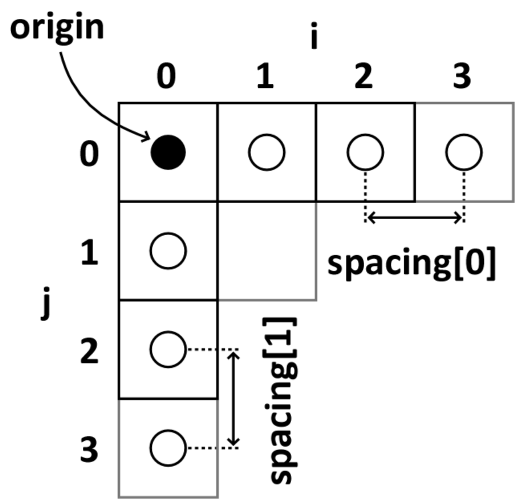
5.1.3 Image coordinate system

The image coordinate system describes how an image was acquired with respect to the anatomy. Medical scanners create regular, rectangular arrays of points and cells which start at the upper left corner. The i axis increases to the right, the j axis to the bottom and the k axis backwards.

In addition to the intensity value of each voxel (ijk) the origin and spacing of the anatomical coordinates are stored too.

- The origin represents the position of the first voxel (0, 0, 0) in the anatomical coordinate system, e.g. (100, 50, -25) mm.
- The spacing specifies the distance between voxels along each axis, e.g. (1.5, 0.5, 0.5) mm.

The following 2D example shows the meaning of origin and spacing:



Using the origin and spacing, the corresponding position of each (image coordinate) voxel in anatomical coordinates can be calculated.

5.2 Image transformation

The transformation from an image space vector $(ijk)'$ to an anatomical space vector $\vec{x}$ is an affine transformation, consists of a linear transformation $\mathbf{A}$ followed by a translation $\vec{t}$.

$$\vec{x} = A \begin{pmatrix} i & j & k \end{pmatrix}' + \vec{t}$$

The transformation matrix $\mathbf{A}$ is a 3×3 matrix and carries all information about space directions and axis scaling.

$\vec{t}$ is a 3×1 vector and contains information about the geometric position of the first voxel.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix}$$

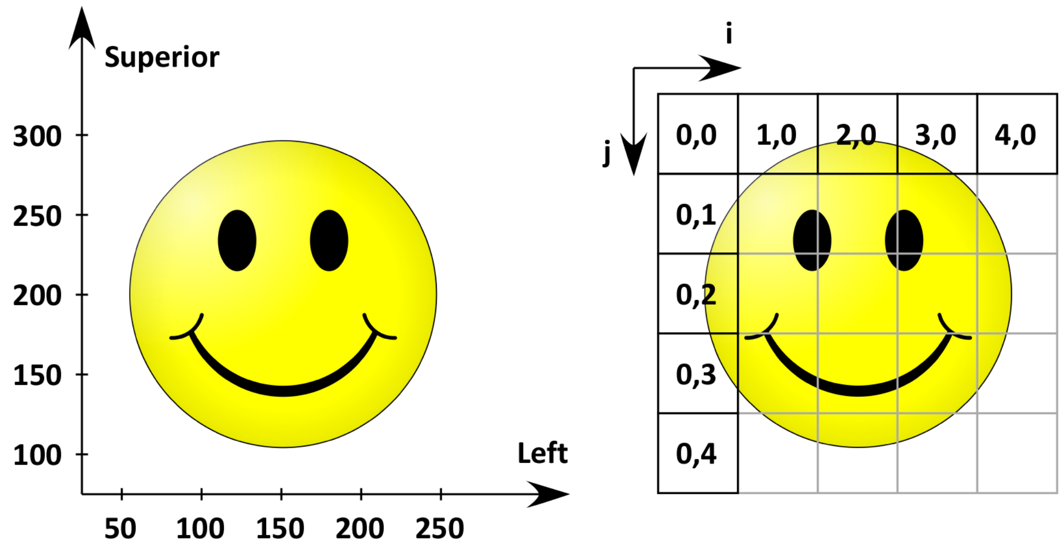
The last equation shows that the linear transformation is performed by a matrix multiplication and the translation by a vector addition. To represent both, the transformation and the translation, by a matrix multiplication an augmented matrix must be used. This technique requires that the matrix $\mathbf{A}$ is augmented with an extra row of zeros at the bottom, an extra column-the translation vector-to the right, and a 1 in the lower right corner. Additionally, all vectors have to be written as homogeneous coordinates, which means that a 1 is augmented at the end.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & t_1 \\ A_{21} & A_{22} & A_{23} & t_2 \\ A_{31} & A_{32} & A_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ k \\ 1 \end{pmatrix}$$

Depending on the used anatomical space (LPS or RAS) the 4×4 matrix is called **IJKtoLPS**- or **IJKtoRAS**-matrix, because it represents the transformation from IJK to LPS or RAS.

5.3 2D example or calculating an *IJtoLS*-matrix

The following figure shows the anatomical space with an L(P)S basis on the left and the corresponding image coordinates on the right.



The origin (the coordinates of the first pixel in anatomical space) is (50, 300) mm and the spacing (the distance between two pixels) is (50, 50) mm.

As this is a 2D example $\mathbf{A}$ is a 2×2 matrix and $\vec{t}$ a 2×1 vector. Therefore, the equation of the affine transformation is:

$$\begin{pmatrix} L \\ S \\ 1 \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & t_1 \\ A_{21} & A_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \\ 1 \end{pmatrix}$$

By multiplying the **IJtoLS**-matrix and the vector of the right side, the following product will be obtained:

$$\begin{array}{c|c} & \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \\ \hline \begin{pmatrix} A_{11} & A_{12} & t_1 \\ A_{21} & A_{22} & t_2 \\ 0 & 0 & 1 \end{pmatrix} & \begin{pmatrix} A_{11} \cdot i + A_{12} \cdot j + t_1 \cdot 1 \\ A_{21} \cdot i + A_{22} \cdot j + t_2 \cdot 1 \\ 0 \cdot i + 0 \cdot j + 1 \cdot 1 \end{pmatrix} \end{array}$$

The last equation and the matrix product show that a total of 6 unknown variables $(A_{11}, A_{12}, A_{21}, A_{22}, t_1, t_2)$ have to be determined. The knowledge of origin and spacing however allows the following relations between image and anatomical space:

$$\begin{pmatrix} L \\ S \end{pmatrix} \equiv \begin{pmatrix} i \\ j \end{pmatrix} \qquad \begin{pmatrix} 50 \\ 300 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 100 \\ 300 \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad \begin{pmatrix} 50 \\ 250 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad \dots$$

Thus, at least six equations can be derived:

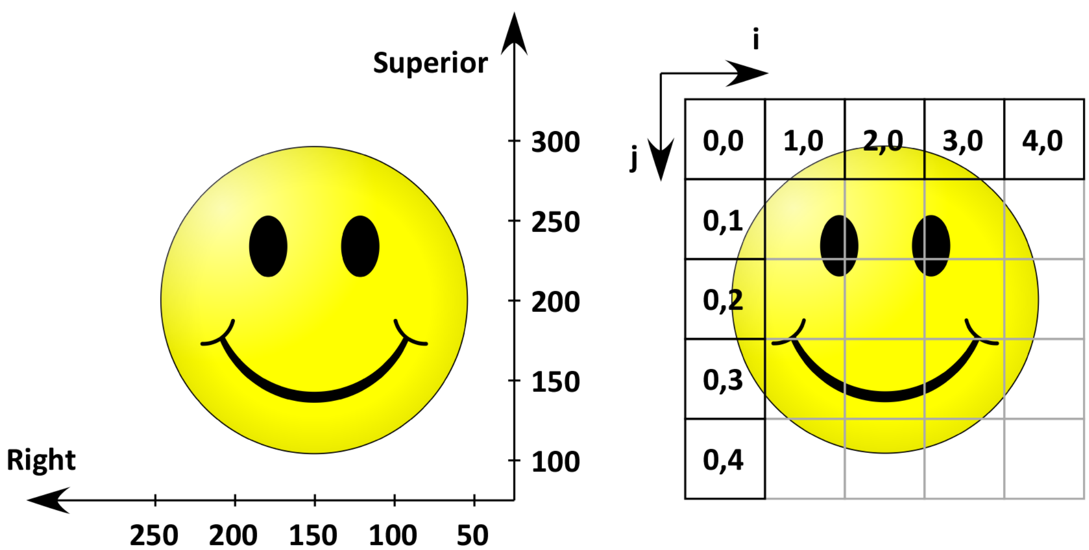
$$\begin{aligned}
50 &= A_{11} \cdot 0 + A_{12} \cdot 0 + t_1 \cdot 1 \\
300 &= A_{21} \cdot 0 + A_{22} \cdot 0 + t_2 \cdot 1 \\
100 &= A_{11} \cdot 1 + A_{12} \cdot 0 + t_1 \cdot 1 \\
300 &= A_{21} \cdot 1 + A_{22} \cdot 0 + t_2 \cdot 1 \\
50 &= A_{11} \cdot 0 + A_{12} \cdot 1 + t_1 \cdot 1 \\
250 &= A_{21} \cdot 0 + A_{22} \cdot 1 + t_2 \cdot 1
\end{aligned}$$

As mentioned above, the translation $\vec{t}$ contains the information about the geometric position of the first pixel and is therefore equivalent to the origin. This result is also confirmed by the first equations.

The solution of the other equations leads to the following **IJtoLS**-matrix:

$$IJtoLS = \begin{pmatrix} 50 & 0 & 50 \\ 0 & -50 & 300 \\ 0 & 0 & 1 \end{pmatrix}$$

In the event that a R(A)S basis was used, just the left and anterior axis of the anatomical space are flipped, and the image coordinate system appears in the same way as in the L(P)S case.



For this 2D example the **IJtoRS**-matrix would be:

$$IJtoRS = \begin{pmatrix} -50 & 0 & 250 \\ 0 & -50 & 300 \\ 0 & 0 & 1 \end{pmatrix}$$

This matrix looks very similar to the **IJtoLS**-matrix with 2 differences:

- The translation $\vec{t}$ has changed because of another origin.
- The right axis is flipped, so the first column of the **IJtoRS**-matrix has just an inverted sign.

5.4 Coordinate system convention in Slicer

DICOM and most medical imaging software use the **LPS coordinate system** for storing all data. The choice of origin is arbitrary because only relative differences have meaning, so there is no universal standard, but it is often set to some geometric center of the imaging system, or it is chosen to be near the center of an object of interest.

Both LPS and RAS were in wide use in the early 2000s when development of Slicer was started and Slicer developers chose to use the RAS coordinate system. Historically scans by GE equipment used RAS while Siemens and others used LPS. Since several GE researchers were early contributors to Slicer, RAS was adopted for the internal representation.

Slicer still **uses RAS coordinate system for storing coordinate values internally** for all data types, but for compatibility with other software, it **assumes that all data in files are stored in LPS coordinate system** (unless the coordinate system in the file is explicitly stated to be RAS). To achieve this, whenever Slicer reads or writes a file, it may need to flip the sign of the first two coordinate axes to convert the data to RAS coordinate system.

5.4.1 Relations to other software/conventions

ITK

ITK uses the LPS convention.

Using MATLAB to map Slicer RAS coordinates (e.g. fiducials) to voxel space of a NIfTI Image

To extract the "voxel to world" transformation matrix from a NIFTI file's header (entry: `qto_xyz:1-4`) in MATLAB:

```
d = inv(M) * [ R A S 1 ]'
```

where `M` is the matrix and `R A S` are coordinates in Slicer, then `d` gives a vector of voxel coordinates.

(Solution courtesy of András Jakab, University of Debrecen)

5.5 References

- https://people.cs.uchicago.edu/~glk/unlinked/nrrd-iomf.pdf
- http://www.grahamwideman.com/gw/brain/orientation/orientterms.htm
- https://nifti.nimh.nih.gov/nifti-1/documentation/faq
- https://teem.sourceforge.net/nrrd/format.html
- DICOM 2013 PS3.3 Image Position and Image Orientation

CHAPTER

SIX

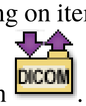
DATA LOADING AND SAVING

There are two major types of data that can be loaded to Slicer: DICOM and non-DICOM.

6.1 DICOM data

DICOM is a widely used and sophisticated set of standards for digital radiology.

Data can be loaded from DICOM files into the scene in two steps:

1. Import: add files into the application's DICOM database, by switching to DICOM module and drag-and-dropping files to the application window
2. Load: get data objects into the scene, by double-clicking on items in the DICOM browser. The DICOM browser is accessible from the toolbar using the DICOM button 

.

Data in the scene can be saved to DICOM files in two steps:

1. Export to database: save data from the scene into the application's DICOM database
2. Export to file system: copy DICOM files from the database to a chosen folder in the file system

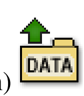
More details are provided in the *DICOM module documentation*.

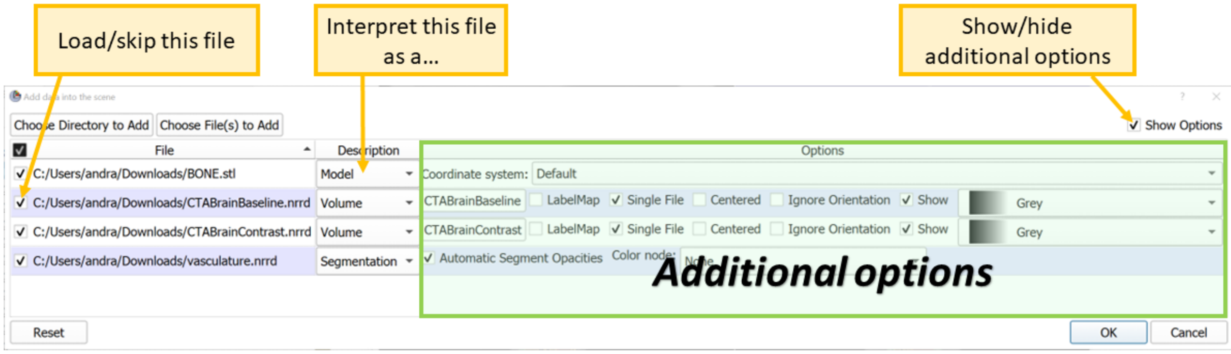
6.2 Non-DICOM data

Non-DICOM data, covering all types of data ranging from images (nrrd, nii.gz, …) and models (stl, ply, obj, …) to tables (csv, txt), point lists (json), etc.

6.2.1 Load data

To load data:

- drag&drop file on the application window, or
- in application menu: `File -> Add Data` (or `Add Data` toolbar button) 

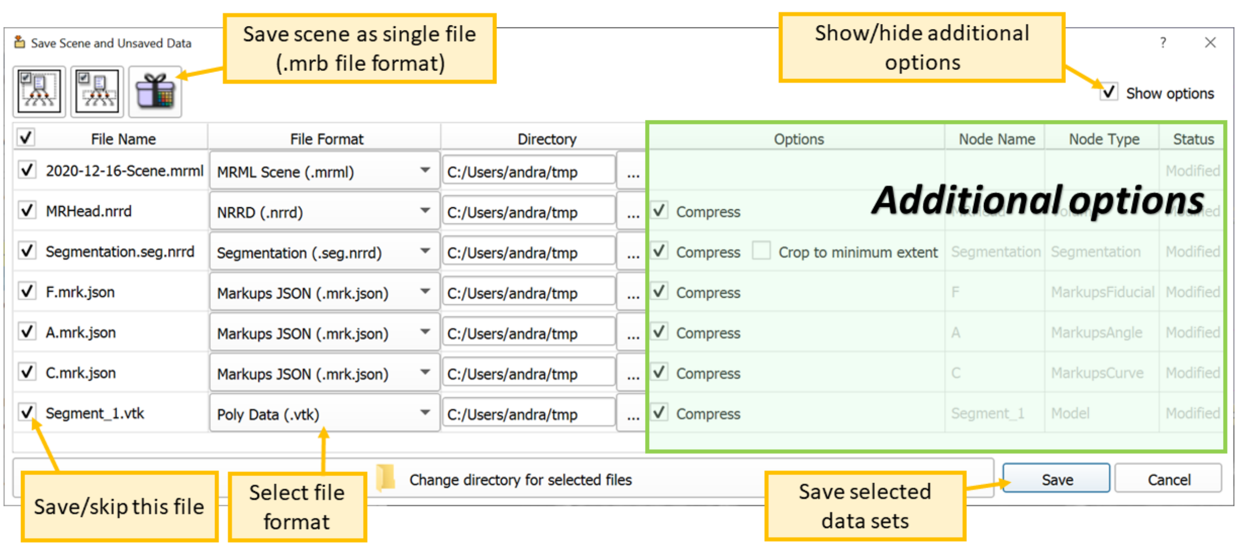


6.2.2 Save data

To save the entire scene (all data, visualization and processing settings, etc.):

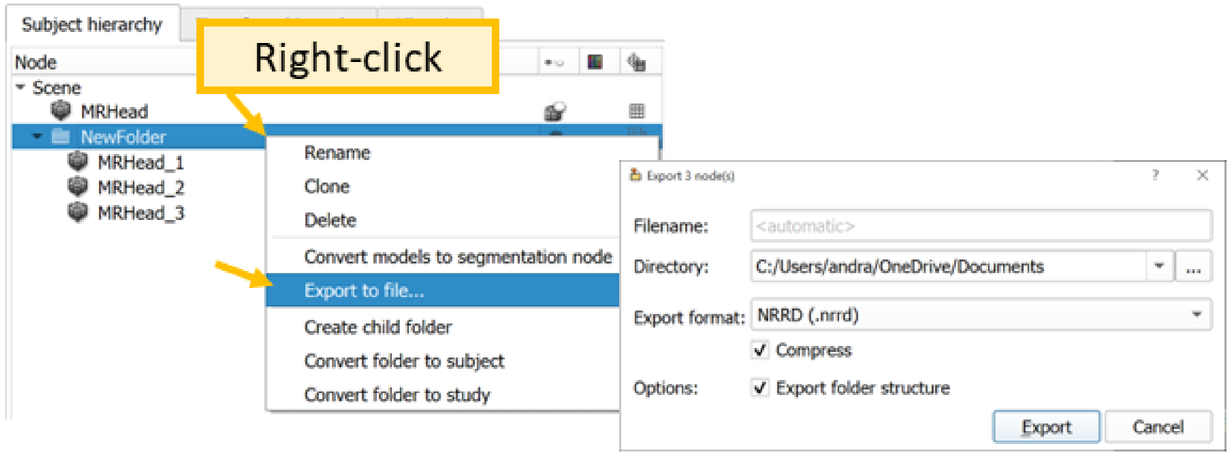
- in application menu `File -> Save Data`, or
- `Save Data` toolbar button 

> **Tip**
>
> The entire workspace, including all data and settings can be saved into a single, independent, self-contained `.mrb` file by clicking on the small package icon at the top-left corner. A new copy of all files is written and zipped into a single file, therefore saving takes longer time than an incremental saving of only the modified files.



6.2.3 Export data

To export selected data sets - for sharing with others or for loading into other applications: go to `Data` module, right-click on an item, and choose `Export to file...`. Settings used for exporting (file format, filename, options) do not modify settings used for saving.



Tip

Multiple nodes can be exported at once by placing them into a folder and then by exporting the folder. When exporting an entire folder hierarchy the `Export folder structure` option can be enabled to have the directory structure in the output directory match the subject hierarchy folder structure.

6.3 Supported Data Formats

Note

On use of LPS/RAS coordinate systems

DICOM and medical imaging software use the LPS (Left, Posterior, Superior) coordinate system, while Slicer's internal representation employs RAS (Right, Anterior, Superior). For file compatibility, Slicer assumes data in files are in LPS coordinates and may flip the first two axes during read or write operations.

To learn more, see the *Coordinate systems* documentation, and the *Coordinate system convention in Slicer*.

Note

Be careful with uncommon encoding options

Research data formats like NRRD and NIfTI, and even widely used clinical formats like DICOM, support many encoding options that are technically valid but are so rarely seen that support libraries and other code in Slicer may not interpret them correctly.

If you are performing data interchange between Slicer and other software, prefer widely used conventions (such as LPS or RAS for coordinate systems) and double-check data integrity if you deviate from them.

6.3.1 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
      - DICOM Segmentation objects (SEG modality)
      - Parametric Maps
      - Encapsulated STL (M3D modality)
      - Certain flavors of Structured Reports that follow TID 1500 Measurement report template
      - Secondary capture with Slicer scene (MRB) in private tag
    - 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
  - To load an image file as segmentation (also known as label image, mask, region of interest) see *Segmentations module documentation*

- **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 strongle 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).
  - To load an image file as segmentation (also known as label image, mask, region of interest) see *Segmentations module documentation*
- **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** (.lsm)
- **Scanco** (.isq)
- **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; reading only)
  - **Philips 4D ultrasound**: from Cartesian DICOM exported from QLab
  - **GE Kretz 3D ultrasound** (.vol, .v01)
  - **Eigen Artemis 3D ultrasound**
  - Any 3D/4D ultrasound image and ECG signal: if the user obtains Image3dAPI plugin from the vendor (GE Voluson, Philips, Siemens, etc.)
- 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
- Sandbox extension:
  - **Topcon OCT image file** (.fda, reading only)

6.3.2 Models

Surface or volumetric meshes.

- **VTK Polygonal Data** (.vtk, .vtp): Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header. Full color (RGB or RGBA) meshes can be read and written (color must be assigned as point scalar data of `unsigned char` type and 3 or 4 components). Texture image can be applied using "Texture model" module (in SlicerIGT extension).
- **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. Texture image can be applied using "Texture model" module (in SlicerIGT extension). The non-standard technique of saving vertex color as additional values after coordinates is not supported - if vertex coloring is needed then convert to PLY, VTK, or VTP format using another software.
- **Stanford Triangle Format** (.ply): Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in header. Full color (RGB or RGBA) meshes can be read and written (color must be assigned to vertex data in `uchar` type properties named `red`, `green`, `blue`, and optional `alpha`). Texture image can be applied using "Texture model" module (in SlicerIGT extension).
- **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; reading only)
- SlicerHeart extension:
  - **CARTO surface model** (.vtk; writing only): special .vtk polydata file format variant, which contains patient name and ID to allow import into CARTO cardiac electrophysiology mapping systems

6.3.3 Segmentations

- **Segmentation labelmap representation** (.seg.nrrd, .nrrd, .seg.nhdr, .nhdr, .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, .nhdr, .nii, .nii.gz, .hdr): segment names can be defined by using a color table. To write segmentation in NIFTI formats, use Export to file feature or export the segmentation node 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)
- Sandbox extension:
  - **Osirix ROI file** (.json, reading only)
  - **sliceOmatic tag file** (.tag, reading only)

6.3.4 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**: For storing grid transform as a vector image, each voxel containing displacement vector. Voxel values are in physical space (not in voxels).
  - NRRD (.nrrd, .nhdr): Coordinate system: LPS. Metadata: `dimension: 4`, `sizes: 3 I J K`, `space directions: none (ix, iy, iz) (jx, jy, jz) (kx, ky, kz)`, `kinds: vector domain domain domain`
  - MetaImage (.mha, .mhd): Coordinate system: LPS. Metadata: `NDims = 3`, `DimSize = I J K`, `ElementNumberOfChannels = 3`
  - NIFTI (.nii, .nii.gz). Coordinate system: RAS. Metadata: Number of dimensions: 5. Dimensions: `[I, J, K, 1, 3]`. Intent code: `NIFTI_INTENT_DISPVECT (1006)`.
- SlicerRT extension
  - **Pinnacle DVF** (.dvf)

6.3.5 Markups

- **Markups JSON** (.mrk.json): point list, line, curve, closed curve, plane, etc. Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in image header. JSON schema is available here.
- **Markups CSV** (.fcsv): legacy file format for storing point list. Default coordinate system: LPS. Coordinate system (LPS/RAS) can be specified in image header. This format only stores control point coordinates and a limited set of display properties; use Markups JSON format instead. The file format is deprecated and support for it will be removed in future software versions.
- **Annotation CSV** (.acsv): legacy file format for storing markups line, ROI.

6.3.6 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)

6.3.7 Other

- **Text** (.txt, .xml., json)
- **Table** (.csv, .tsv)
- Color tables:
  - **Slicer color table** (.ctbl, .txt)
  - ITK-Snap label description file (.txt, .label) (reading only) This can be used for loading segmentations that were created in ITK-Snap. The color table must be loaded in the scene first. Then, when the label image file is loaded then in `Add Data` window select `Segmentation` in the `Description` column and select the loaded color table in the `Color node` column.
- ***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)

6.3.8 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.

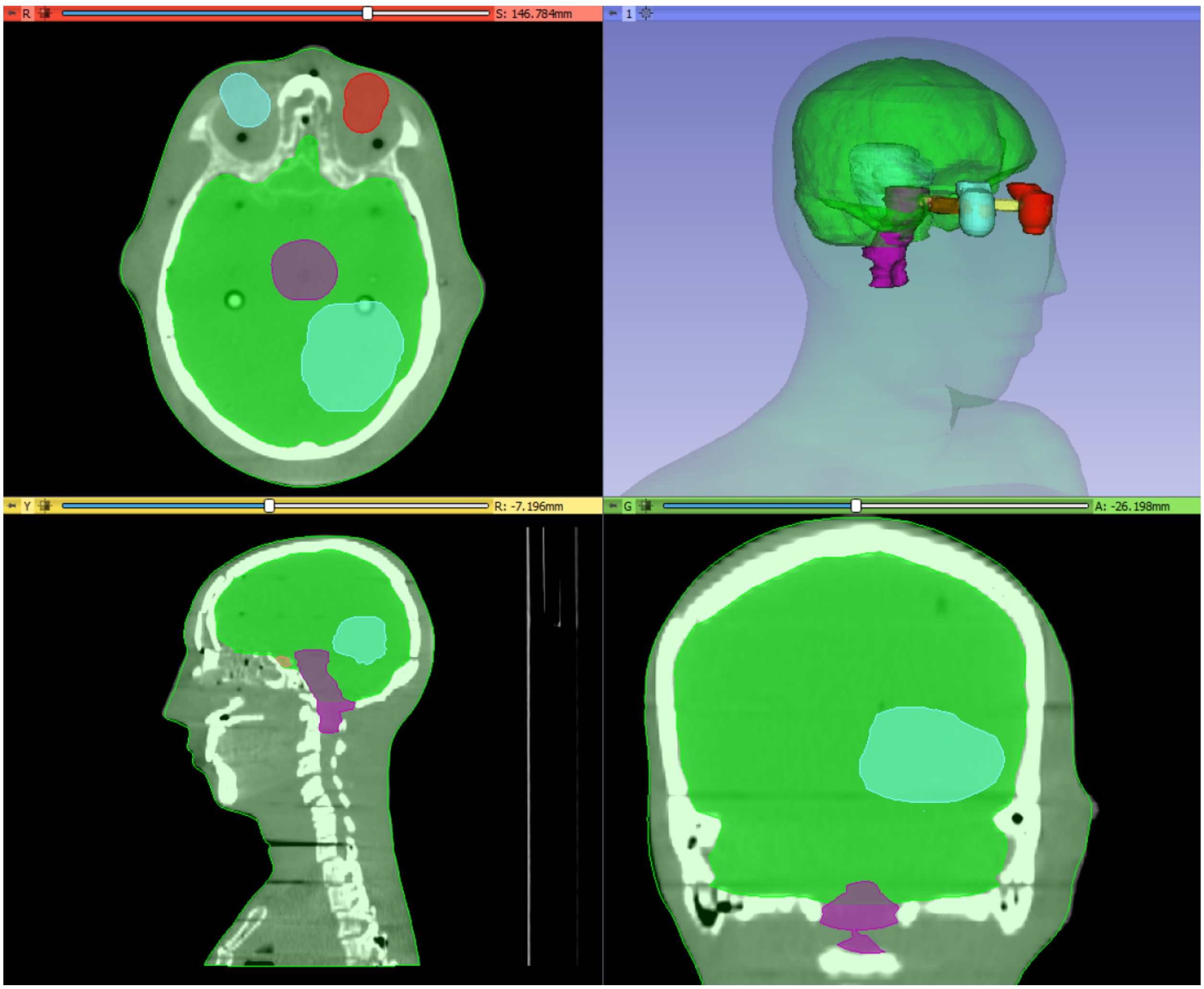
If you have a file of binary data and you know the data is uncompressed and you know the way it is laid out in memory, then one way to load it in Slicer is to create a .nhdr file that points to the binary file. RawImageGuess extension can be used to explore an unknown data set, determining unknown loading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

CHAPTER

SEVEN

IMAGE SEGMENTATION



7.1 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.

7.1.1 Segmentation and segment

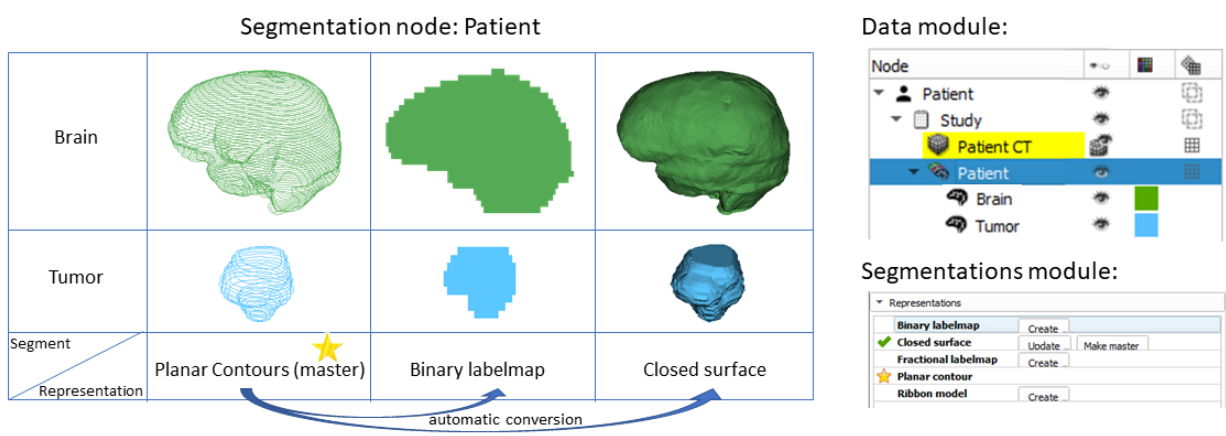
Result of a segmentation is stored in `segmentation` node in 3D Slicer. A segmentation node consists of multiple segments.

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.

7.1.2 Representations

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 `source representation` (marked with a "gold star" on the user interface). The source 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.



7.1.3 Binary labelmap representation

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.

In a segmentation with its source representation set to binary labelmap, each layer is allowed to have different geometry (origin, spacing, axis directions, extents) temporarily - to allow moving segments between segmentations without unnecessary quality loss (each resampling of a binary labelmap can lead to slight changes). All layers are forced to have the same geometry during certain editing operations and when the segmentation is saved to file.

7.2 Segmentation modules

There are many modules in 3D Slicer for manipulating segmentations. Overview of the most important is provided below.

Segmentation modules in Slicer core:

- *Segmentations*: Adjust display options, manage segment representations and layers, copy/move segments between segmentations, convert between segmentation and models and labelmap volumes, export to files.
- *Segment Editor*: Create and edit segmentations from images using manual (paint, draw, …), semi-automatic (thresholding, region growing, interpolation, …) and automatic tools. A number of editor effects are built into the Segment Editor module and many more are provided by extensions (in "Segmentations" category in the Extensions Manager).
- *Segment statistics*: Computes intensity and geometric properties for each segment, such as volume, surface, minimum/maximum/mean intensity, oriented bounding box, sphericity, etc. See more information in.

Extensions for creating/editing segmentations:

- SegmentEditorExtraEffects: Adds 8 more effects to Segment Editor.
- SurfaceWrapSolidify: fill in internal holes in a segmented image region or retrieve the largest cavity inside a segmentation.
- MONAILabel: AI-based segmentation of various organs using MONAILabel.
- TotalSegmentator: AI-based fully automatic segmentation of 104 structures in whole-body CT images.
- DensityLungSegmentation: AI-based fully automatic lung segmentation.
- HDBrainExtraction: AI-based fully automatic skull stripping in brain MRI images.
- NVIDIA-AIAA: AI-based fully automatic segmentation of several organs. Segmentation is performed on a remote server.
- RVesselX: Semi-automatic liver parenchyma and vessels segmentation.

Extensions for analyzing and processing segmentations:

- Segment comparison: Compute similarity between two segments based on metrics such as Hausdorff distance and Dice coefficient.
- Segment registration (provided by SegmentRegistration extension): Compute rigid or deformable transform that aligns two selected segments.
- SegmentMesher: Creating volumetric (tetrahedral) meshes from segmentations.
- OpenAnatomy: Export segmentations or model hierarchies for external viewers in glTF or OBJ format.
- Sandbox: provides importer for Osirix ROI and SliceOmatic segmentation files.

For more extensions related to segmentations, open the "Segmentations" category in the Extensions Catalog.

7.3 Tutorials

To get started, check out these pages:

- Segmentation tutorials
- *Segment Editor module documentation*: detailed description of Segment Editor user interface and effects

CHAPTER

EIGHT

REGISTRATION

Goal of registration is to align position and orientation of images, models, and other objects in 3D space. 3D Slicer offers many registration tools, this page only lists those that are most commonly used.

8.1 Manual registration

Any data nodes (images, models, markups, etc.) can be placed under a transform and the transform can be adjusted interactively in *Transforms* module (using sliders) or in 3D views.

Advantage of this approach is that it is simple, applicable to any data type, and approximate alignment can be reached very quickly. However, achieving accurate registration using this approach is tedious and time-consuming, because many fine adjustments steps are needed, with visual checks in multiple orientations after each adjustment.

8.2 Semi-automatic registration

Registration can be computed automatically from corresponding landmark point pairs specified on the two objects. Typially 6-8 points are enough for a robust and accurate rigid registration.

Recommended modules:

- Landmark registration: for registering slightly misaligned images. Supports rigid and deformable registration with automatic local landmark refinement, live preview, image comparison.
- Fiducial registration wizard (in SlicerIGT extension): for registering any data nodes (even mixed data, such as registration of images to models), and for images that are not aligned at all. Supports rigid and deformable registration, automatic point matching, automatic collection from tracked pointer devices. See U-12 SlicerIGT tutorial for a quick introduction of main features.

8.3 Automatic image registration

Grayscale images can be automatically aligned to each other using intensity-based registration methods. If an image does not show up in the input image selector then most likely it is a color image, which can be converted to grayscale using *Vector to scalar volume* module.

Intensity-based image registration methods require reasonable initial alignment, typically less than a few centimeter translation and less than 10-20 degrees rotation error. Some registration methods can perform initial position alignment (e.g., using center of gravity) and orientation alignment (e.g., matching moments). If automatic alignment is not robust then manual or semi-automatic registration methods can be used as a first step.

It is highly recommended to crop the input images to cover approximately the same anatomical region. This allows faster and much more robust registration. Images can be cropped using Crop volume module.

Recommended modules:

- General registration (Elastix) (in SlicerElastix extension): Its default registration presets work without the need for any parameter adjustments.
- General Registration (ANTs) (in SlicerANTS extension): Similarly to Elastix, default parameter set should work well for most image registration tasks. The module also exposes many registration parameters that users can tweak.
- General registration (BRAINS): recommended for brain MRIs but with parameter tuning it can work on any other imaging modalities and anatomical regions.
- Sequence registration: Automatic 4D image (3D image time sequence) registration using Elastix. Can be used for tracking position and shape changes of structures in time, or for motion compensation (register all time points to a selected time point).

8.4 Segmentation and binary image registration

Registration of segmentation and binary images are very different from grayscale images, as only the boundaries can guide the alignment process. Therefore, general image registration methods are not applicable to binary images.

Recommended module:

- Segment registration (in SegmentRegistration extension): registers a selected pair of segments fully automatically. Supports rigid, affine, and deformable registration. Binary images can be registered by converting to segmentation nodes first.

8.5 Model registration

During registration of models containing surface meshes, only the boundaries can guide the alignment process.

Manual and semi-automatic registration methods described above are applicable to model registration. The following modules are recommended for automatic registration:

- Segment registration (in SegmentRegistration extension): this module can be used after importing a model to a segmentation node. See details in the section above.
- Model registration (in SlicerIGT extension): uses iterative closest points. this method tends to get stuck in a local optimum, therefore it is important to start it from a good initial position (e.g., computed using manual or semi-automatic registration).
- FastModelAlign automatic surface registration method in SlicerMorph extension: more robust (can converge from farther initial registration error, has higher chance of finding global optimum) than iterative closest point based algorithms. Allows rigid (with or without scaling) and affine registration of models.

8.6 More information

Over the years, vast amount of information was collected about image registration, which are not kept fully up-to-date, but still offer useful insights.

- Registration Library: list of example cases with data sets and steps to achieve the same result.
- Registration FAQ: frequently asked questions related to registration and resampling
- Former registration main page: not fully up-to-date, but still useful information about registration

CHAPTER

NINE

MODULES

Main modules

9.1 Data

9.1.1 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

9.1.2 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

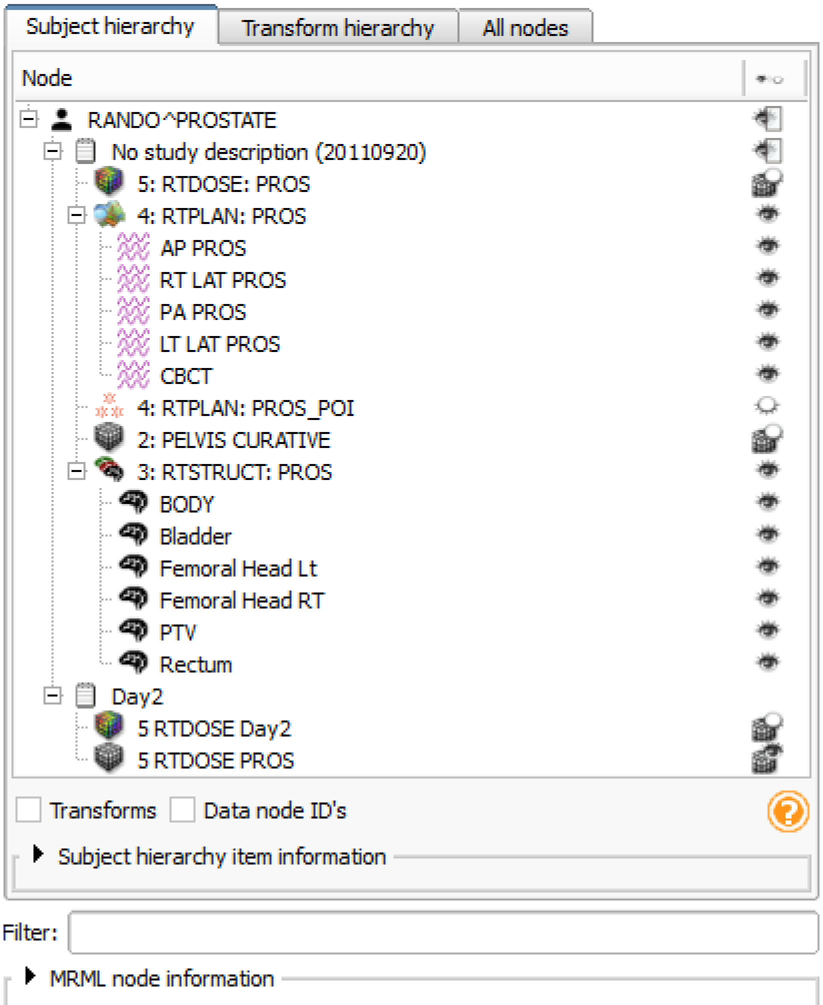
Apply transform on node or branch

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

9.1.3 Panels and their use

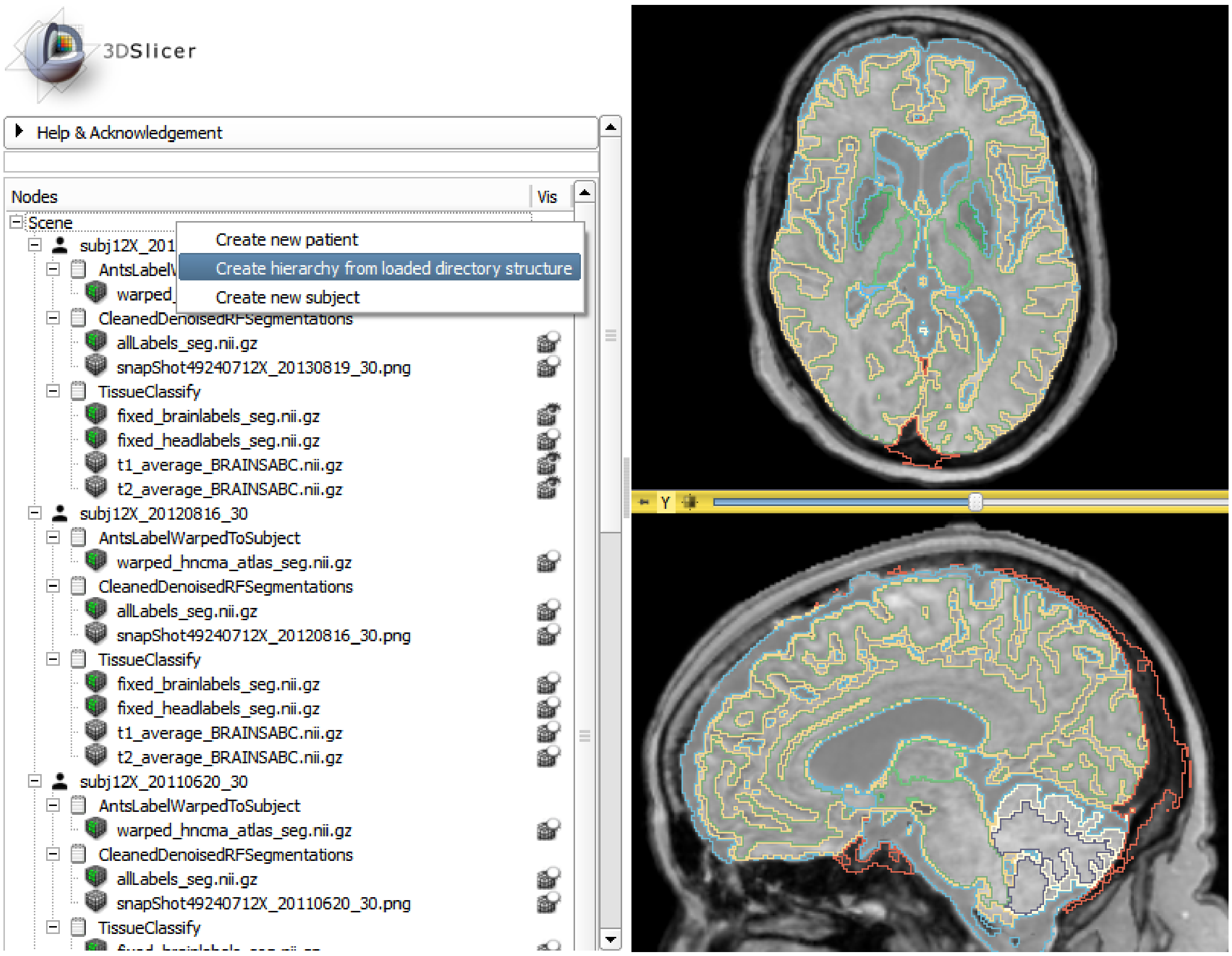
Subject hierarchy tab

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 `Scene` item at the top
- Data loaded from **DICOM** are automatically added to the tree of 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
  - Legacy model and annotation hierarchies from old scenes are imported as 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 Segment 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

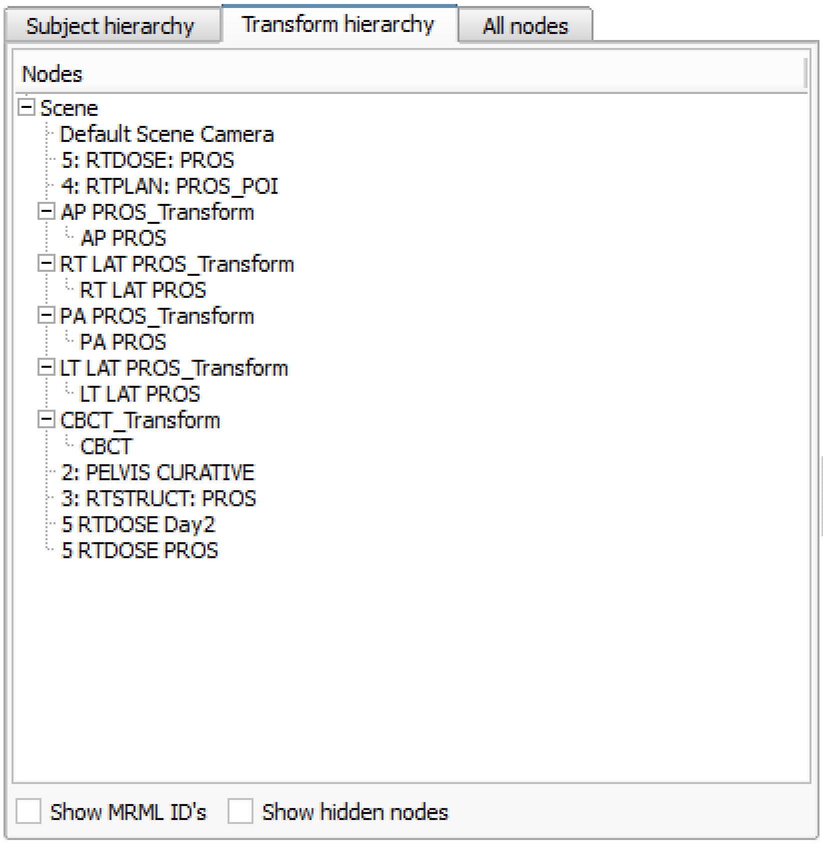
Highlights: when an item is selected, the related items are highlighted. Meaning of colors:

- 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.

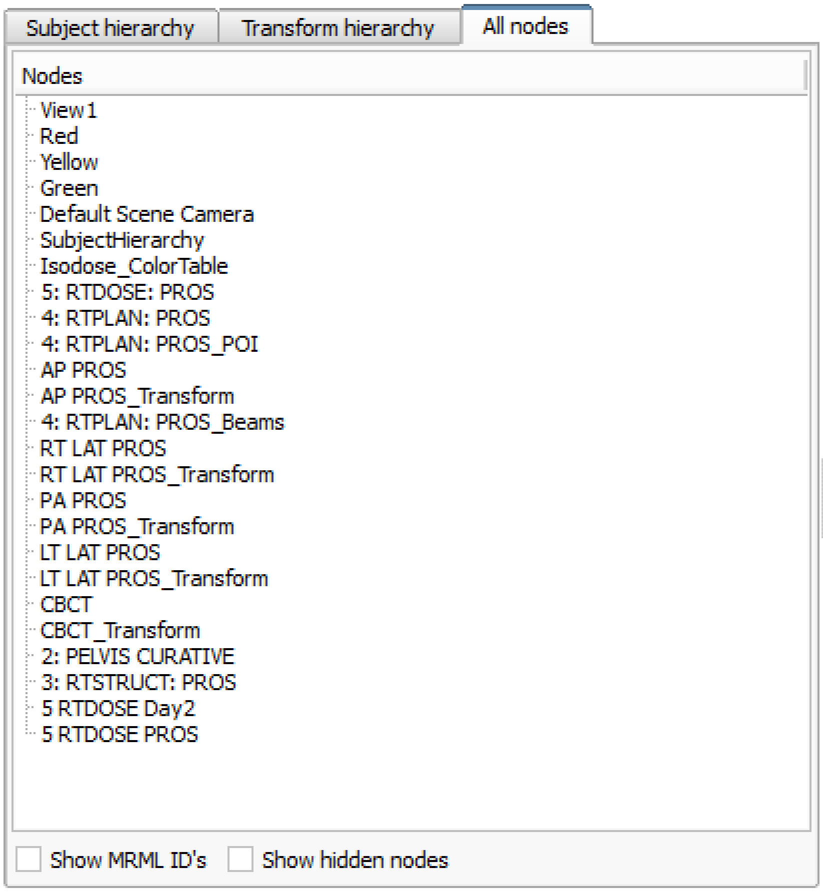
Transform hierarchy tab

- **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).

9.1.4 Tutorials

- [Data loading and visualization]https://training.slicer.org/#stc-vis-102-data-loading-and-3d-visualization) demonstrates the basic usage of Data module.

9.1.5 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++ qSlicerSubjectHierarchyScriptedPlugin class
    * Example: role plugin in SlicerHeart extension, function plugin in Segment Editor module

  - **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.

9.1.6 References

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

9.1.7 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)

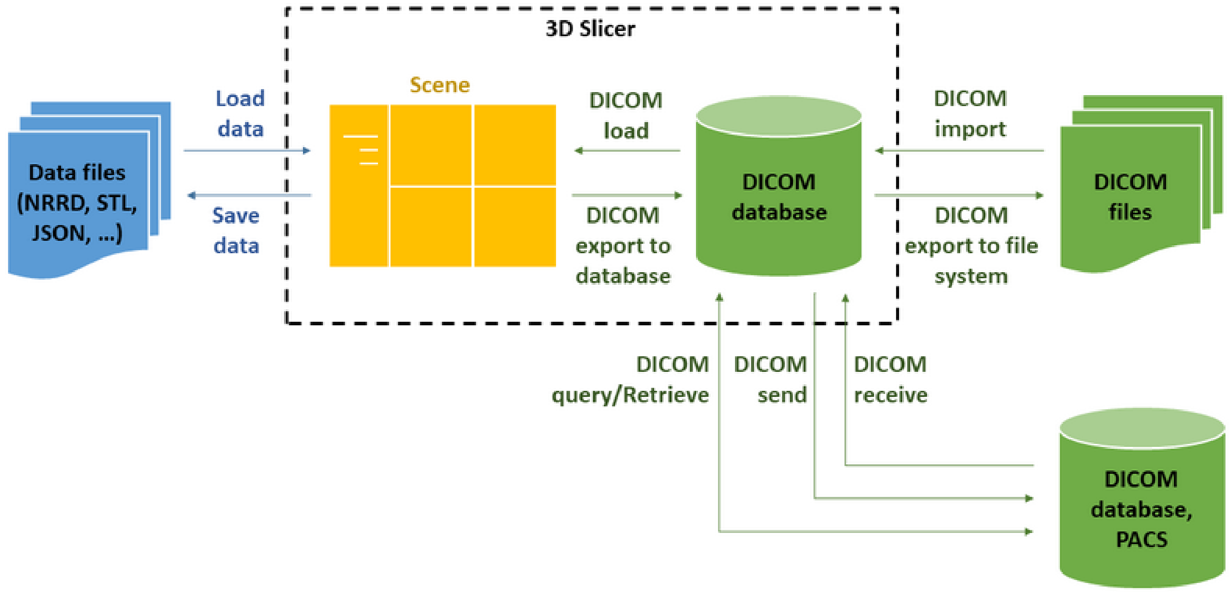
9.1.8 Acknowledgements

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

9.2 DICOM

9.2.1 Overview

This module allows importing and exporting and network transfer of DICOM data. Slicer provides support for the most commonly used subset of DICOM functionality, with the particular features driven by the needs of clinical research: **reading** and **writing** data sets from/to disk in DICOM format and network transfer - **querying**, **retrieving**, and **sending** and **receiving** data sets - using DIMSE and DICOMweb networking protocols.



DICOM introduction

Digital Imaging and Communications in Medicine (DICOM) is a widely used standard for information exchange digital radiology. In most cases, imaging equipment (CT and MR scanners) used in the hospitals will generate images saved as DICOM objects.

DICOM organizes data following the hierarchy of

- **Patient** ... can have 1 or more
  - **Study** (single imaging exam encounter) ... can have 1 or more
    - **Series** (single image acquisition, most often corresponding to a single image volume) ... can have 1 or more
      - **Instance** (most often, each Series will contain multiple Instances, with each Instance corresponding to a single slice of the image)

As a result of imaging exam, imaging equipment generates DICOM files, where each file corresponds to one Instance, and is tagged with the information that allows to determine the Series, Study and Patient information to put it into the proper location in the hierarchy.

There is a variety of DICOM objects defined by the standard. Most common object types are those that store the image volumes produced by the CT and MR scanners. Those objects most often will have multiple files (instances) for each series. Image processing tasks most often are concerned with analyzing the whole image *volume*, which most often corresponds to a single Series.

More information about DICOM standard:

- The DICOM Homepage: https://dicom.nema.org/

- DICOM on wikipedia: https://en.wikipedia.org/wiki/DICOM
- Clean and simple DICOM tag browser: https://dicom.innolitics.com
- A useful tag lookup site: http://dicomlookup.com/
- A hyperlinked version of the standard: https://web.archive.org/web/20180624030937/http://dabsoft.ch/dicom/

Slicer DICOM Database

To organize the data and allow faster access, Slicer keeps a local DICOM Database containing copies of (or links to) DICOM files, and basic information about content of each file. You can have multiple databases on your computer at a time, and switch between them if, for example, they include data from different research projects. Each database is simply a directory on your local disk that has a few SQLite files and subdirectories to store image data. Do not manually modify the contents of these directories. DICOM data can enter the database either through file import or via a DICOM network transfer. Slicer modules may also populate the DICOM database with computation results.

Note that the DICOM standard does not specify how files will be organized on disk, so if you have DICOM data from a CDROM or otherwise transferred from a scanner, you cannot in general tell anything about the contents from the file or directory names. However once the data is imported to the database, it will be organized according to the DICOM standard Patient/Study/Series hierarchy.

DICOM plugins

A main function of the DICOM module is to map from *acquisition* data organization into *volume* representation. That is, DICOM files typically describe attributes of the image capture, like the sequence of locations of the table during CT acquisition, while Slicer operates on image volumes of regularly spaced pixels. If, for example, the speed of the table motion is not consistent during an acquisition (which can be the case for some contrast 'bolus chasing' scans, Slicer's DICOM module will warn the user that the acquisition geometry is not consistent and the user should use caution when interpreting analysis results such as measurements.

This means that often Slicer will be able to suggest multiple ways of interpreting the data (such as reading DICOM files as a diffusion dataset or as a scalar volume. When it is computable by examining the files, the DICOM module will select the most likely interpretation option by default. As of this release, standard plugins include scalar volumes and diffusion volumes, while extensions are available for segmentation objects, RT data, and PET/CT data. More plugins are expected for future versions. It is a long-term objective to be able to represent most, if not all, of Slicer's data in the corresponding DICOM data objects as the standard evolves to support them.

9.2.2 How to

Create DICOM database

Creating a DICOM database is a prerequisite to all DICOM operations. When DICOM module is first opened, Slicer offers to create a new database automatically. Either choose to create a new database or open a previously created database.

You can open a database at another location anytime in DICOM module panel / DICOM database settings / Database location.

Read DICOM files into the scene

Since DICOM files are often located in several folders, they can cross-reference each other, and can be often interpreted in different ways, reading of DICOM files into the scene are performed as two separate steps: *import* (indexing files to be able to show them in the DICOM database browser) and *loading* (displaying selected DICOM items in the Slicer scene).

DICOM import

1. Make sure that all required Slicer extensions are installed. Slicer core contains DICOM import plugin for importing images, segmentations, parametric maps, and some structured reports, but additional extensions may be needed for other information objects. For example, *SlicerRT extension is needed for importing/exporting radiation therapy information objects (RT structure set, dose, image, plan). SlicerHeart extension is needed for importing certain ultrasound images.* See complete list in *supported data formats section*.
2. Go to DICOM module
3. Select folders that contain DICOM files
   - Option A: Drag-and-drop the folder that contains DICOM files to the Slicer application window.
   - Option B: Click "Import" button in the top-left corner of the DICOM browser. Select folder that contains DICOM files.
     - The import button has a drop-down menu, which can be displayed by clicking the small down-arrow button at the right side. Currently the only item in the menu is the "Copy imported files to DICOM database" option. Enable this option to copy all the imported DICOM files from their original location into the Slicer DICOM database. This is useful when loading data from removable media (CD/DVD/USB drives or remote drives), to be able to load the data set even after media is ejected. If the copy option is disabled then only the path of the imported files will be stored in the DICOM database (along with values of most commonly used DICOM tags).

Note

When a folder is drag-and-dropped to the Slicer application, Slicer displays a popup, asking what to do - click OK ("Load directory in DICOM database"). After import is completed, go to DICOM module.

DICOM loading

1. Go to DICOM module. Click "Show DICOM database" if the DICOM database window is not visible already (it shows a list of patients, studies, and series).
2. Double-click on the patient, study, or series to load.
3. Click "Show DICOM database" button to toggle between the database browser (to load more data) and the viewer (to see what is loaded into the scene already)

Tip

Selected patients/studies/series can be loaded at once by first selecting items to load. Shift-click to select a range, Ctrl-click to select/unselect a single item. If an item in the patient or study list is selected then by default all series that belong to that item will be loaded. Click "Load" button to load selected items.

Advanced data loading: It is often possible to interpret DICOM data in different ways. If the application loaded data differently than expected then check "Advanced" checkbox, click "Examine" button, select all items in the list in the bottom (containing DICOM data, Reader, Warnings columns), and click "Load".

Delete data from the DICOM database

By right clicking on a Patient, Study, or Series, you can delete the entry from the DICOM database. Note that to avoid accidental data loss, Slicer does not delete the corresponding image data files if only their link is added to the database. DICOM files that are copied into the DICOM database will be deleted from the database.

Export data from the scene to DICOM database

Data in the scene can be exported to DICOM format, to be stored in DICOM database or exported to DICOM files:

1. Make sure that all required Slicer extensions are installed. Slicer core contains DICOM export plugin for exporting images, DICOM segmentation objects, structured reports, and parametric maps, but additional extensions may be needed for other information objects.
   - `SlicerRT` extension is needed for importing/exporting radiation therapy information objects: RT structure set, RT dose, RT image, RT plan.
   - See complete list in *Supported data formats page*.
2. Go to Data module or DICOM module.
3. Right-click on a data node in the data tree that will be converted to DICOM format.
4. Select the export type in the bottom left of the export dialog. This is necessary because there may be several DICOM information objects that can store the same kind of data. For example, segmentation can be stored as DICOM segmentation object (modern DICOM) or RT structure set (legacy representation, mostly used by radiation treatment planning).
   - "Slicer data bundle" export type writes the entire scene to DICOM format by encapsulating the scene MRB package inside a DICOM file. The result as a DICOM secondary capture object, which can be stored in any DICOM archival system. This secondary capture information stores all details of the scene but only 3D Slicer can interpret the data.
   - Export type: Once the user selected a node, the DICOM plugins generate exportables for the series they can export. The list of the results appear in this section, grouped by plugin. The confidence number will be the average of the confidence numbers for the individual series for that plugin.
5. Optional: Edit DICOM tags that will be used in the exported data sets. The metadata from the select study will be automatically filled in to the Export dialog and you can select a Slicer volume to export.
   - DICOM tag editor consists of a list of tables. Tables for the common tags for the patient and study on the top, and the tags for the individual series below them.
   - "Tags" in the displayed table are not always written directly to DICOM tags, they are just used by the DICOM plugins to fill DICOM tags in the exported files. This allows much more flexibility and DICOM plugins can auto-populate some information and plugins can expose other export options in this list (e.g. compression, naming convention).
   - Save modified tags: check this checkbox to save the new tag values in the scene persistently.
   - How to set unique identifier tags:
     - `StudyInstanceUID` tag specifies which patient and study the new series will be added to. If the value is set to empty then a new study will be created. It is recommended to keep all patient and study values (`PatientName`, `PatientID`, `StudyID`, etc.) the same among series in the same study.
     - `SeriesInstanceUID` tag identifies an image series. Its value is set to empty by default, which will result in creation of a new UID and thereby a new series. In very rare cases users may want to specify a UID, but the UID cannot be any of the existing UIDs because that would result in the exported image slices being mixed into another series. Therefore, the UID is only accepted if it is not used for any of the images that are already in the database.
     - `FrameOfReferenceUID` tag specifies a spatial reference. If two images have the same frame of reference UID value then it means that they are spatially aligned. By default, the value is empty, which means that a new frame of reference UID is created and so the exported image is not associated with any other image. If an image is spatially registered to another then it is advisable to copy the frame of reference UID value from the other image, because this may be required for fused display of the images in some image review software.

6. Click Export

    - In case of any error, a short message is displayed. More details about the error are provided in the application log.

Notes:

- To create DICOM files without adding them to the DICOM database, check "Export to folder" option and choose an output folder.
- You should exercise extreme caution when working with these files in clinical situations, since non-standard or incorrect DICOM files can interfere with clinical operations.
- To prepare DICOM patient and study manually before export, go to Data module (subject hierarchy tab), right-click in the empty space in the data tree and choose Create new subject. New studies can be created under patients the same way.

This workflow is also explained in a 2-minute video tutorial.

Export data from the scene to DICOM files

DICOM data stored in the database can be exported to DICOM files by right-clicking in patient/study/series list and choosing "Export to file system".

Data nodes loaded into the scene can be directly exported as DICOM files in the file system by right-clicking on the item in Data module, choosing Export to DICOM, enabling "Export to folder" option, and specifying an output folder.

DICOM networking

DICOM is also a network communication standard, which specifies how data can be transferred between systems. Slicer offers the following feaures:

- DICOM listener (C-STORE SCP): to receive any data that is sent from a remote computer and store in Slicer DICOM database
- DICOM sender (C-STORE SCU): select data from Slicer DICOM database and send it to a remote computer. Supports both traditional DIMSE and new DICOMweb protocols.
- Query/retrieve (C-FIND SCU, C-FIND SCU): query list of images available on a remote server and retrieve selected data.

Note: In order to use these features, you must coordinate with the operators of the other DICOM nodes with which you wish to communicate. For example, you must work out agreement on such topics as network ports and application entity titles (AE Titles). Be aware that not all equipment supports all networking options, so configuration may be challenging and is often difficult to troubleshoot.

Connection ports: Port 104 is the standard DICOM port. All ports below 1024 require root access on unix-like systems (Linux and Mac). So you can run Slicer with the sudo command to be able to open the port for the DICOM Listener. Or you can use a different port, like 11112. You need to configure that on both sides of the connection. You can only have one process at a time listening on a port so if you have a listener running the second one won't start up. Also if something adverse happens (a crash) the port may be kept open an you need to either kill the storescp helper process (or just reboot the computer) to free the port. Consult the Look at error log for diagnostic information.

DICOMweb networking

Slicer supports sending of DICOM items to a remote server using DICOMweb protocol. In send data window, set the full server URL in "Destination Address" and choose "DICOMweb" protocol.

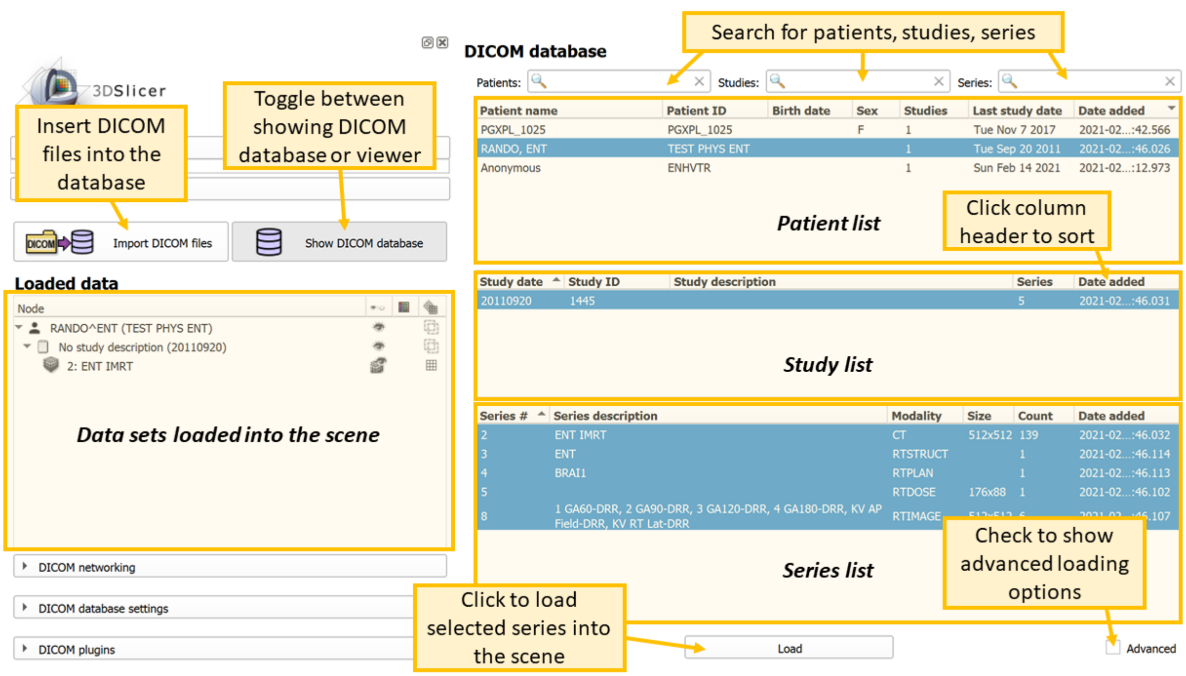
View DICOM metadata

1. Go to DICOM module
2. Right-click on the item in the DICOM database window that you want to inspect
3. Choose "View DICOM metadata"

9.2.3 Panels and their use

Basic usage

- **Import DICOM files**: all DICOM files in the selected folder (including subfolders) will be scanned and added to the Slicer DICOM database. If "Import directory mode" is set to "Copy" then Slicer will make a copy of the imported files into the database folder. It is recommended to copy data if importing files from removable media (CD/DVD/USB drives) to be able to load the data set even after media is ejected. Otherwise they will only be referenced in their original location.
- **Show DICOM database**: toggle between DICOM browser and viewers (slice view, 3D view, etc.)
- **Patient list**: shows patients in the database. Studies available for the selected patient(s) are listed in study list. Multiple patients can be selected.
- **Study list**: shows studies for the currently selected patient(s). Multiple studies can be selected.
- **Series list**: shows list of series (images, structure sets, segmentations, registration objects, etc.) available for selected studies.
- **Load**: click this button to load currently selected loadables into Slicer.
- **Loaded data**: shows all content currently loaded into the scene, which can be displayed in viewers by clicking the eye icon



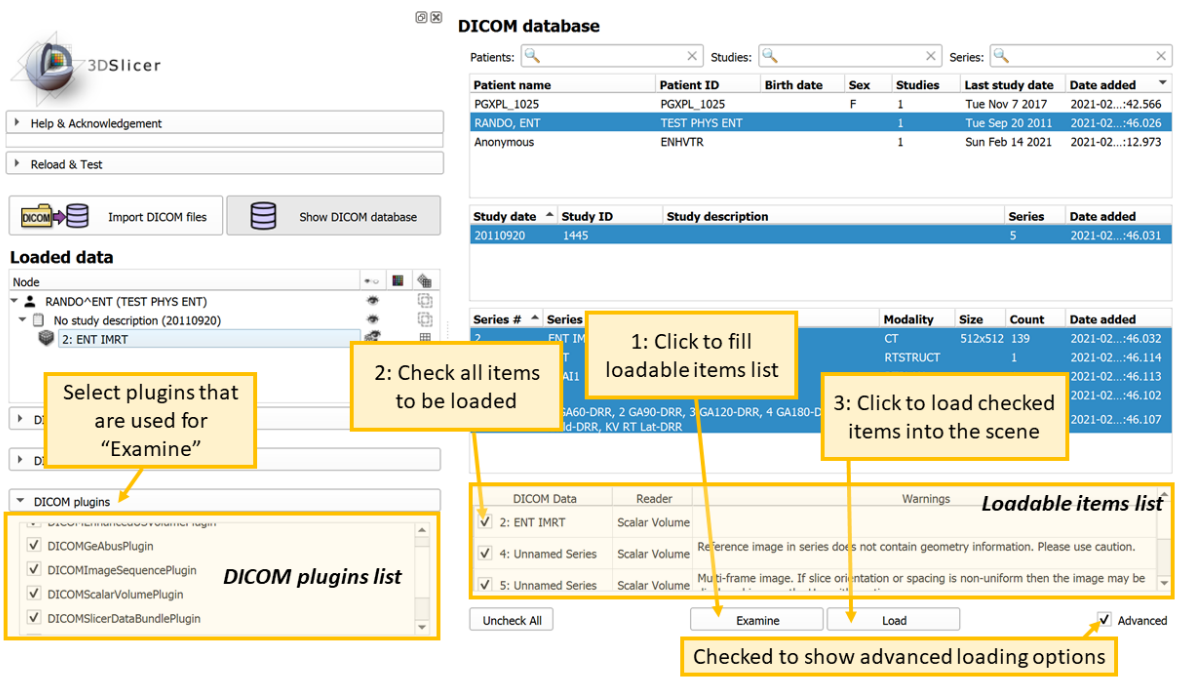
Additional options:

- **Search boxes**: each patient/study/series can be filtered by typing in these fields.
- Right-click menu item in patient/study/series list:

   - **View DICOM metadata:** view metadata stored in file headers of selected series
   - **Delete:** delete the selected item from the database. If the data set was copied into the DICOM database then the DICOM files are deleted, too.
   - **Export to file system:** export selected items to DICOM files into a selected folder
   - **Send to DICOM server:** send selected items to a remote DICOM server using DIMSE (C-store SCP) or DICOMweb (STOW-RS) protocol.

Advanced loading (allows loading DICOM data sets using non-default options):

- **Advanced:** check this checkbox to show advanced loading options
- **Plugin selector section:** you can choose which plugins will be allowed to examine the selected series for loading. This section is displayed if you click on "DICOM plugins" collapsible button at the bottom of DICOM module panel.
- **Examine button:** Runs each of the DICOM Plugins on the currently selected series and offers the result in the Loadable items list table.
- **Loadable items list:** displays all possible interpretations of the selected series by the selected plugins. The plugin that most likely interprets the series correctly, is selected by default. You can override the defaults if you want to load the data in a different way. There will not always be a one-to-one mapping of selected series to list of loadable items.



DICOM module settings:

- **DICOM networking**: download data from remote server using query retrieve, set up receiving data via C-store SCP
- **DICOM database settings**: allows you to select a location on disk for Slicer's database of DICOM files. The application manages content of this folder (stores metadata and copy of imported DICOM files): do not manually copy any data into this folder.
- Additional settings are available in menu: Edit / Application Settings / DICOM:
   - Generic DICOM settings:

    * Load referenced series will give you the option of easily loading, for example, the source volume of a segmentation when you open the segmentation. This can also be made to happen automatically.

  – DICOMScalarVolumePlugin settings:

    * You can choose what back-end library to use (currently GDCM, DCMTK, or GDCM with DCMTK fallback with the last option being the default. This is provided in case some data is unsupported by one library or the other.

    * Acquisition geometry regularization option supports the creation of a nonlinear transform that corrects for things like missing slices or gantry tilt in the acquisition. The regularization transformation can also be hardened to the volume. See more information here

    * Autoloading subseries by time is an option break up some 4D acquisitions into individual volume, but is optional since some volumes are also acquired in time unites and should not be split.

9.2.4 Visual DICOM Browser

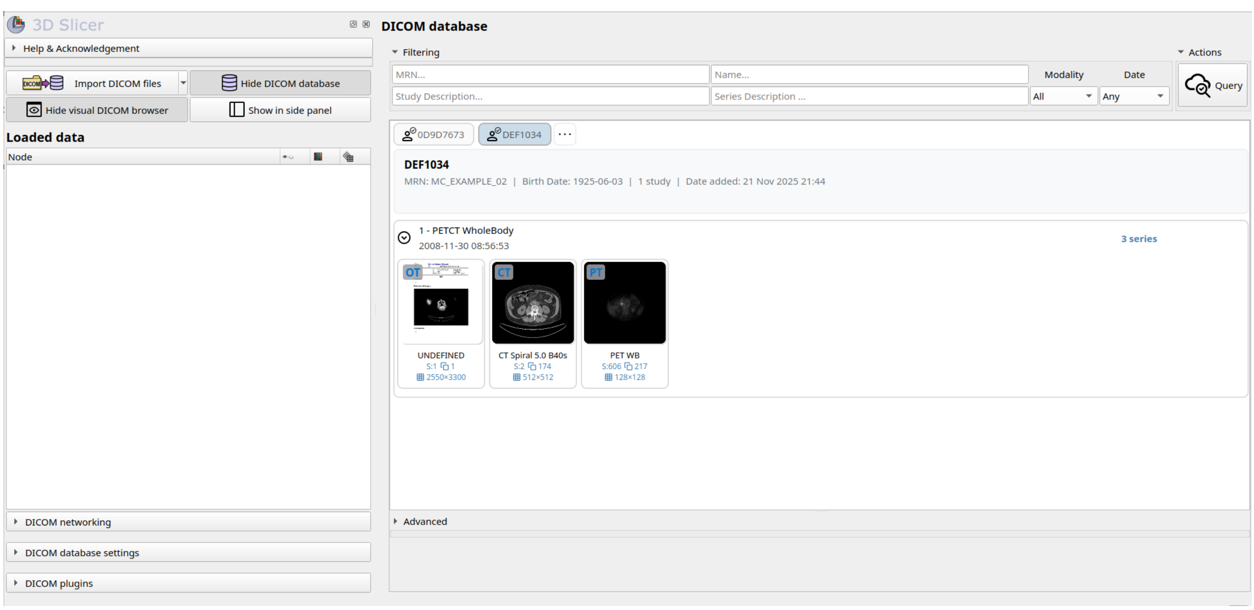
The Visual DICOM Browser provides a thumbnail-based interface for browsing and loading DICOM data. This interface is particularly useful when you need to quickly identify images visually.

When to use

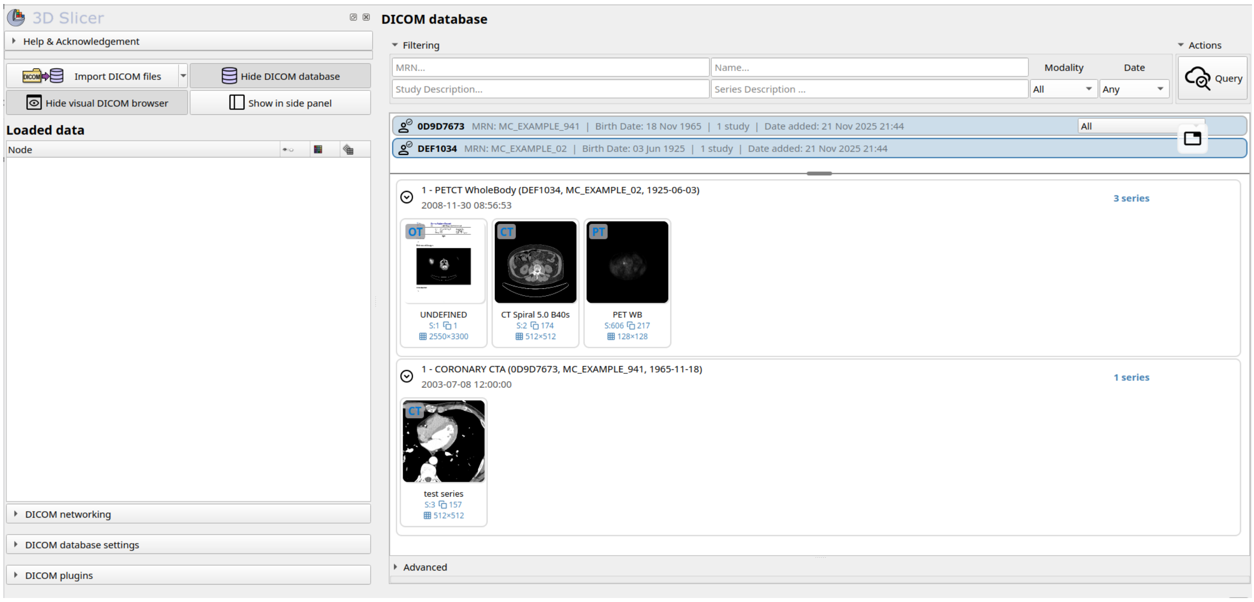
- When you need to quickly identify anatomical regions or specific acquisitions visually
- In multi-monitor setups where you want the browser displayed on a secondary display

Features

- **Thumbnail previews**: View preview images of all series in the database
- **Quick visual identification**: Easily identify the correct series by viewing the images
- **Tab patient mode and list mode**: Toggle between a tab-based patient view (optimized for clinical workflows) and a list mode (ideal for managing large research databases)

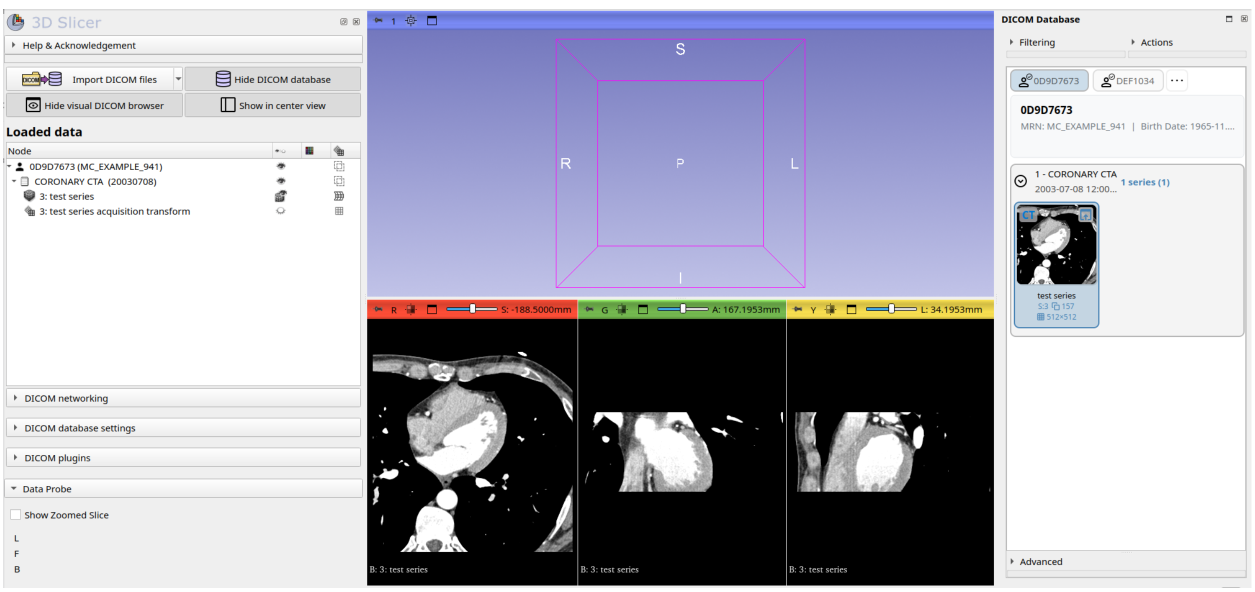


Tab patient mode: organized by patient tabs for clinical workflows



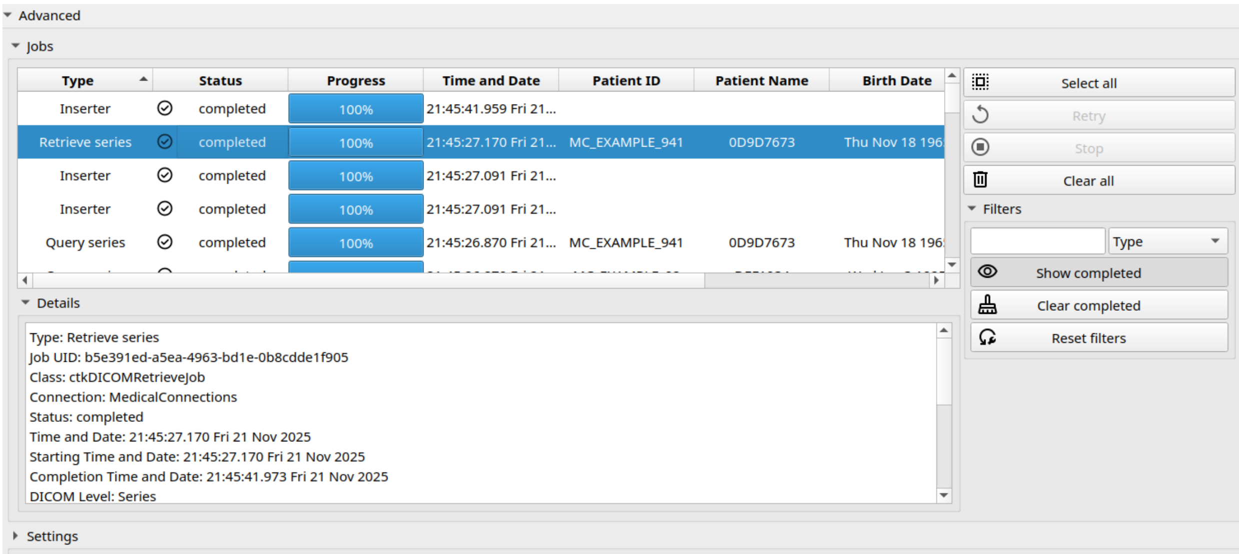
List mode: flat list view ideal for large research databases

- **Dynamic thumbnails size settings**: Choose between different thumbnail sizes (small, medium, large) to suit your workflow
- **Side panel mode**: Display the Visual DICOM Browser in a dockable side panel, allowing you to keep the browser visible while working with data in the main views



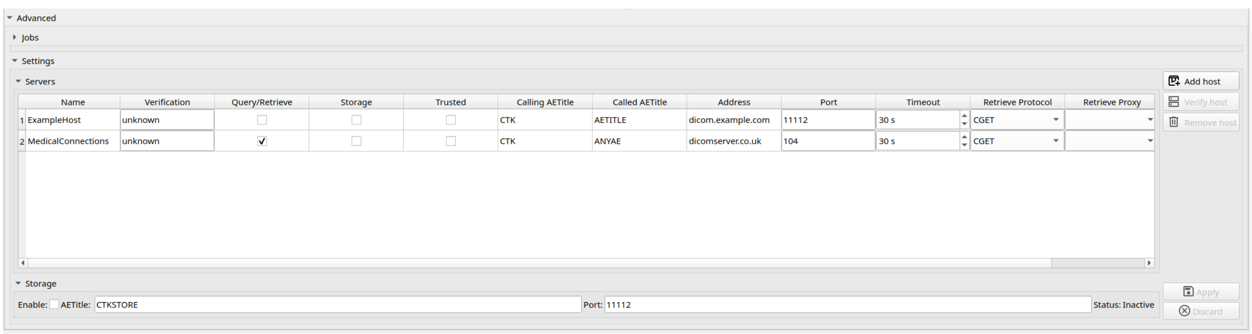
Side panel mode: keep the browser visible while working with data

- **Integrated query/retrieve**: Query and retrieve functionality is built directly into the UI, allowing the browser to display both local database content (with interactive filtering) and remote database content (with loading progress bars and status reports)
- **Advanced job tracking**: The advanced section includes comprehensive job tracking with detailed logging for each query and retrieve operation



Job tracking: monitor query and retrieve operations with detailed logs

- **Server configuration**: Configure multiple servers with their individual settings



Server settings: configure multiple DICOM servers

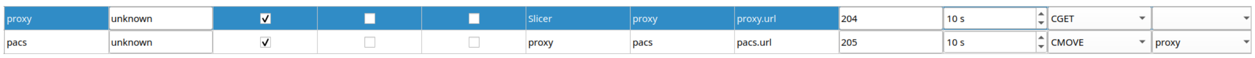
How to use

1. In the DICOM module, click the "Visual browser" button to toggle between the traditional text-based browser and the Visual DICOM Browser
2. Navigate and load data:
   - **Double-click** on a series to load all selected series into the scene
   - **Right-click** on a series to open a context menu with additional options
3. Adjust thumbnail size using any of these methods:
   - **Keyboard shortcuts**: Press `Ctrl+` to increase or `Ctrl-` to decrease thumbnail size
   - **Mouse scroll**: Hold `Ctrl` and scroll with the mouse wheel while hovering over the widget
   - **Application Settings**: Navigate to Edit / Application Settings / DICOM / Thumbnails size to set a specific size
4. (Optional) Click "Side panel" to dock the browser to the side of the main window, allowing you to work with your data while keeping the browser visible
5. The docked browser can be:

- Moved to different dock areas (left, right, top, bottom)
- Undocked and floated as a separate window
- Maximized to fullscreen using the maximize button in the dock's title bar
- Returned to center view by clicking "Show in center view"

Advanced proxy server configuration

Advanced configuration allows you to specify a PACS system using the C-MOVE protocol along with a proxy using the C-GET protocol. When configured this way, the C-MOVE operation will be automatically executed on the proxy server, and C-GET will automatically start to retrieve the data to Slicer once the C-MOVE retrieve is finished.



proxy	unknown	☑	☐	☐	Slicer	proxy	proxy.url	204	10 s	CGET	
pacs	unknown	☑	☐	☐	proxy	pacs	pacs.url	205	10 s	CMOVE	proxy

Proxy server configuration: advanced setup for C-MOVE with proxy C-GET

Note: DICOMweb protocol is not yet supported. See the development roadmap for more information.

9.2.5 Troubleshooting

How do I know if the files I have are stored using DICOM format? How do I get started?

DICOM files do not need to have a specific file extension, and it may not be straightforward to answer this question easily. However, if you have a dataset produced by a clinical scanner, it is most likely in the DICOM format. If you suspect your data might be in DICOM format, it might be easiest to try to load it as DICOM:

1. drag and drop the directory with your data into Slicer window. You will get a prompt "Select a reader to use for your data? Load directory into DICOM database." Accept that selection. You will see a progress update as the content of that directory is being indexed. If the directory contained DICOM data, and import succeeded, at the completion you will see the message of how many Patient/Study/Series/Instance items were successfully imported.
2. Once import is completed, you will see the window of the DICOM Browser listing all Patients/Studies/Series currently indexed. You can next select individual items from the DICOM Browser window and load them.
3. Once you load the data into Slicer using DICOM Browser, you can switch to the "Data" module to examine the content that was imported.

When I click on "Load selection to Slicer" I get an error message "Could not load … as a scalar volume"

A common cause of loading failure is corruption of the DICOM files by incorrect anonymization. Patient name, patient ID, and series instance UID fields should not be empty or missing (the anonymizer should replace them by other valid strings). Try to load the original, non-anonymized sequence and/or change your anonymization procedure.

If none of the above helps then check the Slicer error logs and report the error on the Slicer forum. If you share the data (e.g., upload it to Dropbox and add the link to the error report) then Slicer developers can reproduce and fix the problem faster.

I try to import a directory of DICOM files, but nothing shows up in the browser

DICOM is a complex way to represent data, and often scanners and other software will generate 'non-standard' files that claim to be DICOM but really aren't compliant with the specification. In addition, the specification itself has many variations and special formats that Slicer is not able to understand. Slicer is used most often with CT and MR DICOM objects, so these will typically work.

If you have trouble importing DICOM data here are some steps to try:

- Make sure you are following the *DICOM loading instructions*.

- We are constantly improving the application (new preview version is released every day), so there is a chance that the problem you encountered is addressed in a recent version. Try loading the data using the latest stable and the latest nightly versions of Slicer.
- Make sure the Slicer temporary folder is writeable. Temporary folder can be selected in menu: Edit / Application Settings / Modules / Temporary directory.
- Try moving the data and the database directory to a path that includes only US English characters (ASCII) to avoid possible parsing errors. No special, international characters are allowed.
- Make sure the database directory is on a drive that has enough free space (1GB free space should be enough). If you are running out of space then you may see this error message in an "Internal Error" popup window: *Exception thrown in event: Calling methods on uninitialized ctkDICOMItem*
- Import the files from local storage - physical drive or USB stick connected directly to the computer (not network drive, shared drive, cloud drive, google drive, virtual file system, etc.)
- Make sure filename is not very long (below a few ten characters) and full file path on Windows is below about 200 characters
- To confirm that your installation of Sicer is reading data correctly, try loading other data, such as this anonymized sample DICOM series (CT scan)
- Try import using different DICOM readers: in Application settings / DICOM / DICOMScalarVolumePlugin / DICOM reader approach: switch from DCMTK to GDCM (or GDCM to DCMTK), restart Slicer, and attempt to load the data set again.
- See if the SlicerDcm2nii extension will convert your images. You can install this module using the Extension manager. Once installed you will be able to use the Dcm2niixGUI module from Slicer.
- Try the *DICOM Patcher* module.
- Review the Error Log (menu: View / Error log) for information.
- Try loading the data by selecting one of the files in the *Add data*. *Note: be sure to turn on Show Options and then turn off the Single File option in order to load the selected series as a volume*. In general, this is not recommended, as the loaded data may be incomplete or distorted, but it might work in some cases when proper DICOM loading fails.
- If you are still unable to load the data, you may need to find a utility that converts the data into something Slicer can read. Sometimes tools like FreeSurfer, FSL, dcm2niix, dicom2nifti, or MRIcron can understand special formats that Slicer does not handle natively. These systems typically export NIfTI files that Slicer can read.
- For archival studies, are you sure that your data is in DICOM format, or is it possible the data is stored in one of the proprietary MR or CT formats that predated DICOM? If the latter, you may want to try the dcm2nii tool distributed with MRIcron up until 2016. More recent versions of MRIcorn include dcm2niix, which is better for modern DICOM images. However, the legacy dcm2nii includes support for proprietary formats from GE, Philips, Siemens and Elscint.
- If none of the above help, then you can get help from the Slicer developer team, by posting on the Slicer forum a short description of what you expect the data set to contain and the following information about the data set:
  - You may share the DICOM files if they do not contain patient confidential information: upload the dataset somewhere (Dropbox, OneDrive, Google drive, …) and post the download link. *Please be careful not to accidentally reveal private health information (patient name, birthdate, ID, etc.)*. If you want to remove identifiers from the DICOM files you may want to look at DicomCleaner, gdcmanon or the RSNA Clinical Trial Processor software.
  - If it is not feasible to share the DICOM files, you may share the DICOM metadata and application log instead. Make sure to **remove patient name, birthdate, ID, and all other private health information** from the text, upload the files somewhere (Dropbox, OneDrive, Google drive, …), and post the download link.

* To obtain DICOM metadata: right-click on the series in the DICOM browser, select View metadata, and click Copy Metadata button. Paste the copied text to any text editor.
* To obtain detailed application log of the DICOM loading: Enable detailed logging for DICOM (menu: Edit / Application settings / DICOM / Detailed logging), then attempt to load the series (select the series in the DICOM browser and click "Load" button), and retrieve the log (menu: Help / Report a Bug -> Copy log messages to clipboard).

Something is displayed, but it is not what I expected

I would expect to see a different image

When you load a study from DICOM, it may contain several data sets and by default Slicer may not show the data set that you are most interested in. Go to Data module / Subject hierarchy section and click the "eye" icons to show/hide loaded data sets. You may need to click on the small rectangle icon ("Adjust the slice viewer's field of view...") on the left side of the slice selection slider after you show a volume.

If none of the data sets seems to be correct then follow the steps described in section "I try to import a directory of DICOM files, but nothing shows up in the browser".

Image is stretched or compressed along one axis

Some non-clinical (industrial or pre-clinical) imaging systems do not generate valid DICOM data sets. For example, they may incorrectly assume that slice thickness tag defines image geometry, while according to DICOM standard, image slice position must be used for determining image geometry. *DICOM Patcher* module can fix some of these images: remove the images from Slicer's DICOM database, process the image files with DICOM Patcher module, and re-import the processed file into Slicer's DICOM database. If image is still distorted, go to *Volumes* module, open *Volume information* section, and adjust *Image spacing* values.

Scanners may create image volumes with varying image slice spacing. Slicer can represent such images in the scene by apply a non-linear transform. To enable this feature, go to menu: Edit / Application settings / DICOM and set *Acquisition geometry regularization* to *apply regularization transform*. Slice view, segmentation, and many other features work directly on non-linearly transformed volumes. For some other features, such as volume rendering, you need to harden the transform on the volume: go to Data module, in the row of the volume node, right-click on the transform column, and choose *Harden transform*.

Note that if Slicer displays a warning about non-uniform slice spacing then it may be due to missing or corrupted DICOM files. There is no reliable mechanism to distinguish between slices that are missing because they had not been acquired (for example, to reduce patient dose) or they were acquired but later they were lost.

9.2.6 Slicer Integration in hospital/PACS context

This section provides practical recommendations for working with hospital IT/PACS teams to integrate Slicer (or Slicer-based applications) into clinical or research workflows.

Core assumptions & constraints

It is not possible to *scan the network* to discover PACS. DICOM Application Entities (AEs) don't self-advertise; endpoints must be explicitly provisioned (host/IP, port, AE Title) and allowed through firewalls. Leveraging its DICOM networking capabilities, Slicer is itself a DICOM AE.

There are two fundamental flows to work with DICOM:

1. **Push:** An AE (e.g., scanner, PACS/router) pushes (`C-STORE`) to Slicer's AE.
2. **Pull:** Slicer queries (`C-FIND`) and retrieves (`C-MOVE` or `C-GET`) from a PACS/VNA/router. (Slicer may also use **DICOMweb**: `QIDO-RS`/`WADO-RS`/`STOW-RS`.)

In hospitals with many modalities/AEs, **prefer central integration** via the enterprise **PACS/VNA** (or a router/broker) rather than connecting Slicer to each modality individually.

Supported technical patterns

Push to Slicer (`C-STORE` → Slicer)

Architecture: A modality, PACS, or routing node pushes specific studies/series to Slicer's AE.

- **Pros:** Simple user experience; deterministic; works well for scheduled research/processing pipelines.
- **Cons:** Requires routing rules; local copies on the Slicer host can become a shadow archive unless retention is governed; ensure consistent Patient ID/Accession handling.
- **Recommendations**
  - Register Slicer AE with PACS: **AE Title / Host / Port** (commonly 104 or 11112).
  - Create **routing rules** (e.g., by Accession Number, Modality, Study Description, Research flag).
  - Use **TLS** (DICOM over TLS) if mandated by policy; exchange certs with PACS.
  - Set storage quota & PHI retention policy on the Slicer host.

Best for: Controlled workflows (e.g., "all cardiac MR from clinic X to Slicer-Lab").

Pull from PACS/VNA (Query/Retrieve)

Architecture: Slicer performs `C-FIND` queries and `C-MOVE`/`C-GET` retrievals against PACS/VNA or a DICOM router.

- **Pros:** On-demand; one integration point covers many modalities.
- **Cons:** Users need patient/study identifiers; retrieved studies also create local copies—enforce retention/purge to avoid a shadow archive; `C-MOVE` must target a valid destination AE.
- **Recommendations**
  - Provision PACS endpoint in Slicer (AE/host/port).
  - **C-MOVE destination:** either
    * **Slicer's own AE** (built-in **C-STORE SCP**), or
    * a **local receiving AE** on the same host/subnet (e.g., **Orthanc**, **dcmtk `storescp`**) that writes to a folder or local store. Slicer then **imports** from that local store—either by performing **DIMSE Query/Retrieve** against the local AE, or by **importing from a designated "hot folder"** that the receiver writes to. If your local store exposes **DICOMweb**, you may also fetch via `QIDO-RS`/`WADO-RS`
  - Enforce role-appropriate **access control** and **auditing**.
  - **TLS** per security policy.

Best for: Interactive case selection; ad-hoc research pulls; multi-modality environments; many upstream AEs via one PACS.

DICOMweb (`QIDO-RS` / `WADO-RS` / `STOW-RS`)

Architecture: Slicer uses `QIDO-RS` to search, `WADO-RS` to fetch, and optionally `STOW-RS` to push.

- **Pros:** Modern HTTP(S); easier firewalling/reverse proxy; integrates with enterprise identity when used with a reverse proxy/broker; **well-suited to hybrid setups**—for example, **Slicer running on-prem** accessing a **cloud-hosted archive/service** through an HTTPS API gateway. The inverse also works (cloud-hosted Slicer reaching an on-prem PACS/VNA via a reverse proxy). In both cases you typically standardize on port **443**, use normal TLS termination, and leverage identity brokering without bespoke VPNs.

- **Cons:** Requires DICOMweb enablement on PACS/VNA and a compatible Slicer service for DICOMweb workflows. Core Slicer does not natively handle OIDC flows; use a proxy/broker.

Recommendations

- PACS/VNA DICOMweb base URLs; **mutual TLS** and/or **OIDC** tokens per site policy.
- Confirm **multipart** support and accepted **transfer syntaxes**; document any required auth headers or proxy paths.
- **Reachability:** The DICOMweb reverse proxy does not need to be internet-wide. Prefer **private connectivity** (VPN/PrivateLink) with a private DNS name and firewall rules allowing only hospital subnets. If a public URL is used, restrict by **IP allowlists**, require **mTLS** and/or **OIDC** at the proxy, and protect with a **WAF/rate-limiting** policy.

Definition

> A **reverse proxy** is an HTTPS front end (e.g., NGINX, Apache, Kong) placed **in front of** the PACS/VNA. It terminates TLS and can perform **OIDC/OAuth2** authentication with the hospital identity provider (IdP). The reverse proxy forwards authorized **DICOMweb** requests to the internal service. This setup allows **Slicer (on-prem or in cloud)** to connect over HTTPS to a single URL, while the **PACS/VNA** remains isolated on its private network or VPC (reachable via private connectivity or a tightly restricted public endpoint).

Best for: Web-native environments, **on-prem Slicer cloud archive** integrations via API gateways, or sites standardizing on DICOMweb.

Broker/Router middle tier (Orthanc, dcm4chee-arc, commercial routers)

Architecture: A broker sits between PACS/VNA and Slicer to handle routing, study filtering, de-identification, caching, throttling, and DIMSEDICOMweb translation.

- **Pros:** Decouples Slicer from PACS; central place for **filters, de-identification, study whitelists**, and **access auditing**.
- **Cons:** Additional system to deploy/patch/monitor.

Best for: Research programs, multi-site feeds, de-identification pipelines, or batch processing.

Security, privacy, and governance

- **Intended use:** Slicer/Slicer-based apps are typically **not cleared as diagnostic devices**; document intended use in SOPs.
- **PHI handling:** For research, perform **de-identification upstream** (preferably at the router) so only compliant data reaches Slicer.
- **Transport security:** Use **DICOM over TLS** (mutual certificates) and/or **HTTPS** for DICOMweb; plan certificate ownership and rotation.
- **Identity & access:** Prefer centralized auth (AD/LDAP/SSO via broker or DICOMweb). Apply least-privilege access to query/pull.
- **Audit:** Enable and retain **ATNA-style** logs (who accessed what/when). See **DICOM PS3.15 – A.5 Audit Trail Message Format Profile** (link) and **IHE ITI TF Vol. 1, Ch. 9 – ATNA** (link). Forward audit records to the organization's **SIEM (Security Information and Event Management)** for centralized monitoring and retention.
- **Storage & retention:** Define local **quota** and **auto-purge** policy (e.g., delete after N days) to avoid shadow archives; encrypt disks.

9.2.7 Information for developers

See examples and other developer information in *Developer guide* and *Script repository*.

9.2.8 Related extensions and modules

- *Add data* dialog can be used to load some DICOM images directly, with bypassing the DICOM database. This may be faster in some cases, but it is not recommended, as it only supports certain kind of images and consistency and correctness of the data is not verified.
- SlicerRT extension reads and write DICOM Radiation Therapy objects (RT structure set, dose, image, plan, etc.) and provides tools for visualizing and analyzing them.
- SlicerHeart extension reads certain Philips, GE, Eigen Artemis ultrasound images.
- LongitudinalPETCT extension reads all PET/CT studies for a selected patient and provides tools for tracking metabolic activity detected by PET tracers.
- *DICOM Patcher* module can be used before importing to fix common DICOM non-compliance errors.

9.2.9 Contributors

Authors:

- Steve Pieper (Isomics Inc.)
- Michael Onken (Offis)
- Marco Nolden (DFKZ)
- Julien Finet (Kitware)
- Stephen Aylward (Kitware)
- Nicholas Herlambang (AZE)
- Alireza Mehrtash (BWH)
- Csaba Pinter (PerkLab, Queen's)
- Andras Lasso (PerkLab, Queen's)
- Jean-Christophe Fillion-Robin (Kitware)
- Davide Punzo (DNA-HIVE/PI3)

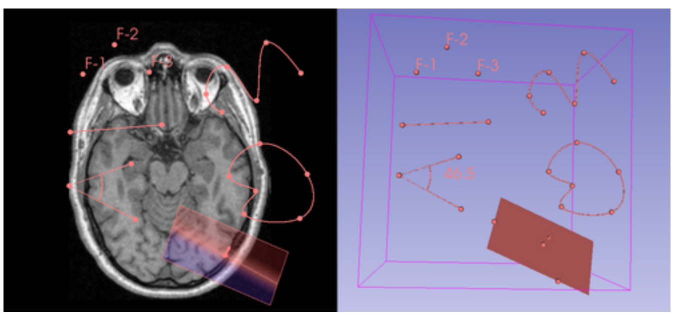
9.2.10 Acknowledgements

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, and by Quantitative Image Informatics for Cancer Research (QIICR) (U24 CA180918).

9.3 Markups

9.3.1 Overview

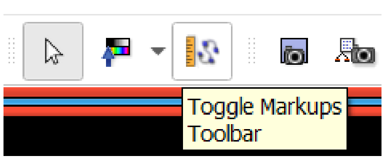
This module is used to create and edit markups (point list, line, angle, curve, closed curve, plane, ROI etc.) and adjust their display properties.



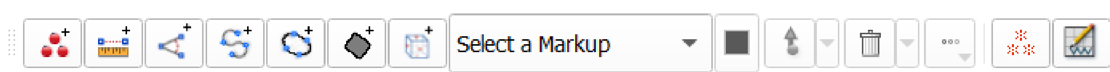
9.3.2 How to

Place new markups

1. Click the "Toggle Markups Toolbar" button in the Mouse Interaction toolbar to show/hide the Markups toolbar.



Using the Markups toolbar, click a markups type button to create a new object. The mouse interaction mode will automatically switch into control point placement mode.



Click the down arrow of the control point place button in the Markups toolbar to select the "Place multiple control points" checkbox to keep placing control points continuously, without the need to click the place button after each point.

2. Left-click in a slice view or 3D view to place points.

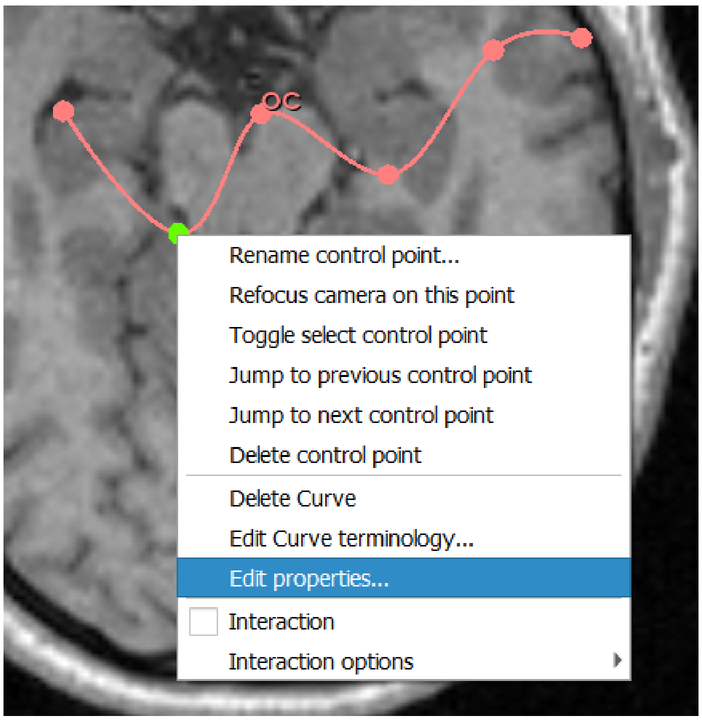
3. Double-left-click or right-click to finish point placement.

Edit control point positions in existing markups

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

Edit properties of a markup that is picked in a view

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



Edit Plane markups

- Planes can be defined using 3 "plane types": Point normal (default, place one or two points defining the origin and normal), 3 points (place 3 points to define the origin and plane axes), and plane fit (place any number of points that will be fit to a plane).
- When placing a plane with the "point normal" plane type, Alt + Left-click will allow the placement of 2 points. Placing the first point will define the origin of the plane, while the second point will define the normal vector.
- If the handles are not visible, right-click on the plane outline, or on a control point, and check "Interaction handles visible".
- Plane size can be changed using handles on the corners and edges of the plane.
- Left-click-and-drag on interaction handles to change the plane size.

- Resizing a plane will change the size mode to “absolute”, preventing changes in the control points from affecting the plane size.
- Plane type and size mode can be changed from the “Plane settings” section of the Markups module.

Edit ROI markups

- ROI size can be changed using handles on the corners and faces of the ROI.
- If the handles are not visible, right-click on the ROI outline, or on a control point, and check “Interaction handles visible”.
- Left-click-and-drag on interaction handles to change the ROI size.
- Alt + Left-click-and-drag to symmetrically adjust the ROI size without changing the position of the center.

9.3.3 Keyboard shortcuts

The following keyboard shortcuts are active when the markups toolbar is displayed.

9.3.4 Panels and their use

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

Display section

- Visibility: Toggle the markup visibility, which will override the individual control point 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 markup.
- 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 handles:
  - Visibility: Check `Visible` to enable translation/rotation/scaling of the entire markups in slice and 3D views with an interactive widget.
  - Translate, Rotate, Scale: enable/disable adjustment types.
  - Size: size of the handles (relative to the application window size).
  - Opacity: overall opacity of the handle. It is combined with the orientation-based fading of handles.
  - More options: Clicking more options will show/hide check boxes for controlling the visibility of each interaction handle axis separately.
- Advanced:
  - View: Select which views the markup 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.

  - Active Color: Select the color that will be used to display the glyph and text when the mouse hovers over the markup.
  - Glyph Type: Select the symbol that will be used to mark each location. Default is the Sphere3D.
  - Line thickness: The thickness of lines in markups. Defined as either an absolute thickness, or as a percentage of the glyph size.
  - Outline: Visibility and opacity of the markups outline.
  - Fill: Visibility and opacity of the markups fill.
  - Properties Label: Check to display node name and measurements.
  - Control Point Labels: Check to display a label for each control point.
  - Text display:
    * Font: Change the properties of the font used to display the labels.
    * Background: Change the label background color and opacity.
  - 3D Display:
    * Placement mode: Defines how points are placed and moved in views
      · Unconstrained: Point is moved independently from displayed objects in 3D views (e.g., in parallel with camera plane).
      · Snap to visible surface: Point is snapped to any visible surface in 3D views.
    * Occluded visibility: Controls the visibility and opacity of markups that are occluded. If enabled, the markup will remain visible even when it is blocked from view by other nodes (eg. volume rendering, segmentations, models, etc.).
  - 2D Display:
    * Projection visibility: Check to enable or disable visualization of projected control points on 2D viewers. The projection of the control points in the 2D viewers will be displayed onto slices around the one on which the control points have been placed.
    * Use Markup Color: If checked, color the projections the same color as the markup.
    * Projection Color: If not using markup color for the projection, use this color.
    * Outlined Behind Slice Plane: Control point 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).
- Scalars: Color markup according to a scalar, e.g. a per-control-point measurement (see Measurements section below)
  - Visibility: Controls the visibility of the scalars on the markups node.
  - Active Scalar: Select the scalar value that should be displayed.
  - Color Table: Select the color table that should be used to display the scalars.
  - Scalar Range Mode: Select the mode that should be used to control the mapping the scalar range onto the color node.
    * Manual: range is set manually
    * Data scalar range: range is set to the value range of the active scalar
    * Color table: use range specified in the color table. Useful for showing several nodes using the same color mapping.

    * Data type: range is set to the possible range of the active scalar's data type. This is only useful mostly for 8-bit scalars.
    * Direct color mapping: if active scalar has 3 or 4 components then those are interpreted as RGB or RGBA values.
  - Displayed Range: The currently used scalar range.
- Color Legend: Controls the color legend for the currently active scalars.
  - Visibility: Controls the visibility of the color legend in the views.
  - Views: Select which views the color legend should be displayed in.
  - Title: Set the title of the color legend.
  - Label text: Display either the scalar values or the name of the color.
  - Number of labels: Change the number of value labels that should be displayed on the legend.
  - Number of colors: Change the maximum number of colors to display.
  - Orientation: Change the display of the color legend between vertical and horizontal.
  - Position: Adjust the position of the legend in the views.
  - Size: Adjust the size of the legend in the views.
  - Title/Label properties: Controls the display of the title.
    * Format: Change the format used to display the scalar values in the legend (number of decimals, etc.) using a printf style string.
    * Color: Change the color of the title/label.
    * Opacity: Change the opacity of the title/label.
    * Font: Change the font used to render the title/label.
    * Style: Change the display properties of the title/label.
    * Size: Change the size of the title.
- Save to Defaults: Save the display properties of this markup to be the new system defaults. The control point label visibility and properties label visibility are settings that are not included when saving defaults, as typically it is better to initialize these based on the markup type (control point labels are more useful for markups point lists, while the properties label is more useful for other markup types).
- Reset to Defaults: Reset the display properties of this markup to the system defaults. Measurements section below)
  - Visible: Whether scalar coloring should be shown or the original color of the markup
  - Active Scalar: Which scalar array to use for coloring
  - Color Table: Palette used for coloring
  - Scalar Range Mode: Method for determining the range of the scalars (automatic range calculation based on the data is the default)

Control points section

- Interaction:
  - Locked: Toggle the markups lock state (if it can be moved by mouse interactions in the viewers), which will override the individual control points lock settings.

  - Fixed list of points: Toggle whether control points can be added or removed from the markups. Control point position can still be undefined.
- 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 control point or not after the jump. Right click in the table allows jumping 2D slices to a highlighted control point (uses the center/offset radio button settings). Once checked, arrow keys moving the highlighted row will also jump slice viewers to that control point position.
- Show slice intersections: Toggle visibility of the slice intersection visibility in the 2D views.
- Buttons: These buttons apply changes to control points in the selected list.
  - Toggle visibility flag: Toggle visibility flag on all control points in the list. Use the drop down menu to set all to visible or invisible.
  - Toggle selected flag: Toggle selected flag on all control points in the 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 control points in the list. Use the drop down menu to set all to locked or unlocked.
  - Skip highlighted control points: Clear the current the position and sets the control point state to skip. When placing multiple control points, control points with this state will be skipped over for placement, moving on to the next unplaced control point.
  - Clear highlighted control points: Clear the current the position and sets the control point state to clear. The control point position can be redefined using place mode.
  - Delete the highlighted control points from the active list: After highlighting rows in the table to select control points, press this button to delete them from the list.
  - Remove all control points from the active list: Pressing this button will delete all of the control points 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 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 control points, delete the highlighted control point, jump slice viewers to that location, refocus 3D viewers to that location, or if there are other lists, to copy or move the control point to another list.
  - Selected: A check box is shown in this column, it's check state depends on if the control point is selected or not. Click to toggle the selected state. Only selected control points will be passed to command line modules.
  - Locked: An open or closed padlock is shown in this column, depending on if the control point 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 control point is visible or not. Click it to toggle the visibility state.
  - Name: A short name for this control point, displayed in the viewers as text next to the glyphs.
  - Description: A longer description for this control point, not displayed in the viewers.
  - X, Y, Z: The RAS coordinates of this control point, 3 places of precision are shown.

  - State: The current state of the control point. Clicking on the current state will cycle through the possible states.
    * Edit: The control point is currently being placed. Only one control point can be in the edit state at a time. If the state of another control point is set to edit, then the current control point state will be set to clear.
    * Skip: The control point is not currently defined, and cannot be selected for placement.
    * Restore: The control point has a defined position. Entering this state will restore the last known position of the control point.
    * Clear: The control point has not yet been placed, and can be selected for placement.
- Advanced section:
  - Move Up: Move a highlighted control point up one spot in the list.
  - Move Down: Move a highlighted control point down one spot in the list.
  - Add Control Point: Add a new unplaced control point to the selected list, creating it with an undefined position.
  - Naming:
    * Name Format: Format for creating names of new control points, using sprintf format style. `%N` is replaced by the list name, `%d` by an integer.
    * Apply: Rename all control points in this list according to the current name format, trying to preserve numbers. A quick way to re-number all the control points 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. Note that if the control point label contains multiple numbers then the first number is assumed to be part of the name and the second number is used as the control point number.
    * Reset: Reset the name format field to the default value, `%N-%d`.
  - Convert annotation fiducials: Uses annotation fiducial hierarchies to convert them to markups. Removes the annotation nodes once completed.

Measurements section

- This section lists the available measurements of the selected markup
  - `length` for line and curve
  - `angle` for angle markups
  - `curvature mean` and `curvature max` for curve markups
  - `area` for plane markups
  - `volume` for ROI markups
- In the table below the measurement descriptions, the measurements can be enabled/disabled
  - Basic measurements (e.g. length, angle) are enabled by default
  - Curve markups support curvature calculation, which is off by default
    * When turned on, the curvature data can be displayed as scalar coloring (see Display/Scalars above)

Export/Import Table section

- This section controls the import and export of markups to `vtkMRMLTableNode`.
- Operation: Select the operation, either Export or Import.
- Output/Input table: Select the input/output node.
- Advanced:
  - Export coordinate system: Choose if the markups are exported in RAS or LPS.
- Import/Export: Execute export/import operation.

Curve setting 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 points, connected with a minimal-cost path across the model mesh's edges
- Constrain to Model: Model to constrain the curve to. For curve types linear, spline, Kochanek spline, and polynomial the curves will be generated from the control points and then projected onto the surface. For `shortest distance on surface` curve type the curve is generated directly on this model.
- Surface: surface used for `shortest distance on surface` curve type and cost function that is minimized to find path connecting two control points
- Advanced:
  - Maximum projection distance: The maximum search radius tolerance defining the allowable projection distance for projecting curve points. It is specified as a percentage of the model's bounding box diagonal in world coordinate system.

Resample section

- Output node: Replace control points by curve points sampled at equal distances.
- Constrain points to surface: If a model is selected, the resampled points will be projected to the chosen model surface.
- Advanced:
  - Maximum projection distance: The maximum search radius tolerance defining the allowable projection distance for projecting resampled control points. It is specified as a percentage of the model's bounding box diagonal in world coordinate system.

Plane settings section

- Plane type: The method used to define the plane using control points.
  - Three points: Plane can be defined by placing 3 control points. The origin of the plane will be at the first control point, the x-axis will be defined by the second, and plane normal will be defined by the cross product of the vectors from the second and third points to the first point.
  - Point normal: Plane is defined using a single point and a normal vector. The normal vector will be parallel with the view direction. If placing in 3D, can be placed on a surface or volume rendering, which will align the plane normal with the surface normal.

  - Plane fit: The plane is defined by fitting a plane to any number of control points (minimum 3).
- Size mode: Method used define the size of the plane.
  - Auto: Plane size will be automatically defined based on the plane type.
  - Absolute: Plane size will be fixed.
- Size: The width and length of the plane. Can only be modified in absolute plane size mode.
- Bounds: The minimum and maximum bounds of the plane along the plane XY axes. Can only be modified in absolute plane size mode.
- Normal: Controls the plane normal direction arrow visibility and opacity.

ROI settings section

- ROI type:
  - Box: ROI is specified by a single control point in the center, axis directions, and size along each axis direction.
  - BoundingBox: ROI is specified as the bounding box of all the control points.
- Inside out: If enabled then the selected region is outside the ROI boundary. It is up to each module to decide if this information is used. For example, if ROI is used for cropping a volume then this flag is ignored, as an image cannot have a hole in it; but for example inside out can make sense for blanking out region of a volume.
- L-R, P-A, I-S Range: Extents of the ROI box along the ROI axes.
- Display ROI: show/hide the ROI in all views.
- Interactive mode: allow changing the ROI position, orientation, and size in views using interaction handles. If interaction handles are disabled, the ROI may still be changed by moving control points (unless control points are locked, too).

9.3.5 Creating Template Landmarks

In order to define a workflow in which a known list of markups must be placed, it is possible to save/load predefined markups lists to use as templates.

Defining a template:

1. Create a new point list using the toolbar, or the create button in the Markups module. You may want to exit place mode by clicking on the toolbar, or by right-clicking in a view.
2. In the Markups module, select the newly created points list
3. In the Control Points - Advanced section, create the required number of control points for the template by clicking on the Add Control Point button.
4. Set the name/description of the control points in the control points table.
5. To prevent points from being added or removed, click on the "Fixed list of points" button at the top of the control points section.
6. Export the template to file by right-clicking on the points list in the markups module, or in the subject hierarchy, and select "Export to file…".

Loading a template:

1. Drag-and-drop the template file into 3D Slicer.
2. Select the loaded template in the markups toolbar.
3. Enter place mode with "Place multiple control points" enabled.

4. Place points until all of the points in the template have been placed. Each point in the template will be selected for placement sequentially. Once all points have been placed, place mode will be automatically disabled.

9.3.6 Information for developers

See examples and other developer information in *Developer guide* and *Script repository*.

9.3.7 Related modules

- *Endoscopy* module uses control points
- This module replaced the legacy Annotations module.

9.3.8 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)
- Csaba Pinter (Pixel Medical / Ebatinca)
- Sara Rolfe (University of Washington, Seattle Children's Research Institute)

9.3.9 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.

9.4 Models

9.4.1 Overview

This module is used for changing the appearance of and organizing 3d surface models.

9.4.2 Use cases

Visualize hierarchy of models

Models can be organized into folders and display properties (visibility, color, opacity) can be overridden for an entire branch.

Folder nodes can be created by right-clicking on a folder and choosing "Create child folder".

Edit models

Models can be edited using Surface toolbox or Dynamic modeler module, or by converting to segmentation and *editing with Segment Editor module*.

9.4.3 Panels and their use

- **Model tree**:
  - **Filter by name…**: Enter a model name to filter the model hierarchy tree to items that have matching name.
  - **Hide All Models**: Turn off the visibility flag on all models in the scene. If any hierarchies have the "Force color to children" checkbox selected, the hierarchy node visibility will override model visibility settings.
  - **Show All Models**: Turn on the visibility flag on all models in the scene. If any hierarchies have the "Force color to children" checkbox selected, the hierarchy node visibility will override model visibility settings.
  - **Scene tree view**: The tree view of all models and model hierarchies in the current MRML scene.
    - Show/hide models by clicking the eye icon.
    - Change selected display properties by right-clicking on the eye icon.
    - Change color by double-clicking on the color square.
    - Organize models into a hierarchy: Create child folders by right-clicking on a folder item and drag-and-drop models into them. Visibility and opacity specified for a folder is applied to all children. Color specified for a folder is applied to all children if "Apply color to all children" option is enabled (in the right-click menu of the eye icon).
- **Information Panel**:
  - **Information**: Information about this surface model
  - **Surface Area**: The calculated surface area of the model, in square millimeters
  - **Volume**: The volume inside the surface model, in cubic millimeters
  - **Number of Points**: Number of vertices in the surface model
  - **Number of Cells**: Number of cells in the surface model
  - **Number of Point Scalars**: Shows how many arrays of scalars are associated with the points of the surface model.
  - **Number of Cell Scalars**: Shows how many arrays of scalars are associated with the cells of the surface model.
  - **Filename**: Path to the file from which this surface model was loaded and/or where it will be saved by default.
- **Display**: Control the display properties of the currently selected model in the Scene.
  - **Visibility**: Control the visibility of the model.
    - **Visibility**: The model is visible in views if this is checked.
    - **Opacity**: Control the opacity of the model. 1.0 is totally opaque and 0.0 is completely transparent. 1.0 by default.
    - **View**: Specifies which views this modelis visible in. If none are checked, the model is visible in all 2D and 3D views.
    - **Color**: Control the color of the model. Note that the lighting can alter the color. Gray by default.
- **3D Display**:
  - **Representation**: Control the surface geometry representation (points, wireframe, surface, or surface with edges).

  - **Visible sides**: This option can be used to only show front-facing elements, which may make rendering slightly faster, but the inside of the model will be no longer visible (when the viewpoint is inside the model or when the model is clipped). when it is clipped). Showing of backface elements only allows seeing inside the model.
  - **Clipping**: Enable clipped display of the model. Slice planes are used as clipping planes, according to options defined in the "Clipping planes" section at the bottom.
  - **Advanced**
    * **Point Size**: Set the diameter of the model points (if the model is a point cloud or if the representation is "Points".) The size is expressed in screen units. 1.0 by default.
    * **Line Width**: Set the width of the model lines (if the model is a polyline or representation is "Wireframe"). The width is expressed in screen units. 1.0 by default.
    * **Backface Color Offset**: Control the color of the inside of the model (which is visible when the model is open or the viewpoint is inside the model). The values correspond to hue, saturation, and lightness offset compared to the base color.
  - **Edge Color**: Control the color of the model edges (if Edge Visibility is enabled). Black by default.
  - **Lighting**: Control whether the model representation is impacted by the frontfacing light. If enabled, Ambient, Diffuse and Specular parameters are used to compute the lighting effect. Enabled by default.
  - **Interpolation**: Control the shading interpolation method (Flat, Gouraud, Phong) for the model. Gouraud by default. Gouraud and Phong shading requires surface normals. If surface normals are missing then the model will be displayed with flat shading (faceted appearance). Surface Toolbox module can compute normals for a model.
  - Material Properties: Material properties of the currently selected model
    * **Ambient**: Control the constant brightness of the model.
    * **Diffuse**: Control the amount of light that is scattered back from the model. This is direction-dependent: regions that are orthogonal to the view direction appear brighter.
    * **Specular**: Control specular reflection ("shininess") of the model surface.
    * **Power**: The specular power.
    * **Preview**: A rendering of a sphere using the current material properties.
- **Slice Display**:
  - **Visibility**: Control visibility of the model in slice views.
  - **Opacity**: Opacity of the model in slice views.
  - **Mode**: Intersection shows intersection lines of the model with the slice plane. Projection projects the entire model into the slice plane. Distance encoded projection makes the projected model colored based on the distance from the slice plane.
  - **Line width**: Width in pixels of the intersection lines of the model with the slice planes.
  - **Color table**: Specifies colors for "Distance encoded projection" mode.
  - **Scalars**: Scalar overlay properties
    * **Visible**: Color the model using the active scalar.
    * **Active Scalar**: A drop down menu listing the current scalar overlays associated with this model and shows which one is currently active. Most models will have normals, Freesurfer surface models can have multiple scalar overlay files associated with them (e.g., lh.sulc, lh.curv).

        * **Color Table**: Select a color look up table used to map the scalar overlay's values (usually in the range of 0.0 to 1.0) to colors. There are built in color maps that can be browsed in the Colors module. The most commonly used color map for FreeSurfer scalar overlays is the GreenRed one. Legacy color maps from Slicer2 include Grey, Iron, Rainbow, etc. Those color maps with "labels" in their names are usually discrete and don't work well for the continuous scalar overlay ranges.
        * **Scalar Range Type**: Select which scalar range to use:
            - Manual: range is set manually
            - Data scalar range: range is set to the value range of the active scalar
            - Color table: use range specified in the color table. Useful for showing several nodes using the same color mapping.
            - Data type: range is set to the possible range of the active scalar's data type. This is only useful mostly for 8-bit scalars.
            - Direct color mapping: if active scalar has 3 or 4 components then those are interpreted as RGB or RGBA values.
        * **Displayed Range**: currently used scalar range.
        * **Threshold**: if enabled, then regions of the model that are outside the specified range will be hidden.
    - **Clipping Planes**:
        * **Clip selected model**: enable clipping for the selected model.
        * **Clipping Type**: When more than one slice plane is used, this option controls if it's the union or intersection of the positive and/or negative spaces that is used to clip the model. The parts of the model inside the selected space is kept, parts outside of the selection are clipped away.
        * **Red/Yellow/Green Slice Clipping**: Use the positive or negative space defined by the Red/Yellow/Green slice plane to clip the model. Positive side is toward the Superior/Right/Anterior direction. Keeps the part of the model in the selected space, clips away the rest.
    - **Clip only whole cells when clipping**: choose between straight cut (cuts over cells) or crinkle cut (clips the model along cell boundaries).

9.4.4 Contributors

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

9.4.5 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 is partially supported by the Air Force Research Laboratories (AFRL).

9.5 Scene Views

9.5.1 Overview

Scene views are a convenience tool for organizing multiple 'live views' of the data in your scene.

You can create any number of views and control parameters such as the 3D view, model visibility, window layout, and other parameters.

This can be used to set up a series of predefined starting points for looking at portions of your data in detail.

For example, you may have one overview scene which shows an external view of the body along with interior views with the skin surface turned off and slice planes visible to highlight a tumor location.

9.5.2 Panels and their use

- **Scene Views**: Create, edit, restore scene views
  - **Create and Edit**: Create and delete scene views, move them up and down
    - **Create a 3D Slicer Scene View**: Click on the camera button to create a new scene view
    - **Move selected view up**: Move the highlighted scene view in the table up one row
    - **Move selected view down**: Move the highlighted scene view in the table down one row
    - **Delete selected view**: Click on the garbage can button to delete the highlighted scene view
    - **Table of Scene Views**: A table showing the scene views, one per line

9.5.3 Contributors

Nicole Aucoin (SPL, BWH), Wendy Plesniak (SPL, BWH), Daniel Haehn (UPenn), Kilian Pohl (UPenn)

9.5.4 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.

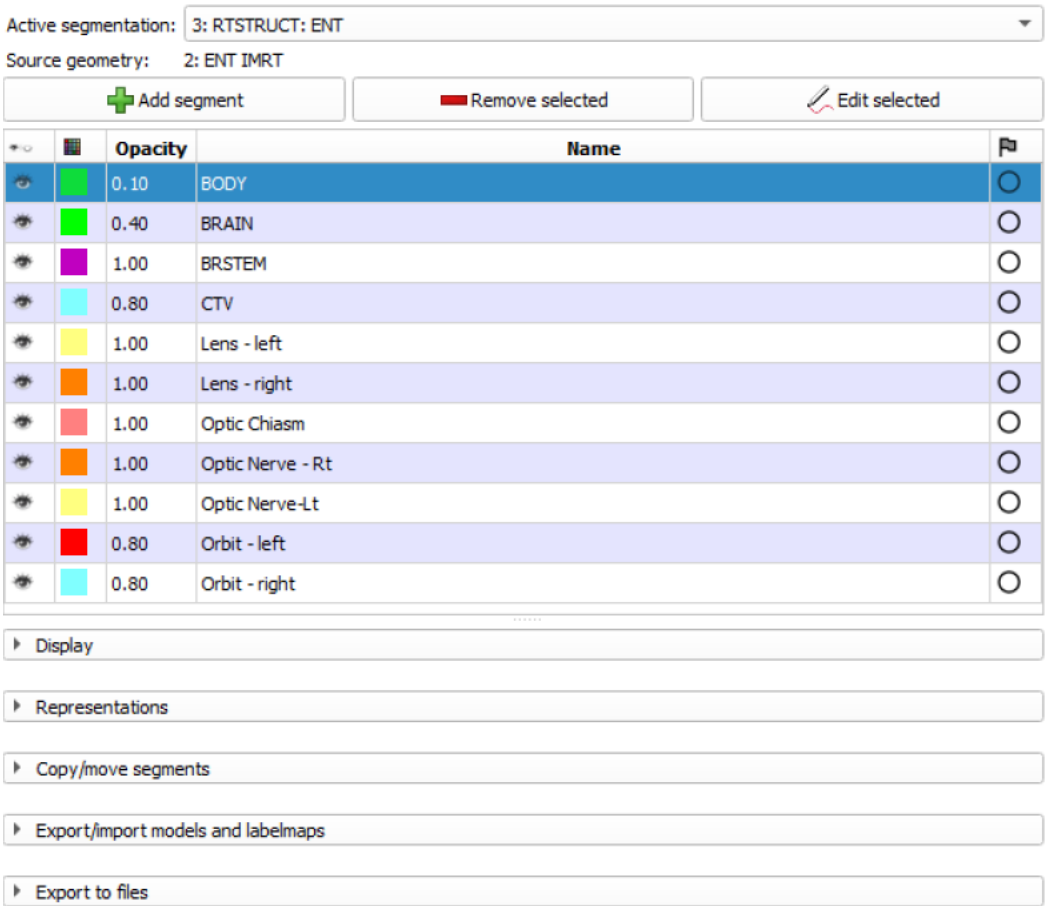
9.6 Segmentations

9.6.1 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 paper *Cs. Pinter, A. Lasso, G. Fichtinger, "Polymorph segmentation representation for medical image computing", Computer Methods and Programs in Biomedicine, Volume 171, p19-26, 2019* (pdf, DOI) and in the related PhD dissertation.



9.6.2 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`
- Optional: if a color table (specifying name and color for each label value) is available then load that first into the application and then select it as `Color node` in the `Options` section. Specification of the color table file format is available *here*.
- Click `OK`

> **Tip**
>
> To avoid the need to always manually select `Segmentation`, save the `.nrrd` file using the `.seg.nrrd` file extension. It makes Slicer load the image as a segmentation by default.

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

- 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 click `Show 3D`. Alternatively, go to `Data` module and drag-and-drop the segmentation into each view where you want to see them - if the segmentation is dragged into a 3D view then it will be shown there 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`

If the model contains very thin and delicate structures then default resolution for binary labelmap representation may be not sufficient for editing. Default resolution is computed so that the labelmap contains a total of approximately 256x256x256 voxels. To make the resolution finer:

- Go to Segmentations module
- In Representations section, click Binary labelmap -> Create, then Update
- In the displayed popup:
  - In Conversion path section, click Closed surface -> Binary labelmap
  - In Conversion parameters section, set oversampling factor to 2 (if this is not enough then you can try 2.5, 3, 4, . . .) - larger values increase more memory usage and computation time (oversampling factor of 2x increases memory usage by 2^3 = 8x).
  - Click Convert

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`

Editing a segmentation imported from model (surface mesh) file

Selection of a `source volume` is required for editing a segmentation. The source volume specifies the geometry (origin, spacing, axis directions, and extents) of the voxel grid that is used during editing.

If no volume is available then it can be created by the following steps:

- Go to `Segment editor` module

- Click `Specify geometry` button (on the right side of `Source volume` node selector)
- For `Source geometry` choose the segmentation (this specifies the extents, i.e., the bounding box so that the complete object is included)
- Adjust `Spacing` values as needed. It is recommended to set the same value for all three axes. Using smaller values preserve more details but at the cost of increased memory usage and computation time.
- Click `OK`
- When an editing operation is started then the Segment Editor will ask if the source representation should be changed to binary labelmap. Answer `Yes`, because binary labelmap representation is required for editing.

> **Note**
>
> Certain editing operations are available directly on models, without converting to segmentation. For example, using Surface Toolbox and Dynamic Modeler modules.

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. Using a reference volume for labelmap export may result in the segmentation being cropped if some regions are outside of the new geometry. A confirmation popup will be displayed before the segmentation is cropped.

Export segmentation to labelmap volume file

If source representation of a a segmentation node is binary labelmap then the segmentation will be saved in standard NRRD file format. This is the recommended way of saving segmentation volumes, as it saves additional metadata (segment names, colors, DICOM terminology) in the image file in custom fields and allows saving of overlapping segments.

For exporting segmentation as NRRD or NIFTI file for external software that uses 3D labelmap volume file + color table file for segmentation storage:

- 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)
- In `File format` selector choose `NRRD` or `NIFTI`. NRRD format is recommended, as it is a simple, general-purpose file format. For neuroimaging, NIFTI file format may be a better choice, as it is the most commonly used research file format in that field.
- Optional: choose Reference volume if you want your segmentation to match a selected volume's geometry (origin, spacing, axis directions) instead of the current segmentation geometry
- Optional: check `Use color table values` checkbox and select a color table to set voxel values in the exported files from values specified in the color table. The label value is index of the color table entry that has the same name as the segment name. If a color table is not specified then, voxel values are based on the order of segments in the segment list (voxels that are outside of all segments are set to 0, voxels of the first segment are set to 1, voxels of the second segment are set to 2, etc).
- Set additional options (destination folder, compression, etc.) as needed
- Click `Export`

Labelmap volumes can be created in any other formats by *exporting segmentation to labelmap volume* then in application menu, choose `File` / `Save`.

Using a reference volume for labelmap export may result in the segmentation being cropped if some regions are outside of the new geometry. A confirmation popup will be displayed before the segmentation is cropped.

Create new representation in segmentation (conversion)

The supported representations are listed in the Representations section. Existing representations are marked with a green tick, the source representation is marked with a gold star. The source representation is the editable (for example, in Segment Editor module) and it is the source of all conversions.

- To create a representation, click on the Create button in the corresponding row. To specify a custom conversion path or parameters (reference geometry for labelmaps, smoothing for surfaces, etc.), click the down-arrow button in the "Create" button and choose "Advanced create…", then choose a conversion path from the list at the top, and adjust the conversion parameters in the section at the bottom.
- To update a representation (re-create from the source representation) using custom conversion path or parameters, click the "Update" button in the corresponding row.
- To remove a representation, click the down-arrow button in the "Update" button then choose "Remove".

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

- Make sure the source representation is `binary labelmap` (as the source representation is used when exporting into DICOM, and currently only binary labelmap representation is supported). The source representation can be set in `Segmentations` module, `Representations` section. If the yellow star (that designates the source representation) is in the `binary labelmap` row then the representation is already good, no further action is needed. If the yellow star is in another row then click `Make source` button in the `binary labelmap` row (if that button is not shown then click `Create` button first).
- DICOM Segmentation Object (`DICOMSegmentation` export type) is recommended for storing segmentations in DICOM. However, if segmentations will be used for radiation therapy then it may be necessary to export to DICOM RT Structure Set. RT Structure Set export type becomes available if `SlicerRT` extension is installed. In general, RT Structure Sets are not recommended for storing segmentations, as they cannot store arbitrarily complex 3D shapes.
- If the segmented image was not loaded from the DICOM database (for example, if the image was loaded from a NRRD or NIFTI file) then a DICOM image needs to be created in the database and the segmentation must be updated to reference this image (this is necessary because a DICOM requires a segmentation to be associated with a DICOM image) by following these steps:
  - *export the source volume to DICOM* (agree to create patient and study when asked)
  - load the exported volume from the DICOM database
  - in `Segment Editor` module, click `Specify geometry` button, choose the loaded volume as `Source geometry`, and click OK
- Go to Data module, right-click on the segmentation, choose `Export to DICOM`, and follow *these instructions*.

9.6.3 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 selected 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 source 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 source. The source 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 selected segmentation
  - Right panel shows the external data container
  - The arrow buttons van 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

9.6.4 Tutorials

- Segmentation tutorials

9.6.5 Limitations

- When segmentation is displayed in 2D views using `Closed surface` representation (either because this is the only available representation or because this representation is chosen to be shown in `Segmentations` module: `Display / Advanced / Representation in 2D views -> Closed surface`) then filling of the contours may appear incomplete and/or inverted. This is just a rendering error and does not affect the segmentation content. To avoid seeing such rendering artifacts, create `Binary labelmap` representation and choose `Representation in 2D views -> Binary labelmap`.

9.6.6 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*

9.6.7 Related modules

- *Segment Editor* module is for editing segments of a segmentation node

9.6.8 References

- Development notes

9.6.9 Contributors

Authors:

- Csaba Pinter (PerkLab, Queen's University)
- Andras Lasso (PerkLab, Queen's University)
- Kyle Sunderland (PerkLab, Queen's University)

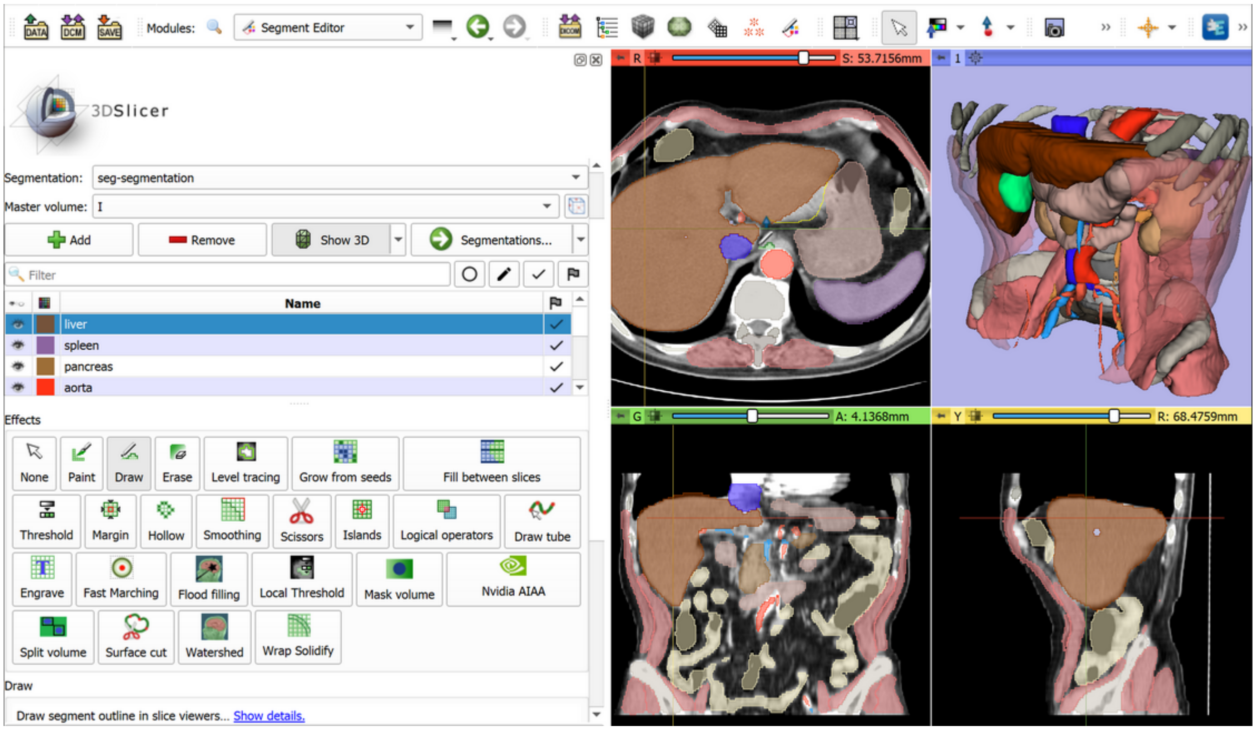
9.6.10 Acknowledgements

This work is funded in part 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.

9.7 Segment editor

This is a module is for specifying segments (structures of interest) in 2D/3D/4D images. Some of the tools mimic a painting interface like photoshop or gimp, but work on 3D arrays of voxels rather than on 2D pixels. The module offers editing of overlapping segments, display in both 2D and 3D views, fine-grained visualization options, editing in 3D views, create segmentation by interpolating or extrapolating segmentation on a few slices, editing on slices in any orientation.

Segment Editor does not edit labelmap volumes or models, but segmentations can be easily converted to/from labelmap volumes and models using the Import/Export section of *Segmentations* module.



9.7.1 How to cite

To cite the Segment Editor in scientific publications, you can cite *3D Slicer* and the Segment Editor paper: *Cs. Pinter, A. Lasso, G. Fichtinger, "Polymorph segmentation representation for medical image computing", Computer Methods and Programs in Biomedicine, Volume 171, p19-26, 2019* (pdf, DOI). Additional references to non-trivial algorithms used in Segment Editor effects are provided below, in the documentation of each effect.

9.7.2 Keyboard shortcuts

The following keyboard shortcuts are active when you are in the Segment 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.

9.7.3 Tutorials

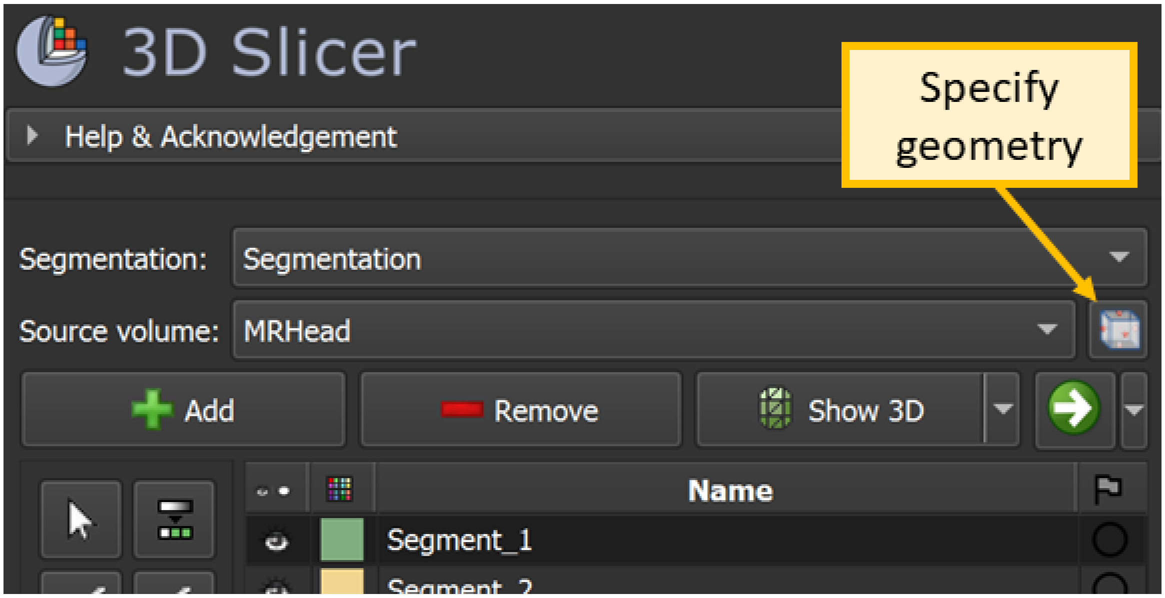
- Segmentation tutorials

9.7.4 Panels and their use

Main options

- **Segmentation**: Choose the segmentation to edit
- **Source volume**: Choose the volume to segment. The source 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 source volume is used by all editor effects that uses intensity of the segmented volume (e.g., thresholding, level tracing). The source volume can be changed at any time during the segmentation process. Note: changing the source 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 source 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 extens, an ROI (region of interest) node can be created and selected as source geometry. If the new geometry will crop a region from the existing

segments, a warning icon will be displayed beside the "Pad output" checkbox. If the "Pad output" is checked, the extent will be expanded so that it contains both the existing segments and the new reference geometry.



- **Add**: Add a new segment to the segmentation and select it.
- **Remove**: Select the segment you would like to delete then click Remove segment to delete from the segmentation.
- **Show 3D**: 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

This table displays the list of all segments.

Table columns:

- **Visibility** (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.
- **State** (flag icon): This column can be used for setting the editing status of each segment that can be used for filtering the table or mark segments for further processing.
  - `Not started`: default starting state, indicates that editing has not happened yet.
  - `In progress`: when a "not started" segment is edited its state is automatically changed to this
  - `Completed`: user can manually select this state to indicate that the segmentat is complete
  - `Flagged`: user can manually select this state for any custom purpose, for example to bring the segment into the attention of an expert reviewer
- **Layer**: advanced information, displays the 3D layer index when there are overlapping segments. Hidden By default, can be shown by right-clicking on the table and enabling `Show layer column`.

Filter bar: It can be used for finding segments when editing segmentations that contain a large number of segments. By default the filter bar may not be shown, right-click on the segments table and click `Show filter bar` to show/hide it.

- Filter: filter by segment name
- Segment state toggle buttons: only segments of the selected states will be displayed in the segment list

Effects section

- **Effect toolbar**: 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.

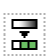
Masking options

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 source 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.

9.7.5 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.

 Threshold

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

 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
- Left double-click to add a point and fill the contour. Alternatively, right click to fill the current contour without adding any more points.

Note

Scissors effect can be also used for drawing. Scissors effect works both in slice and 3D views, can be set to draw on more than one slice at a time, can erase as well, can be constrained to draw horizontal/vertical lines (using rectangle mode), etc.

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. Source 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:

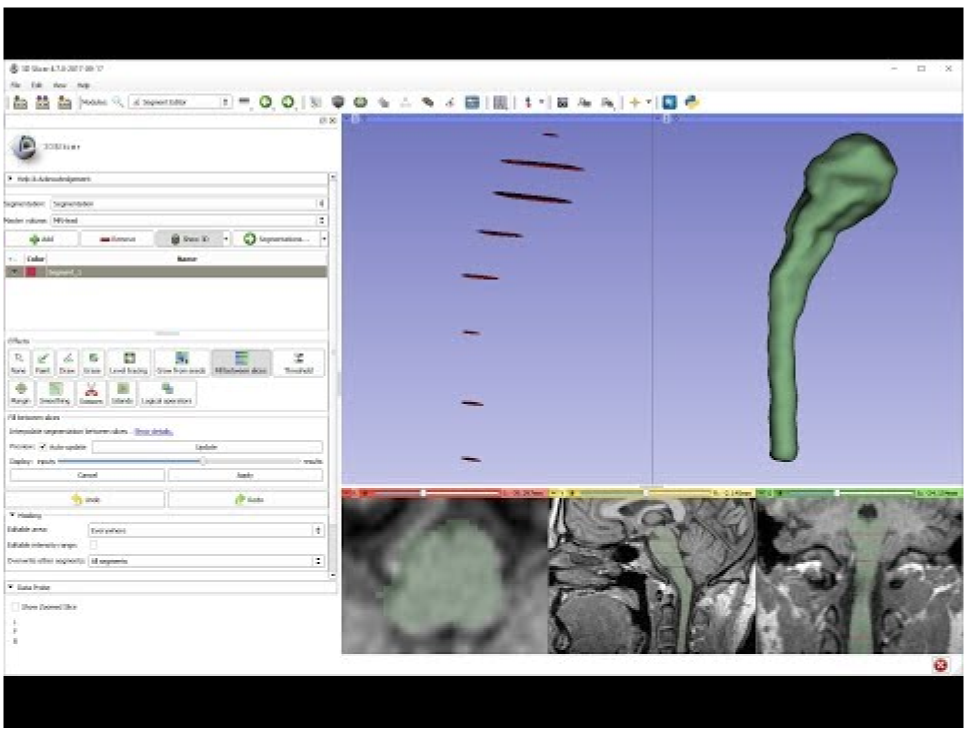
- Only visible segments are used by this effect.
- At least two segments are required.
- If a part of a segment is erased or painting is removed using Undo (and not overwritten by another segment) then it is recommended to cancel and initialize. The reason is that effect of adding more information (painting more seeds) can be propagated to the complete segmentation, but removing information (removing some seed regions) will not change the complete segmentation.

- The method uses an improved version of the grow-cut algorithm described in *Liangjia Zhu, Ivan Kolesov, Yi Gao, Ron Kikinis, Allen Tannenbaum. An Effective Interactive Medical Image Segmentation Method Using Fast GrowCut, International Conference on Medical Image Computing and Computer Assisted Intervention (MICCAI), Interactive Medical Image Computing Workshop, 2014.*

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. Source 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.
- The method does not use the source volume, only the shape of the specified segments.
- The method uses *ND morphological contour interpolation algorithm* described in this paper: https://insight-journal.org/browse/publication/977

Margin

Grows or shrinks the selected segment by the specified margin.

By enabling `Apply to visible segments`, all visible segments of the segmentation will be processed (in the order of the segment list).

Hollow

Makes the selected visible segment hollow by replacing the segment with a uniform-thickness shell defined by the segment boundary.

By enabling `Apply to visible segments`, all visible segments of the segmentation will be processed (in the order of the segment list).

Smoothing

Smoothes segments by filling in holes and/or removing extrusions.

By default, the current segment will be smoothed. By enabling `Apply to visible segments`, all visible segments of the segmentation will be smoothed (in the order of the segment list). This operation may be time-consuming for complex segmentations. The `Joint smoothing` method always smoothes all visible segments.

By clicking `Apply` button, the entire segmentation is smoothed. To smooth a specific region, left click and drag in any slice or 3D view. Same smoothing method and strength is used as for the whole-segmentation mode (size of the brush does not affect smoothing strength, just makes it easier to designate a larger region).

Available methods:

- Median: removes small extrusions and fills small gaps while keeps smooth contours mostly unchanged. Applied to selected segment only.
- Opening: removes extrusions smaller than the specified kernel size. Does not add anything to the segment. Applied to selected segment only.
- Closing: fills sharp corners and holes smaller than the specified kernel size. Does not remove anything from the segment. Applied to selected segment only.
- Gaussian: smoothes all details. Strong smoothing as achievable, but tends to shrink the segment. Applied to selected segment only.
- Joint smoothing: smoothes multiple segments at once, preserving watertight interface between them. If segments overlap, segment higher in the segments table will have priority. Applied to all visible segments.

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

By enabling `Apply to visible segments`, all visible segments of the segmentation will be processed (in the order of the segment list).

Islands

Use this tool to process "islands", i.e., connected regions that are defined as groups of non-empty voxels which touch each other but are surrounded by empty voxels.

- `Keep largest island`: keep the larges connected region.
- `Remove small islands`: keep all connected regions that are larger than `minimum size`.
- `Split islands to segments`: create a unique segment for each connected region of the selected segment.
- `Keep selected island`: after selecting this mode, click in a non-empty area in a slice view to keep that region and remove all other regions.
- `Remove selected island`: after selecting this mode, click in a non-empty area in a slice view to remove that region and preserve all other regions.
- `Add selected island`: after selecting this mode, click in an empty area in a slice view to add that empty region to the segment (fill hole).

Logical operators

Apply basic copy, clear, fill, and Boolean operations to the selected segment(s). See more details about the methods by clicking on "Show details" in the effect description in Segment Editor.

Mask volume

Blank out inside/outside of a segment in a volume or create a binary mask. Result can be saved into a new volume or overwrite the input volume. This is useful for removing irrelevant details from an image (for example remove patient table; or crop the volume to arbitrary shape for volume rendering) or create masks for image processing operations (such as registration or intensity correction).

- `Operation`:
  - `Fill inside`: set all voxels of the selected volume to the specified `Fill value` inside the selected segment
  - `Fill outside`: set all voxels of the selected volume to the specified `Fill value` outside the selected segment
  - `Fill inside and outside`: create a binary labelmap volume as output, filled with `Outside fill value` and `Intside fill value`. Most image processing operations require background (outside, ignored) region to be filled with 0 value.
- `Soft edge`: if set to >0 then transition between the inside/outside the mask is gradual. The value specifies the standard deviation of the Gaussian blurring function. Larger value results in softer transition.
- `Input volume`: voxels of this volume will be used as input for the masking. Geometry and voxel type of the output volume will be the same as this volume's.
- `Output volume`: this volume will store the result of the masking. While it can be the same as the input volume, generally it is better to use a different output volume, because then the options can be adjusted and mask can be recomputed multiple times.

9.7.6 Tips

- 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).

9.7.7 Frequently asked questions

Cannot paint outside some boundaries

When you create a segmentation, internal labelmap geometry (extent, origin, spacing, axis directions) is determined from the source volume *that you choose first*. You cannot paint outside this extent.

If you want to extend the segmentation to a larger region then you need to modify segmentation's geometry using "Specify geometry" button.

Cannot edit the segments

Masking settings (visible at the bottom of the effect options when any effect is selected) may prevent modifications of segments. If painting, erasing, etc. "does not work" then make sure your masking settings are set to default:

- Editable area: everywhere
- Editable intensity range: unchecked (it means that the segmentation is editable, regardless of the intensity values of the source volume)

Segmentation is not accurate enough

If details cannot be accurately depicted during segmentation or the exported surface has non-negligible errors (there are gaps or overlap between segments), then it is necessary to reduce the segmentation's spacing (more accurately: spacing of the internal binary labelmap representation in the segmentation node). *Spacing* is also known as *voxel size* or may be referred to as *resolution* (which is inverse of spacing - higher resolution means smaller spacing).

As a general rule, segmentation's spacing needs to be 2-5x smaller than the size of the smallest relevant detail or the maximum acceptable surface error in the generated surface.

By default, segmentation's spacing is set from the *source volume that is selected first after the segmentation is created*. If the first selected source volume's resolution is not sufficient or highly anisotropic (spacing value is not the same along the 3 axes) then one of the following is recommended:

- Option A. Crop and resample the input volume using *Crop volume* module before starting segmentation. Make spacing smaller (small enough to represent all details but not too small to slow things down and consume too much memory) and isotropic by reducing *Spacing scale* and enabling *Isotropic spacing*. Also adjust the region of interest to crop the volume to minimum necessary size to minimize memory usage and make editing faster.
- Option B. Click *Specify geometry* button in Segment Editor any time to specify smaller spacing. After this smooth segments using *Smoothing* effect. *Joint smoothing* method is recommended as it can smooth all the segments at once and it preserves boundaries between segments. *Joint smoothing* flattens all the processed segments into one layer, so if the segentation contains overlapping segments then segment in several steps, in each step only show a set of non-overlapping segments (or use any of the other smoothing methods, which only operate on the selected segment).

Generated surface contains step artifacts

If 3D surface generated from the segmentation contains step artifacts (looks "blocky") then it is necessary to increase smoothing and/or reduce segmentation's spacing.

Users need to choose between having *smooth surface* vs. *no gaps or overlap between segments*. It is impossible to have both. To achieve the desired results, there are two parameters to control: segmentation's spacing and surface smoothing factor:

1. Choose spacing that allows accurate segmentation (*see* Segmentation is not accurate enough *section above*)

2. Choose smoothing value that removes staircase artifacts but still preserves all details that you are interested in.
3. If you find that the surface smoothing value that is high enough to remove staircase artifacts also removes relevant details then further reduce spacing.

Paint affects neighbor slices or stripes appear in painted segments

Segment Editor allows editing of segmentation on slices of arbitrary orientation. However, since edited segments are stored as binary labelmaps, "striping" artifacts may appear on thin segments or near boundary of any segments. See *Oblique segmentation* segmentation recipe for more details and instructions on how to deal with these artifacts.

9.7.8 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.

9.7.9 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.

9.7.10 Information for Developers

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

9.7.11 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)

9.7.12 Acknowledgements

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

9.8 Welcome

9.8.1 Overview

This module is intended as a welcome screen and basic overview of the Slicer software.

By default Slicer shows this module at startup, but this can be changed by selecting a different module in application settings -> Modules -> Default startup module.

9.8.2 Panels and their use

Updates

This section allows configuring application and extension update checks.

If `Check for updates - Automatically` checkbox is checked then it means that the application periodically queries the Slicer download and extension servers to determine if updates are available for the main application or for any installed extensions. If the checkbox is gray (neither checked nor unchecked) then it means that either application or extension update check is enabled but not both. The servers may collect anonymous statistics of these queries.

`Check now` button can be used to force an immediate check for application and extension updates. This may be useful if automatic update checks are disabled or extensions may have been updated since the last automatic update check.

9.9 Transforms

9.9.1 Overview

This module is used for creating, editing, and visualization of spatial transformations. Transformations are stored in transform nodes and define position, orientation, and warping in the world coordinate system or relative to other nodes, such as volumes, models, markups, or other transform nodes.

See a summary of main features demonstrated in this video.

Supported transform types:

- linear transform: rigid, scaling, shearing, affine, etc. specified by a 4x4 homogeneous transformation matrix
- b-spline transform: displacement field specified at regular grid points, with b-spline interpolation

- grid transform: dense displacement field, with trilinear interpolation
- thin-plate splines: displacements specified at arbitrarily placed points, with thin-plate spline interpolation
- composite transforms: any combinations of the transforms above, in any order, any of them optionally inverted

9.9.2 Use cases

Create a transform

Transform node can be created in multiple ways:

- Method A: In Data module's Subject hierarchy tab, right-click on the "Transform" column and choose "Create new transform". This always creates a general "Transform".
- Method B: In Data module's Subject hierarchy tab, right-click on the "Visibility" column of any transformable node and choose "Interaction" to create a parent transform and display the interaction handles (See *Interaction section*). This always creates a general "Transform".
- Method C: In Data module's Transform hierarchy tab, right-click on an item and choose "Insert transform". This always creates a "Linear transform". Advantage of this method is that it is easy to build and overview hierarchy of transforms.
- Method D: In Transforms module click on the "Transform" node selector and choose one of the "Create new…" options.

How to choose transform type: Create "Linear transform" if you only work with linear transforms, because certain modules only allow you to select this node type as input. In other cases, it is recommended to create the general "Transform" node. Multiple transform node types exist because earlier Slicer could only store a simple transformation in a node. Now a transform node can contain any transformation type (linear, grid, bspline, thin-plate spline, or even composite transformation - an arbitrary sequence of any transformations), therefore transform node types only differ in their name. In some modules, node selectors only accept a certain transform node type, therefore it may be necessary to create that expected transform type, but in the long term it is expected that all modules will accept the general "Transform" node type.

Apply transform to a node

"Applying a transform" means setting parent transform to a node to translate, rotate, and/or warp it. If the parent transform is changed then the position or shape of the node is automatically updated. If the transform is removed then original (non-transformed) state of the node is restored.

A transform can be applied to a node in multiple ways:

- Method A: In Data module's Subject hierarchy tab, right-click on the "Transform" column and choose a transform (or "Create new transform"). The transform can be interactively edited in 3D views by right-clicking on "Transform" column and choosing "Interaction".
- Method B: In Data module's Transform hierarchy tab, drag the nodes under a transform.
- Method C: In Transforms module's "Apply transform" section move nodes from the Transformable list to the Transformed list by selecting them and click the arrow button between them.

Parent transform can be set for transform nodes, thereby creating a hierarchy (tree) of transforms. This transform tree is displayed in Data module's Transform hierarchy tab.

Harden transform on a node

"Hardening a transform" means permanently modify the node according to the currently applied transform. For example, coordinate values of model points are updated with the transformed coordinates, or a warped image voxels are resampled. After hardening the transform, the node is no longer associated with the transform.

A transform can be hardened on a node in multiple ways:

- Method A: In Data module, right-click on Transform column and click "Harden transform"
- Method B: In Transforms module's "Apply transform" section, click the harden button (below the left arrow button).

If non-linear transform is hardened on a volume then the volume is resampled using the same spacing and axis directions as the original volume (using linear interpolation). Extents are updated to fully contain the transformed volume. To specify a different image extent or resolution, one of the image resampling modules can be used, such as "Resample Scalar/Vector/DWI volume".

Modify transform

- Invert: all transforms can be inverted by clicking "Invert" button in Transforms module's Edit section. This is a reversible operation, as the transform's internal representation is not changed, just a flag is set that the transform has to be interpreted as its inverse.
- Split: a composite transform can be split to multiple transform nodes (so that each component is stored in a separate transform node) by clicking "Split" button in Transforms module's Edit section.
- Change translation/rotation:
  - linear transforms can be edited using translation and rotation sliders Transforms module's Edit section. "Translation in global or local reference frame" button controls if translation is performed in the parent coordinate system or the rotated coordinate system.
  - translation and rotation of a linear transform can be interactively edited in 3D by enabling "Visible in 3D view" in Transform's module Display / Interaction section. See *Interaction section*.
- Edit warping transform: to specify/edit a warping transform that translates a set of points to specified positions, you can use *semi-automatic registration methods*

Compute transform

Transforms are usually computed using *spatial registration tools*.

Save transform

Transforms are saved into ITK transform file format. ITK always saves transform "from parent", as this is the direction that is necessary for transforming images.

If a transform node in Slicer has a transformation with "to parent" direction (e.g., it was computed like that or a "from parent" transform got inverted) then:

- linear transforms: the transform is automatically converted to "from parent" direction.
- warping transforms: the transform is saved as special inverse transform class that most ITK-based applications cannot interpret. If compatibility with other applications is needed, the transform can be converted to a displacement field before saving.

A quick way to export a linear transform to another software or to text files is to copy the homogeneous transformation matrix values to the clipboard by clicking "Copy" button in Edit section in Transforms module.

Load transform

Drag-and-drop the transform file to the application window and make sure "Transform" is selected in Description column.

MetaImage (mha), NIFTI (nii) vector volumes can be loaded as displacement field (grid) transform. The volume and its vectors are expected to be stored in LPS coordinate system (during reading they are converted to RAS to match coordinate system conventions in Slicer).

A quick way to import a linear transform from another software or from text files is to copy the homogeneous transformation matrix values to the clipboard, and click "Paste" button in Edit section in Transforms module.

Visualize transform

Transforms can be visualized in both 2D and 3D views, as glyphs representing the displacement vectors as arrows, cones, or spheres; regular grids that are deformed by the transform; or contours that represent lines or surfaces where the displacement magnitude has a specific value. See documentation of *Display section* for details. Linear transforms can also be interacted with using an interaction handle widget. See *Interaction section*

Interaction

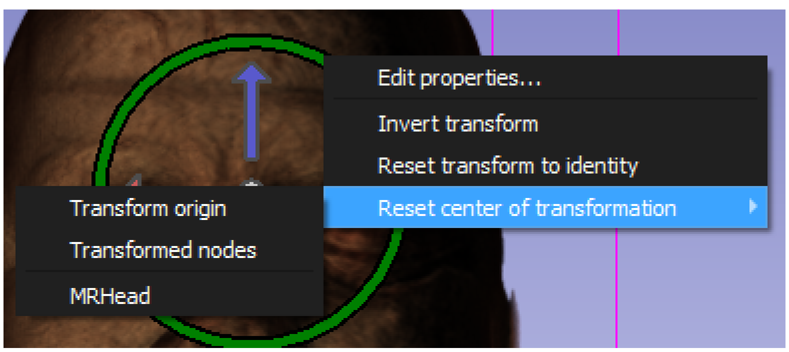
Interaction keybinds

For macOS, refer to *alternate macOS keybindings*.

Note

In slice views, if "Enable view plane translation anywhere" is enabled (Display section / Interaction / More options), left-click and drag anywhere in the slice view adjusts the translation, instead of only when clicking precisely on the small handle at the center of the transform. While this option is enabled, that handle is drawn as a crosshair (with an empty center) instead of a filled circle. Dragging this crosshair modifies the position of the center of rotation.

Context menu



- Edit properties: Switch to the Transforms module and select the specified transform node.
- Invert transform: Invert the transform matrix.
- Reset transform to identity: Reset the transform matrix back to an identity transform.
- Reset center of transformation:
  - Transform origin: Reset the center of transformation to [0,0,0] in the local coordinate system.
  - Transformed nodes: Reset the center of transformation to the bounds center of all nodes that are transformed.
  - Individual nodes: Reset the center of transformation to the center of the bounds for a specific node.

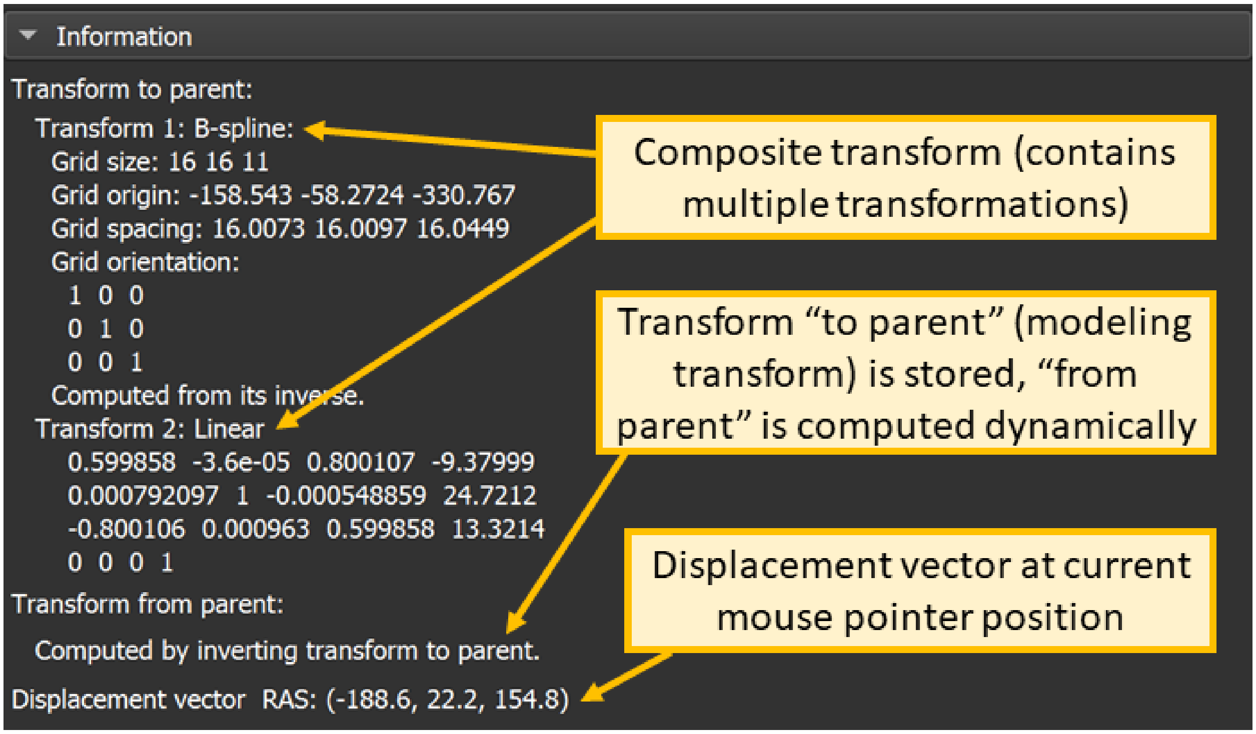
9.9.3 Panels and their use

Transform: Select the transform node to display, control and edit.

Information

Displays details about what transformation(s) a transform node contains, such as type and direction ("to parent" or "from parent).

It also displays displacement value at the current mouse pointer position (in slice views, it is enough to just move the mouse pointer; in 3D views, Shift key must be held down while moving the mouse).

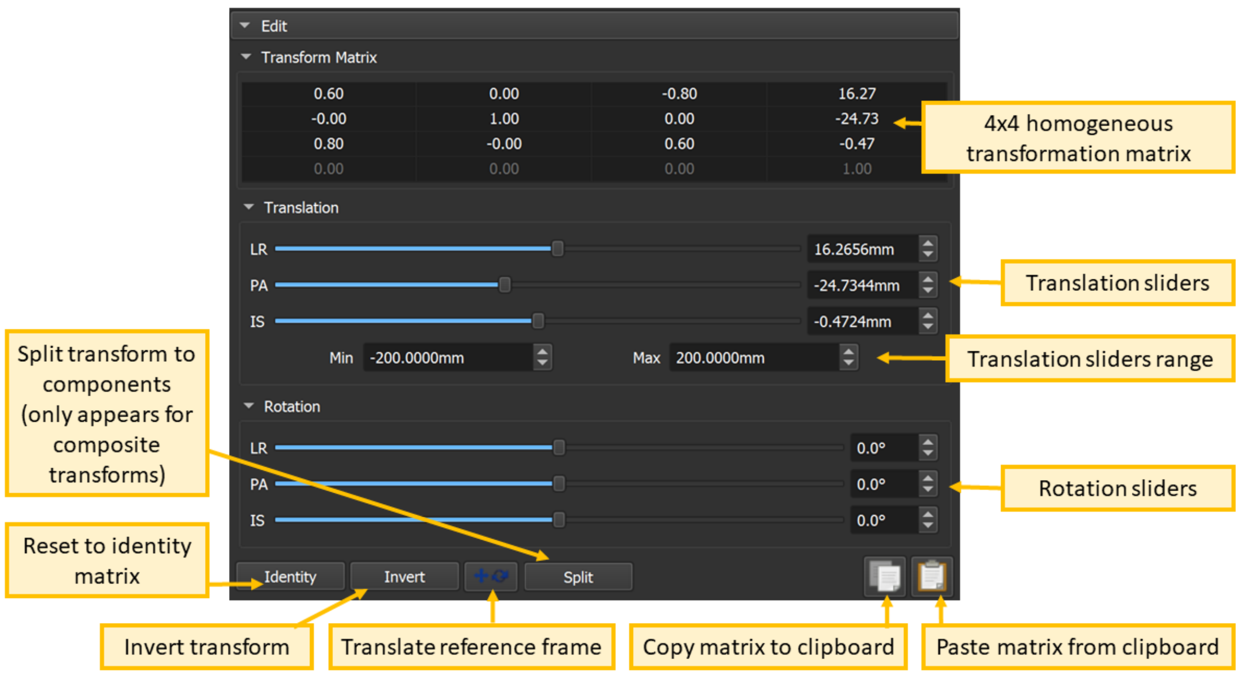


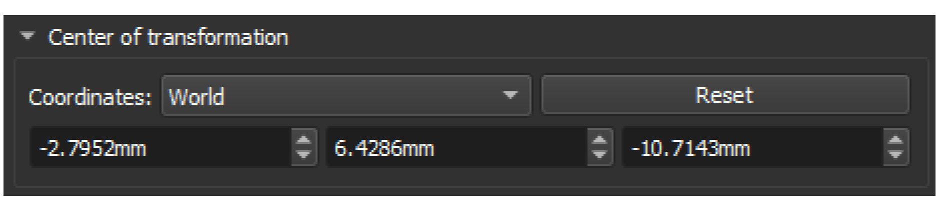
Edit

- Transform matrix: 4x4 homogeneous transformation matrix. Each element is editable on double click. Type Enter to validate change, Escape to cancel or Tab to edit the next element. First 3 columns of the matrix specifies an axis of the transformed coordinate system. Scale factor along an axis is the column norm of the corresponding column. Last column specifies origin of the transformed coordinate system. Last row of the transform is always [0, 0, 0, 1].
- Translation and rotation sliders:
  - Translation: Apply LR, PA, and IS translational components of the transformation matrix in RAS space (in mm). Min and Max control the lower and upper bounds of the sliders.
  - Rotation: Apply LR, PA, and IS rotation angles (degrees) in the RAS space. Rotations are concatenated.
  - Note: Linear transform edit sliders only show relative translation and rotation because a transformation can be achieved using many different series of transforms. To make this clear to users, only one transform slider can be non-zero at a time (all previously modified sliders are reset to 0 when a slider is moved). The only exception is translation sliders in "translate first" mode (i.e., when translation in global/local coordinate system button is not depressed): in this case there is a only one way how a specific translation can be achieved, therefore transform sliders are not reset to 0. An rotating dial widget would be a more

appropriate visual representation of the behavior than sliders, but slider is chosen because it is a standard widget and users are already familiar with it.

- Translation reference frame: Determines what coordinate system the translation (specified by translation sliders) is specified in - in the parent coordinate system or in the rotated coordinate system.
- Identity: Resets transformation matrix to identity matrix.
- Invert: Inverts the transformation matrix.
- Split: Split a composite transform so that each of its component is stored in a separate transform node.
- Copy: Copy the homogeneous transformation matrix values to the clipboard. Values are separated by spaces, each line of the transform is in a separate line.
- Paste: Paste transformation matrix from clipboard. Values can be separated by spaces, commas, or semicolons, which allows copy/pasting numpy array from Python console or matrix from Matlab.
- Center of transformation: The center of transformation of the selected transform. This is the position that the sliders will rotate around, and where the interaction handles are positioned. The center of transformation can be edited/displayed in world or local coordinates. Clicking the 'Reset' button will reset the center of transformation to the local coordinates [0,0,0].

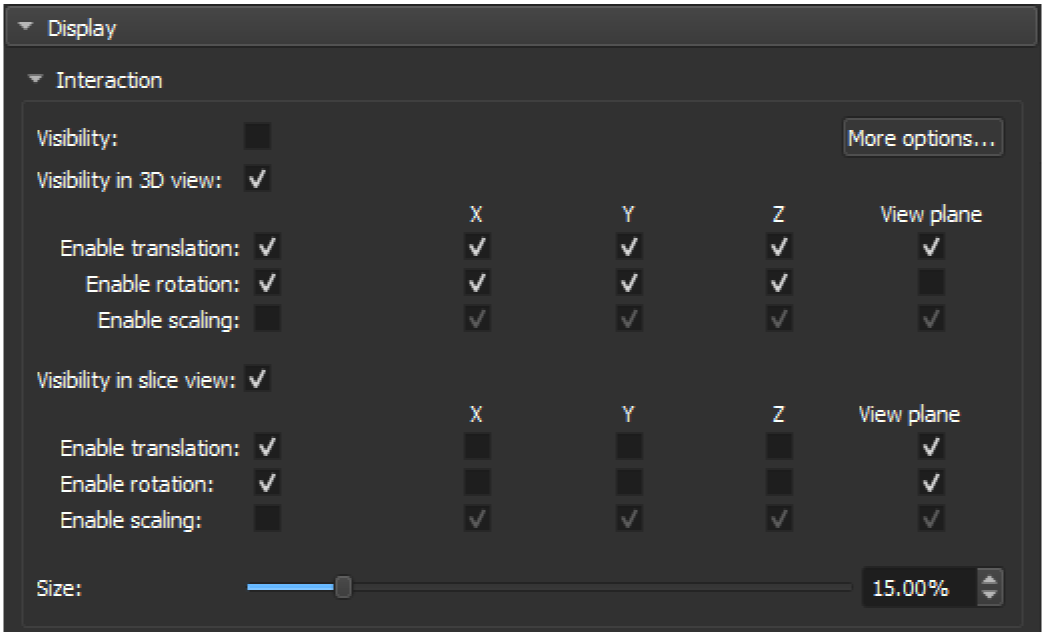




Display

This section allows visualization of how much and what direction of displacements specified by the transform.

Interaction

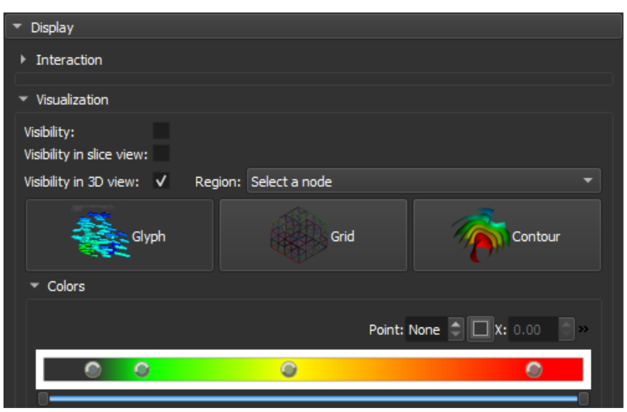


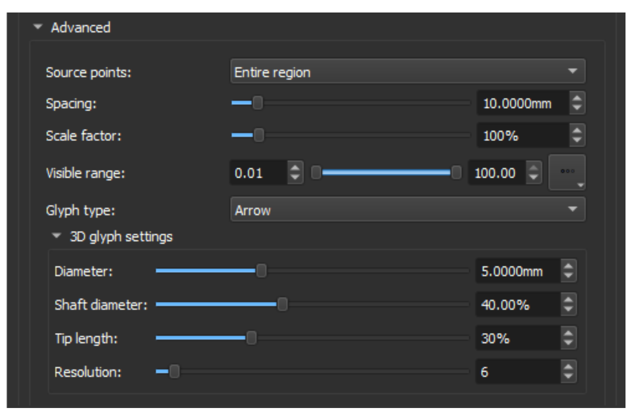
The interaction section controls the visibility of interaction handles for manual positioning of transforms. Individual axes can be enabled/disabled separately for 3D and Slice views by clicking the "More options" button.

In the slice view's "More options" panel, the "Enable view plane translation anywhere" option allows adjusting translation by left-clicking and dragging anywhere in the slice view, instead of only by grabbing the small handle at the center of the transform. While this option is enabled, that handle is drawn as a crosshair (with an empty center) instead of a filled circle. This option is disabled by default, and only takes effect if translation is enabled for the slice view. The "View plane translation sensitivity" slider (default 0.2) controls how much the transform moves relative to the mouse cursor movement while dragging anywhere; it does not affect dragging handles.

- Size: Size of the interaction handles, relative to the application window size (or as an absolute size in mm, if `absolute` is enabled).
- Opacity: Overall opacity of the interaction handles, in addition to the fade effect that is already applied when a handle is viewed edge-on.

Visualization modes





1. Glyph mode

Slice view:

- Arrow: the arrow shows the displacement vector at the arrow starting point, projected to the slice plane
- Cone: the cone shows the displacement vector at the cone centerpoint, projected to the slice plane
- Sphere: the circle diameter shows the displacement vector magnitude at the circle centerpoint

3D view:

- Arrow: the arrow shows the displacement vector at the arrow starting point
- Cone: the cone shows the displacement vector at the cone centerpoint
- Sphere: the sphere diameter shows the displacement vector magnitude at the circle centerpoint

2. Grid mode

- Slice view: Grid visualization (2D): shows a regular grid, deformed by the displacement vector projected to the slice
- 3D view: shows a regular grid, deformed by the displacement vector

3. Contour mode

- Slice view: iso-lines corresponding to selected displacement magnitude values
- 3D view: iso-surfaces corresponding to selected displacement magnitude values

Coloring

Open Transforms module / Display section / Colors section. If you click on a small circle then above the color bar you can see the small color swatch. On its left side is the points index (an integer that tells which point is being edited and that can be used to jump to the previous/next point), and on its right side is the mm value corresponding to that color.

The default colormap is:

- 1mm (or below) = gray
- 2mm = green
- 5mm = yellow
- 10mm (or above) = red

You can drag-and-drop any of the small circles or modify the mm value in the editbox. You can also add more color values by clicking on the color bar. Then, you can assign a color and/or adjust the mm value. If you click on a circle and press the DEL key then the color value is deleted.

If you need to know accurate displacement values at specific positions then switch to contour mode and in the "Levels" list enter all the mm values that you are interested in. For example, if you enter only a single value "3" in the Levels field you will see a curve going through the points where the displacement is exactly 3 mm; on one side of the curve the displacements are smaller, on the other side the displacements are larger.

You can show both contours and grid or glyph representations by loading the same transform twice and choosing a different representation for each.

Apply transform

Controls what nodes the currently selected Transform is applied to.

- Transformable: List the nodes in the scene that *do not* directly use the selected transform node.
- Transformed: List the nodes in the scene that use the selected transform node.
- Right arrow: Apply the selected transform node to the selected nodes in Transformable list.

- Left arrow: Remove the selected transform node from the selected nodes in the Transformed list.
- Harden transform: Harden selected transform on the nodes selected in the Transformed list.

Convert

This section can sample the selected transform on a grid (specified by the selected "Reference volume") and save it to a node. Depending on the type of selected "Output displacement field" node, slightly different information is exported:

- Scalar volume node : magnitude of displacement is saved as a scalar volume
- Vector volume node: displacement vectors are saved as voxel values (in RAS coordinate system). When the vector volume is written to file, the image grid is saved in LPS coordinate system, but displacement values are still kept in RAS coordinate system.
- Transform node: a grid transform is constructed. This can be used for creating an inverted displacement field that any ITK application can read. When the grid transform is written to file, both the image grid and displacement values are saved in LPS coordinate system.

9.9.4 Related modules

- *Registration modules*: transforms can be computed automatically using semi-automatic or automatic registration modules

9.9.5 Information for developers

See examples and other developer information in *Developer guide* and *Script repository*.

9.9.6 Contributors

- Alex Yarmarkovich (Isomics, SPL)
- Jean-Christophe Fillion-Robin (Kitware)
- Julien Finet (Kitware)
- Andras Lasso (PerkLab, Queen's)
- Franklin King (PerkLab, Queen's)
- Kyle Sunderland (PerkLab, Queen's)

9.9.7 Acknowledgements

This work is funded in part by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

9.10 View Controllers

9.10.1 Overview

The View Controllers module provides access to nodes controlling multiple views within a single panel. A view is a display of data packed within layout. Slice Views and 3D Views are two types of views that can appear in a layout and whose controls can also be accessed from the View Controllers module. Each type of view has a separate section of the View Controllers panel. For example, the Slice Controllers are grouped together, followed by the 3D Controllers. An extra panel allows an alternative control over a Slice View.

9.10.2 Panels and their use

- **Slice Controllers**: Slice Controllers for each of the Slice Views visible in the current layout. This is the same controller that is accessible from the bar at the top of a Slice View. It provides access to select the content (foreground, background, label) as well as control reformation, linking, visibility in the 3D view, lightbox, etc.
- **3D View Controllers**: Controllers for each 3D View. This is the same controller that is accessible from the bar at the top of a 3D View. It provides access to the view direction, zooming, spinning, rocking, etc.
- **Slice Information**: An alternative panel to control geometric parameters of a Slice View (field of view, lightbox, slice spacing).

9.10.3 Contributors

Wendy Plesniak (SPL, BWH), Jim Miller (GE), Steve Pieper (Isomics), Ron Kikinis (SPL, BWH), Jean-Christophe Fillion-Robin (Kitware)

9.10.4 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.

9.11 Volume rendering

9.11.1 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.

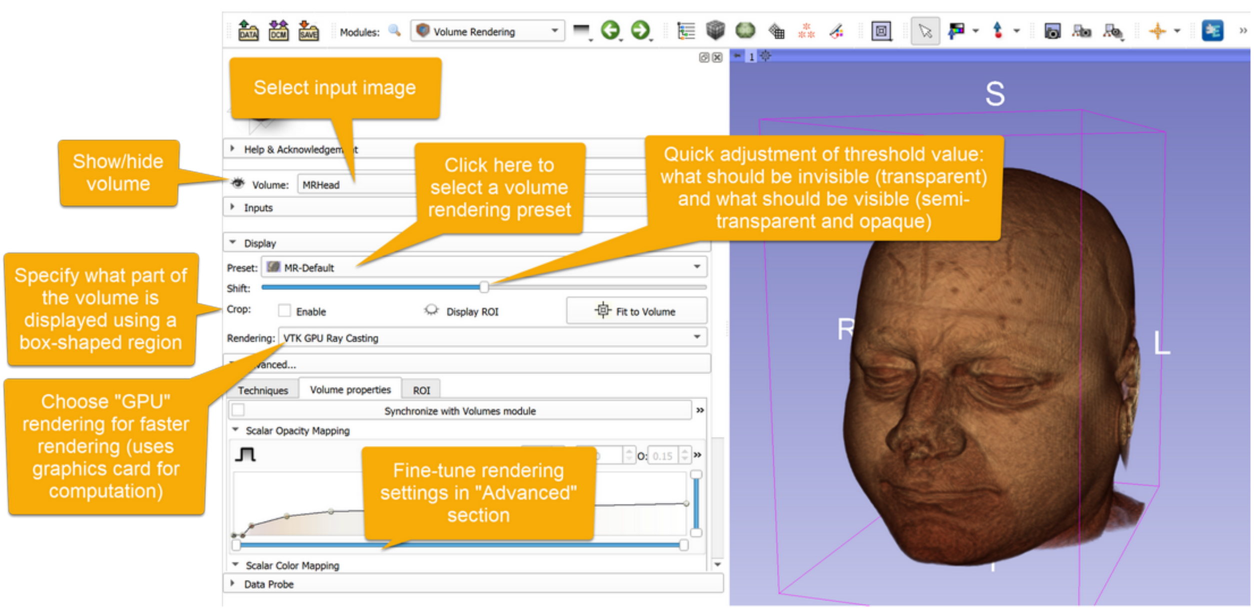
9.11.2 Use cases

Display a CT or MRI volume

- Load a data set (for example, use `Sample Data` module to load "CTChest" data set)
- Go to `Data` module
- Show volume rendering:
  - Option A: drag-and-drop the volume in the subject hierarchy tree into a 3D view
  - Option B: right-click on the eye icon, and choose "Show in 3D views as volume rendering"

To adjust volume rendering settings:

- Option A: Switch to "Adjust window/level" mouse mode using the toolbar. Click on a displayed volume and click-and-drag to adjust the rendering settings.
  - Moving the mouse vertically: shifts the opacity and color transfer functions, which changes what range of voxel values are visible in the image (similar to intensity "level" adjustment).
  - Moving the mouse horizontally: shrinks or stretches the transfer functions (similar to intensity "window" adjustment).
  - Holding down Ctrl key while starting the click-and drag and then moving the mouse vertically: adjusts the opacity of the volume.
- Option B:
  - Go to `Volume rendering` module and select the volume to be adjusted. This can be achieved by right-click on the eye icon in the `Data` module and choose "Volume rendering settings".
  - Choose a different preset in Display section.
  - Adjust "Offset" slider to change what image intensity range is visible.



Note that if the volume is under a non-linear warping transformation then internally the entire volume is resampled at its native resolution. To better control image quality and computation time, `Crop volume` module can be used to resample at a different resolution or resample only a certain part of the volume.

Render different volumes in two views

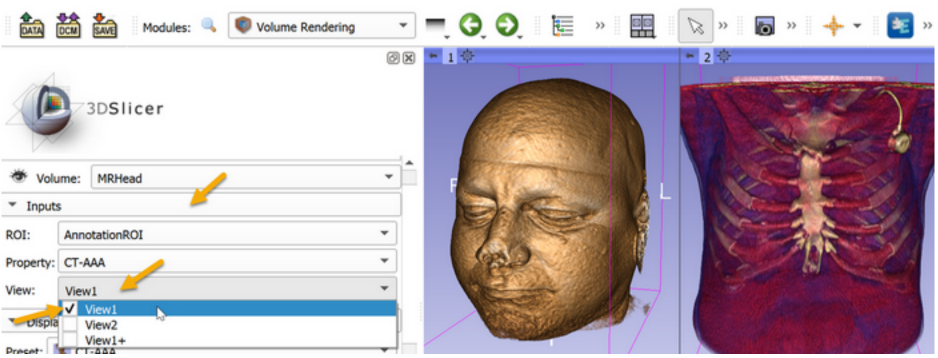
Switch to a layout with multiple 3D views (for example "Dual 3D") using the toolbar and then use one of the two options below.

Option A:

- Go to `Data` module and drag-and-drop each volume into the 3D view

Option B:

- Go to `Volume Rendering` module
- 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"



Hide certain regions of the volume

It may be necessary to hide certain regions of a volume, for example remove patient table, or remove ribs that would occlude the view of the heart or lungs. If the regions cannot be hidden by adjusting the cropping box (ROI) then an arbitrarily shaped regions can be blanked out from the volume by following these steps:

- Go to the Segment Editor module
- Use Paint or Scissors effect to specify the region that will be blanked out
- Use Mask volume effect to fill the region with "empty" values. In CT volume, intensity value is typically set to -1000 (corresponding to HU of air).
- Hide the volume rendering of the original volume and set up volume rendering for the masked volume
- If adjusting the rendered region is needed, go back to Segment Editor module, modify the segmentation using Paint, Scissors, or other effects; and update the masked volume using Mask volume effect

See video demo/tutorial of these steps for details. It is created on an older Slicer version, so some details may be different but the high-level workflow main workflow is still very similar.

9.11.3 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: 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 (default): 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 but it has several limitations (see details in *limitations* section at the bottom of this page.
- 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: quality is reduced while interacting with the view (rotating, changing volume rendering settings, etc.). This mechanism relies on measuring the current rendering time and adjusting quality (number of casted rays, sampling distances, etc.) for the next rendering request to achieve the requested frame rate. This works very well for CPUs because the computation time is very predictable, but for GPU volume rendering fixed quality ("Normal" setting) may be more suitable.
      · 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 (default): 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.

* Auto-release resources: When a volume is shown using volume rendering then graphics resources are allocated (GPU memory, precomputed gradient and space leaping volumes, etc.). This flag controls if these resources are automatically released when the volume is hidden. Releasing the resources reduces memory usage, but it increases the time required to show the volume again. Default value can be set in application settings Volume Rendering panel.
* 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, scalar mapping mode) 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. This section is not displayed for color (RGB or RGBA) volumes, as no scalar to color mapping is performed in this case (but each voxel's color is used directly).
* 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.

9.11.4 Limitations

- To render multiple overlapping volumes, select "VTK Multi-Volume" rendering in "Display" section. This renderer is still experimental and has limitations such as:
    - Cropping is not supported (cropping ROI is ignored)
    - RGB volume rendering is not supported (volume does not appear)
    - Only "Composite with shading" rendering technique is supported (volume does not appear if "Maximum Intensity Projection" or "Minimum Intensity Projection" technique is selected)
- To reduce staircase artifacts during rendering, choose enable "Surface smoothing" in Advanced/Techniques/Advanced rendering properties section, or choose "Normal" or "Maximum" as quality.
- If the application crashes when rotating or zooming a volume: This indicates that you get a TDR error, i.e., the operating system shuts down applications that keep the graphics card busy for too long. This happens because the size of the volume is too large for your GPU to comfortably handle. There are several ways to work around this:
    - Option A: Run the code snippet in the Python console (Ctrl-3) to split the volume to smaller chunks (that way you have a better chance that the graphics card will not be unresponsive for too long).

```
slicer.vtkMRMLVolumeRenderingDisplayableManager.SetMaximum3DTextureSize(400)
for vrDisplayNode in getNodesByClass('vtkMRMLVolumeRenderingDisplayNode'):
    slicer.util.arrayFromVolumeModified(vrDisplayNode.GetVolumeNode())
```

    - Option B: Crop and downsample your volume using Crop volume and volume render this smaller volume.
    - Option C: Increase TDR delay value in registry (see details here)
    - Option D: Use CPU volume rendering.
    - Option E: Upgrade your computer with a stronger graphics card.

9.11.5 Information for developers

See examples and other developer information in *Developer guide* and *Script repository*.

9.11.6 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)

9.11.7 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)

9.12 Volumes

9.12.1 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.

9.12.2 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 simpler options available:

- A. Choose `Segmentation` in `Description` column in `Add data...` window. This only works for nrrd and nifti images.
- B. Sve the volume in nrrd file format and rename it to have `.seg.nrrd` extension, for example: `something.seg.nrrd`. This makes the file loaded as `Segmentation` by default.

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

Consider the Image Stacks provided by the SlicerMorph extension to work with large sets of high resolution images. It allows you to automatically convert RGB to grayscale scalar volume, specify image spacing, and downsample large images.

Alternatively you can load and manipulate the data directly using functionality in the core Slicer modules:

- 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.

9.12.3 Panels and their use

- 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.
  - Presets: Predefined volume display presets that set window/level and the lookup table for common visualization requirements.
  - 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
  - Scalar Mapping Mode: Determines how scalar values within the window/level range are distributed across the lookup table. Useful for reducing saturation of the displayed image and emphasizing lower or higher intensity regions. In the formulas below, value is the voxel value being mapped, and colorIndex is the resulting normalized position (0-1 range) used to look up the corresponding color in the colormap.
    * Linear (default): maps values proportionally. `colorIndex = (value - level + window/2) / window`
    * Logarithmic: emphasizes the low end of the intensity range. `colorIndex = log10(1 + 9 * (value - level + window/2) / window)`
    * Inverse logarithmic: emphasizes the high end of the intensity range. `colorIndex = 1 - log10(1 + 9 * (level + window/2 - value) / window)`
  - 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. An advanced options button can be clicked to display controls 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.

9.12.4 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
  - Image Stacks provided by the SlicerMorph extension.

9.12.5 Information for developers

See examples and other developer information in *Developer guide* and *Script repository*.

9.12.6 Contributors

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

9.12.7 Acknowledgements

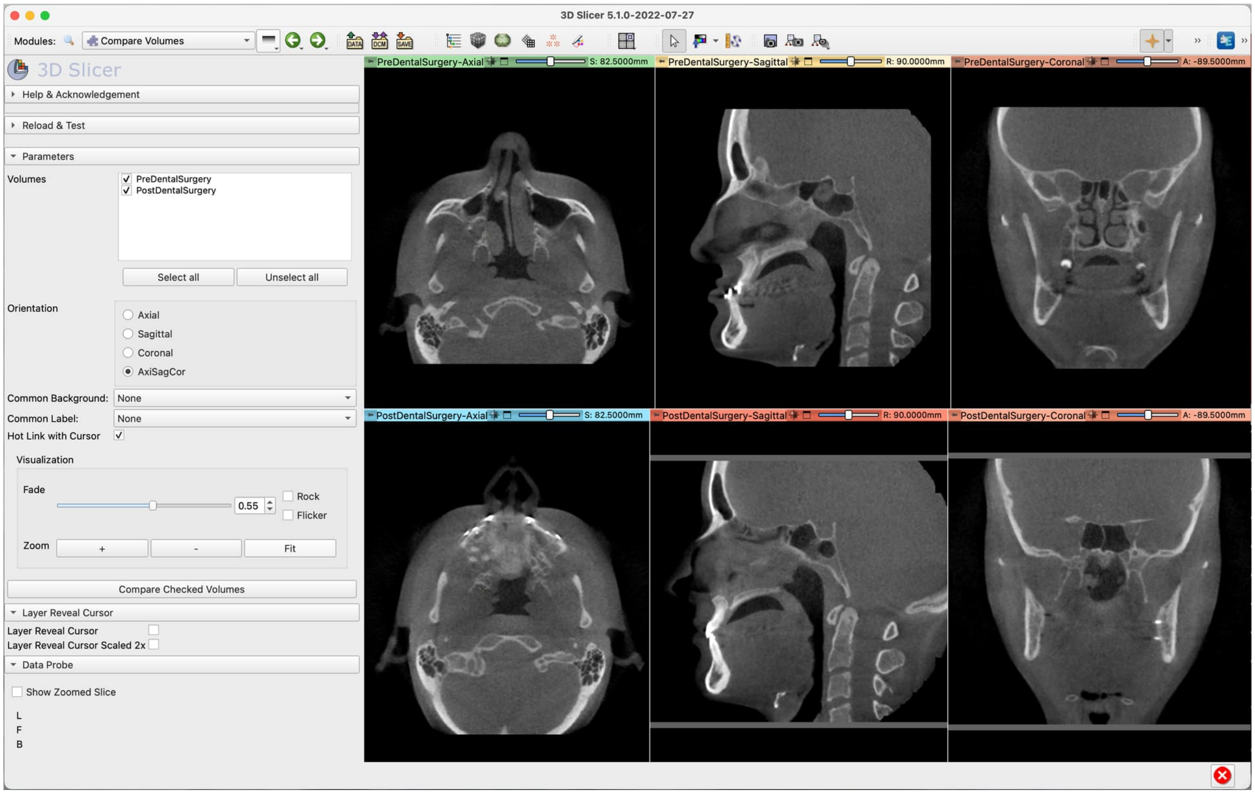
This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

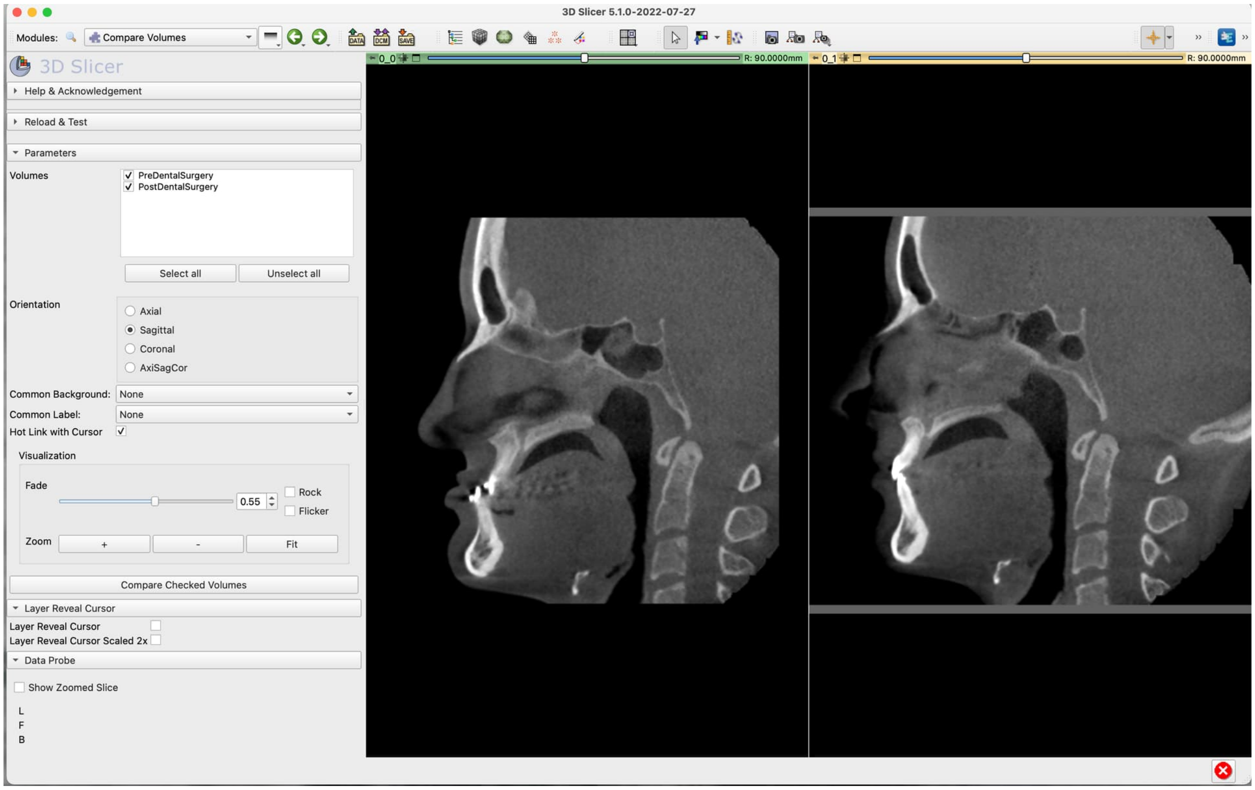
9.13 Wizards

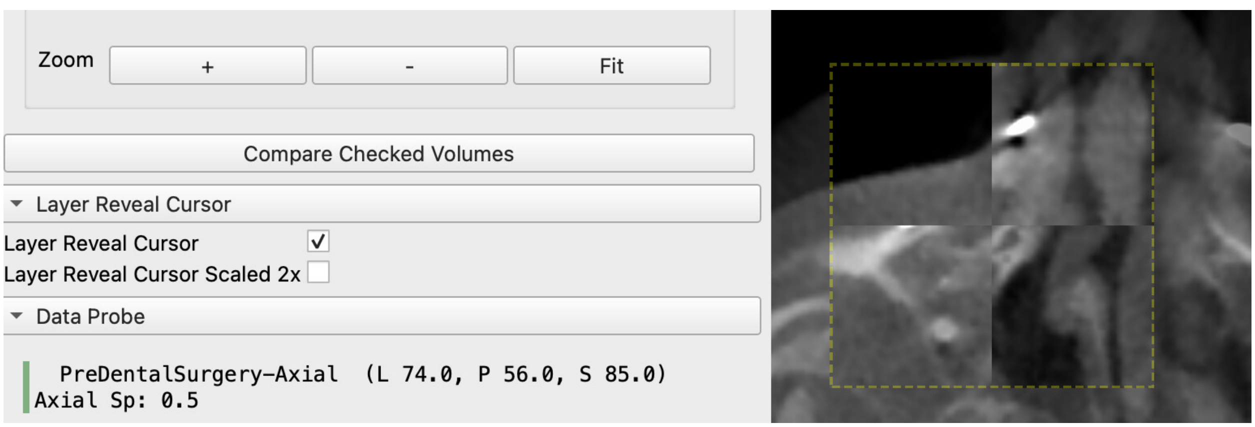
9.13.1 Compare Volumes

Overview

The Compare Volumes module manages the layout and linking of multiple volumes for you and gives other options. It is meant for comparing registration results but is also good for looking at multiple MR contrasts.







Use Cases

Most frequently used for these scenarios:

- **Compare the results of image registration**.
- **Look at all series in a DICOM study**
- **Compare timepoints**
- **View the same segmentation over multiple volumes**

Panels and their use

The **Volumes** section lists all the volumes in the scene. You can select the ones you want to see and can drag them to set the layout order.

The **Orientation** radio buttons lets you pick between one view or a row of three per volume.

The **Common Background** option lets you overlay all selected volumes in the foreground compared to a reference volume in the background (used with the Visualization Fade/Rock/Flicker modes). If you have just two volumes you can select one here and uncheck it in the Volumes section and you will get a layout with one set of views with the volumes in foreground and background.

Common Label isn't useful as much now because Segmentations are shown over all views, but if you have a labelmap you can choose to display it.

Hot Link sets up linking mode so that pan/zoom/scroll happens as you drag vs on mouse release.

Visualization mode allows you to crossfade between foreground and background when you have common background enabled. The Rock and Flicker modes animate this action so you can look for subtle changes without needing to use the mouse. You can also pan/zoom/scroll while the animation modes are active to explore the volumes.

The **+ - Fit** buttons control the field of view of the slice views.

Compare Checked Volumes is the button to apply the Volumes, Orientation, and Common Background/Label options in a custom layout. Note that in current Slicer it seems you sometimes need to click the Fit button after applying a new layout.

The **Layer Reveal Cursor** show a real-time checkerboard of the foreground and background as you move the mouse over the slice views.

Contributors

- Steve Pieper (Isomics, Inc.)

Acknowledgments

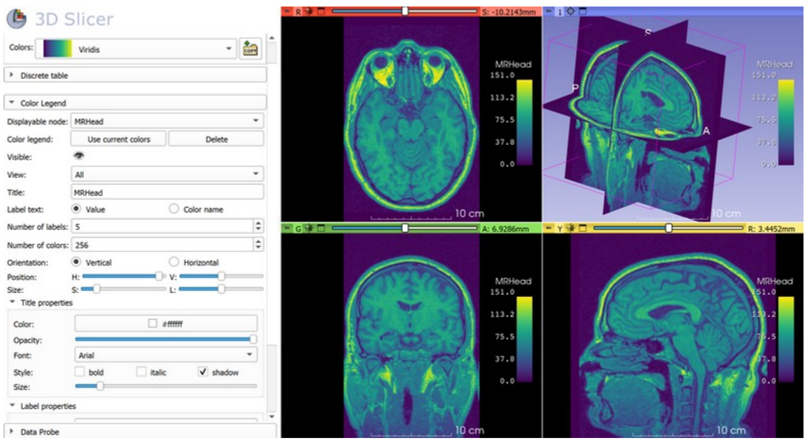
This work was supported by NIH grants R01CA111288 and U01CA151261, NA-MIC, NAC and the Slicer Community.

9.14 Informatics

9.14.1 Colors

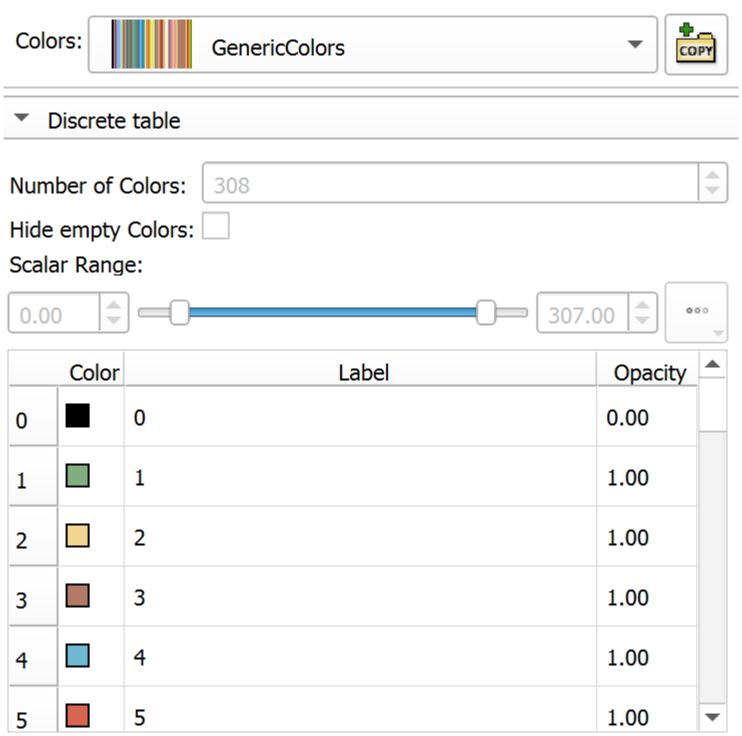
Overview

The Colors module manages color look up tables, stored in Color nodes. These tables translate between a numeric value and a color for displaying of various data types, such as volumes (to specify coloring based on voxel values), models (for displaying scalars), and markups (for displaying curve curvatures and other measurements in 3D views). The module also used for assigning colors to various displayable nodes and show color legend in viewers.

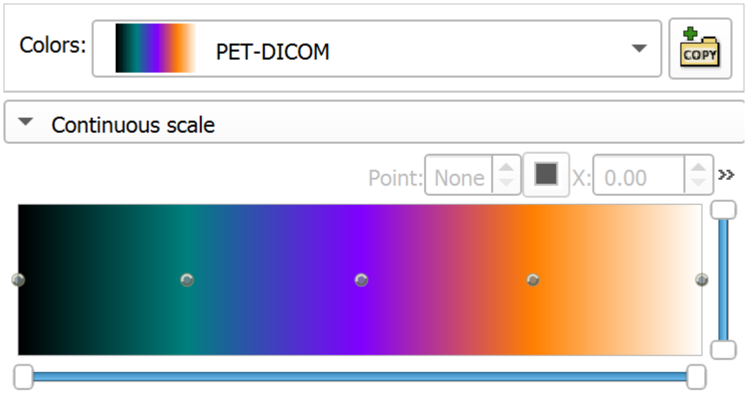


Two lookup table types are available:

- Discrete table: List of named colors are specified (example: GenericAnatomyColors). Discrete tables can be used for continuous mapping as well, in this case the colors are used as samples at equal distance within the specified range, and smoothly interpolating between them (example: Grey).



- Continuous scale: Color is specified for arbitrarily chosen numerical values and color value can be computed by smoothly interpolating between these values (example: PET-DICOM). No names are specified for colors.



How to

Edit a color lookup table

All built-in lookup tables are read-only to ensure consistency when using these colors. To edit a lookup table:

- Go to Colors module
- Choose a built-in lookup table that is similar to the desired color mapping. For example, `Labels` can be used as a starting points for a discrete table, `PET-DICOM` can be used for creating an editable continuous scale.
- Click the yellow "copy" folder + icon next to the colors selector to create a copy.
- Specify a name for the new color lookup table and click OK.

The loaded color table will appear in the "User generated" category at the top. Click on the arrow on the category name (or click to select and hit the right-arrow key) to open it and see the color tables in that category.

Save a color lookup table

To save a modified color table to file, use the application menu: File / Save.

File format is described in the *Developer guide*.

Load a color lookup table

To save a modified color table to file, drag-and-drop the file to the application window (or use the application menu: File / Add Data) and click OK.

The loaded color table will appear in the "File" category at the top. Click on the arrow on the category name (or click to select and hit the right-arrow key) to open it and see the color tables in that category.

Show color legend

There are several ways of showing/hiding color legend:

- In Data module: Right-click on an eye icon in the data tree to open the visibility menu. In that menu, "Show color legend" option can be used for showing/hiding the color legend in all views. The option only appears if a color lookup table is already associated with the selected item.
- In Colors module: Color legend can be added/hidden for any displayable node at one place. Open the Color Legend section, select a displayable node (volume, model, markup). Click `Create` to create a color legend, and adjust visualization options.
- In Volumes, Models, Markups modules: Color Legend section can be used to show legend for the selected node.

- In slice views: Right-click on a volume to open the view context menu. In that menu, "Show color legend" option can be used to show/hide color legend for that volume in slice views.

For displaying mapping from numeric values to colors (for example, for displaying a parametric image), choose `Label text -> Value`. For displaying color names (for example, names of anatomical structures for a labelmap volume), choose `Label text -> Color name`.

> **Tip**
>
> For displaying color legend for segment names: *Export the segments to a labelmap volume*. This creates a color table, which can be displayed as a color legend.

Panels and their use

- **Colors:** a drop down menu from which to select from the list of loaded color nodes
  - Copy Color Node button: Duplicate the current color node to edit the color entries. Necessary, because built-in color nodes are note editable.
- **Discrete table**
  - **Number of Colors:** the number of colors in the currently selected table.
  - **Hide empty Colors:** When checked, hide the unnamed color entries in the list below.
  - **Scalar Range:** The range of scalar values that are mapped to to the full range of colors. This only changes the display range, not the values in the table. It is used if the scalar range in a model or markups node is chosen as "Color table (LUT)"
  - **Table of currently selected colors**
    - **Index:** the integer value giving the index of this color in the look up table, used to match it up with a scalar value in a volume voxel.
    - **Color:** a box showing the current color. When viewing an editable table, double click on it to bring up a color picker.
    - **Label:** the text description of the color, often an anatomical label. For some tables, the name is automatically generated from the RGBA values.
    - **Opacity:** a value between 0 and 1 describing how opaque this color is. The background color at index 0 is usually set to 0 and other colors to 1.
- **Continuous Display Panel**: Inspect the color values by clicking on the circles that determine the points of the continuous function. Click elsewhere in the color display to add new points.
  - **Point:** the index of this point.
  - **Color box:** click on this to edit the color for this point.
  - **X:** the numeric value that maps to this color.
- **Color legend**
  - **Displayable node:** node that the color legened will ve displayed for.
  - **Color legend**
    - **Create**: create color legend for the selected Displayable node, using the Colors node chosen at the top.
    - **Use current colors**: set the Colors node chosen at the top for showing the Displayable node.
    - **Delete**: remove the Displayable node's color legend.

  - **Visible:** show/hide the color legend.
  - **View:** Choose which views the color legend is allowed to be appear in. Color legend for models and markups is only displayable in the views where these nodes appear.
  - **Title:** Title displayed at the top of the color legend. Set to the Displayable node's name by default.
  - **Label text**
    * **Value:** Numeric value is displayed in labels next to the color bar. Intended for displaying continuous numeric scale.
    * **Color name:** The color name is displayed in labels next to the color bar. Intended for displaying names of labels (segments).
  - **Number of labels:** Number of labels to display. Only applicable if `Value` is used as label text.
  - **Number of colors:** Maximum number of colors displayed. Reduce the number to see discrete colors instead of a continuous color gradient. Only applicable if `Value` is used as label text.
  - **Orientation**: choose between horizontal or vertical color bar orientation.
  - **Position**: horizontal and vertical position in the view
  - **Size**: length of the long and short side of the legend box.
  - **Title properties** and **Label properties**: controls the format of the title and label text.
  - **Format:** label format (only for label properties). Defined using printf specifiers. Examples:
    * display with 1 fractional digit: `%.1f`
    * display integer values: `%.0f`
    * display with 4 significant digits: `%.4g`
    * display string label annotation: `%s`
  - **Color:** text color.
  - **Opacity:** allows displaying the text semi-transparently.
  - **Font:** the font used to display the title text. Arial: sans-serif font. Courier: fixed-width font. Times: serif font.
  - **Style:** check these boxes if you wish to adjust the style of the title text font. **Shadow** makes the text visible over both dark and bright background.
  - **Size:** font size of the title. Label size is auto-scaled to fit the specified size of the color legend.

Built-in color lookup tables

- Discrete
  - Labels: A legacy color table that contains some anatomical mapping
  - DiscreteFullRainbow.png]] FullRainbow: A full rainbow of 256 colors, goes from red to red with all rainbow colors in between. Useful for colorful display of a label map.
  - Grey: A grey scale ranging from black at 0 to white at 255. Useful for displaying MRI volumes.
  - Iron: A scale from red to yellow, 157 colors.
  - Rainbow: Goes from red to purple, passing through the colors of the rainbow in between. Useful for a colorful display of a label map.
  - Ocean: A lighter blue scale of 256 values, useful for showing registration results.

  - Desert: Red to yellow/orange scale, 256 colous.
  - InvertedGrey: A white to black scale, 256 colors, useful to highlight negative versions, or to flip intensities of signal values.
  - ReverseRainbow: A colorful display option, 256 colors going from purple to red.
  - fMRI: A combination of Ocean (0-22) and Desert (23-42), useful for displaying functional MRI volumes (highlights activation).
  - fMRIPA: A small fMRI positive activation scale going from red to yellow from 0-19, useful for displaying functional MRI volumes when don't need the blue of the fMRI scale.
  - Random: A random selection of 256 rgb colors, useful to distinguish between a small number of labeled regions (especially outside of the brain).
  - Red: A red scale of 256 values. Useful for layering with Cyan.
  - Green: A green scale of 256 values, useful for layering with Magenta.
  - Blue: A blue scale of 256 values from black to pure blue, useful for layering with Yellow.
  - Yellow: A yellow scale of 256 values, from black to pure yellow, useful for layering with blue (it's complementary, layering yields gray).
  - Cyan: A cyan ramp of 256 values, from black to cyan, complementary ramp to red, layering yields gray.
  - Magenta: A magenta scale of 256 colors from black to magenta, complementary ramp to green, layering yields gray.
  - Warm1: A scale from yellow to red, of 256 colors, ramp of warm colors that's complementary to Cool1.
  - Warm2: A scale from green to yellow, 256 colors, ramp of warm colors that's complementary to Cool2.
  - Warm3: A scale from cyan to green, 256 colors, ramp of warm colors that's complementary to Cool3.
  - Cool1: A scale from blue to cyan, 256 colors, ramp of cool colors that's complementary to Warm1.
  - Cool2: A scale from magenta to blue, 256 colous, ramp of cool colors that's complementary to Warm2.
  - Cool3: A scale from red to magenta, ramp of cool colors that's complementary to Warm3.
  - RandomIntegers: A random scale with 1000 entries.
  - GenericAnatomyColors: a list of whole body anatomy labels and useful colors for them, the default for the Editor module creating new label map volumes
  - Generic Colors: a list of colors with the names being the same as the integer value for each entry

- Shade:
  - WarmShade1: A scale from black to red, of 256 colors, ramp of warm colors with variation in value that's complementary to CoolShade1.
  - WarmShade2: A scale from black to yellow, through green, of 256 colors, ramp of warm colors with variation in value that's complementary to CoolShade2.
  - WarmShade3: A scale from black to green, of 256 colors, ramp of warm colors with variation in value that's complementary to CoolShade3.
  - CoolShade1: A scale from black to cyan, 256 colors, ramp of cool colors with variation in value that is complementary to WarmShade1.
  - CoolShade2: A scale from black to blue through purple, 256 colors, ramp of cool colors with variation in value that is complementary to WarmShade2.

  - CoolShade3: A scale from black to magenta, 256 colors, ramp of cool colors with variation in value that is complementary to WarmShade3.
- Tint:
  - WarmTint1: A scale from white to red, 256 colors, ramp of warm colors with variation in saturation that's complementary to CoolTint1.
  - WarmTint2: A scale from white to yellow, 256 colors, ramp of warm colors with variation in saturation that's complementary to CoolTint2.
  - WarmTint3: A scale from white to green, 256 colors, ramp of warm colors with variation in saturation that's complementary to CoolTint3.
  - CoolTint1: A scale from white to cyan, 256 colors, ramp of cool colors with variations in saturation that's complementary to WarmTint1.
  - CoolTint2: A scale from white to blue, 256 colors, ramp of cool colors with variations in saturation that's complementary to WarmTint2.
  - CoolTint3: A scale from white to magenta, 256 colors, ramp of cool colors with variations in saturation that's complementary to WarmTint3.
- Continuous
  - RedGreenBlue: A scale defined from -6.0 to 6.0 that maps to a rainbow from red to blue through green.
- FreeSurfer
  - Heat: The Heat FreeSurfer color table, shows hot spots with high activation
  - BlueRed: A FreeSurfer color scale, 256 colors, from blue to red
  - RedBlue: A FreeSurfer color scale, 256 colors, from red to blue
  - RedGreen: A FreeSurfer color scale, 256 colors, from red to green, used to highlight sulcal curvature
  - GreenRed: A FreeSurfer color scale, 256 colors, from green to red, used to highlight sulcal curvature
  - FreeSurferLabels: A color table read in from a text file, each line of the format: IntegerLabel Name R G B Alpha
- PET
  - PET-Heat: Useful for displaying colorized PET data.
  - PET-Rainbow: Useful for displaying colorized PET data.
  - PET-MaximumIntensityProjection: Useful for displaying inverse grey PET data.
  - PET-Rainbow2: `rainbow` color table from the FIJI PET-CT plugin.
  - PET-DICOM: DICOM standard color lookup table `PET`.
  - PET-HotMetalBlue: DICOM standard color lookup table `Hot Metal Blue`.
- Cartilage MRI
  - dGEMRIC-1.5T: Useful for displaying 1.5 tesla delayed gadolinium-enhanced MRI of cartilage
  - dGEMRIC-3T: Useful for displaying 3 Tesla delayed gadolinium-enhanced MRI of cartilage
- Default Labels from File: a list of color nodes loaded from files, from the default Slicer color directory.
  - ColdToHotRainbow: a shifted rainbow that runs from blue to red, useful when needing to display a volume for which larger values are hotter.

- HotToColdRainbow: a shifted rainbow that runs from red to blue, useful when needing to display a volume for which larger values are colder.
- PelvisColor: useful for displaying segmented pelvic MRI volumes.
- Slicer3_2010_Brain_Labels: a brain segmentation table with 16 labels defined.
- 64Color-Nonsemantic: A color table with no semantic labels, pure color information.
- MediumChartColors: A Stephen Few palette. Similar to the Dark Bright color but in the medium range of intensity. May be a better choice for bar charts.
- DarkBrightChartColors: Palette designed by Stephen Few in "Practical Rules for Using Colors in Charts". Similar to the Maureen Stone palette. Stephen Few recommends this palette for highlighting data. This palette is useful when display small points or thin lines. Again, this palette is good for categorical data.
- Slicer3_2010_LabelColors: a table with 16 labels defined.
- AbdomenColors: useful for displaying segmented abdominal MRI volumes
- SPL-BrainAtlas-ColorFile: useful for displaying segmented brain MRI volumes
- SPL-BrainAtlas-2012-ColorFile: an updated brain segmentation color node
- LightPaleChartColors: A light pale palette from Stephen Few, useful for charts.
- SPL-BrainAtlas-2009-ColorFile: an updated brain segmentation color node

- File: If you load a color file from File -> Add Data, it will appear here.
- User Generated: A user defined color table, use the editor to specify it. Copies of other color tables get displayed in this category. If you create a new color node, it will appear here.

Information for developers

See examples and other developer information in *Developer guide*.

Contributors

- Nicole Aucoin (SPL, BWH)
- Jim Miller (GE)
- Julien Finet (Kitware Inc.)
- Andras Lasso (PerkLab, Queen's)
- Mikhail Polkovnikov (IHEP)

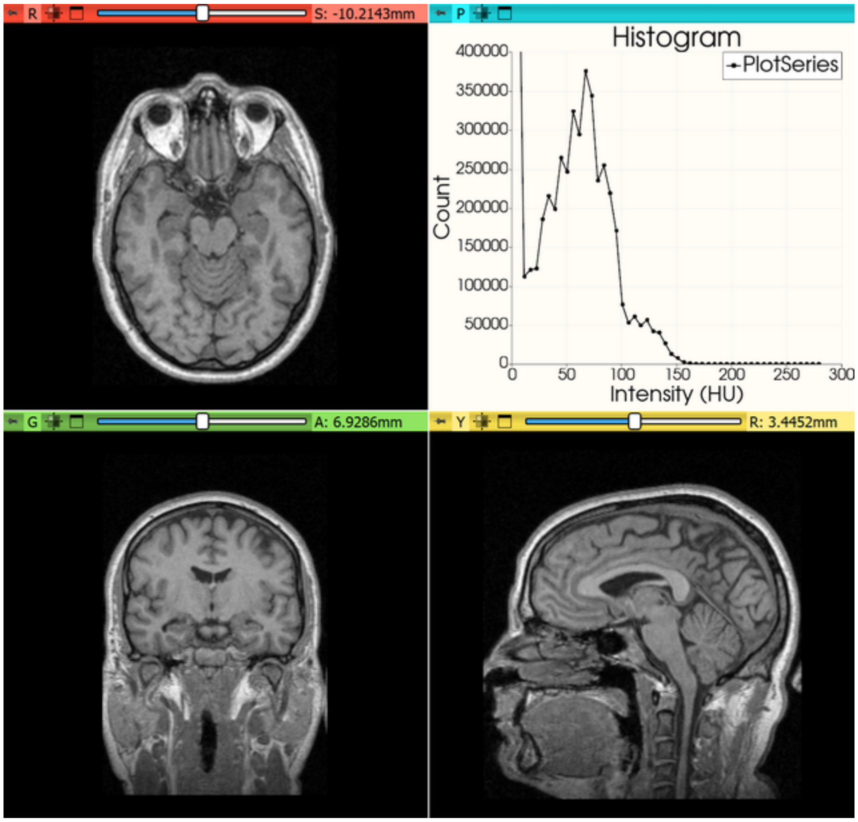
Acknowledgements

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.

9.14.2 Plots

Overview

This module can display interactive line and bar plots in the view layout. A `plot chart` can display one or more `plot series`. The plot series specifies the plot type (line, bar, scatter, scatter bar), data source (table and column for x and y data), and appearance (lines style, marker style, etc.).



Features:

- Multiple plots types (Line, Bar, Scatter, Scatter bar), shown in standard view layouts.
- Configurable chart and axis title, legend, grid lines.
- Use indexes (0, 1, 2, . . .) or a table column for X axis values.
- Multiple plot series can be added to a single plot chart (even showing data from different tables, with different types - plot, bar), and the same series can be added to multiple charts. The same plot chart can be shown in any plot views (even simultaneously).
- Label can be assigned to each data point (shown in tooltip when hovering over a data point).
- Interactive editing of data points by drag-and-drop (data in referenced table nodes is updated).
- Interactive selection of data points (rectangular or free-hand selection).
- Plot data and display properties are stored in the MRML scene.
- Plot view can be embedded into module user interfaces. Plot views can emit signals back to the application as the user interacts with a plot, which allows modules to perform additional processing on the selected data points.

How to

Plot values in a table

- Go to Tables module
- Select or create a new table
- Select at least one column with a numeric `Data type`.
  - Open `Column properties` section below the table to see data type of the selected column.
  - If data type is not numeric (double, float, int, ...) already then choose `double` for floating-point values (or `int` for integer values), and then click `Convert`.
- Click the `Generate an interactive plot...` button above the table to create and display a plot from the selected column(s).

If multiple columns are selected then the first numeric column will be used as x axis and others will be used as y axes (each pair will create a new series).

Multiple columns can be selected by holding down Shift or Ctrl key while clicking in the table. if at least one cell is selected in a column then the column is considered as selected.

Keyboard shortcuts and mouse gestures

The following keyboard shortcuts are active when a plot view has the focus (after clicked in a plot view).

Mouse gestures:

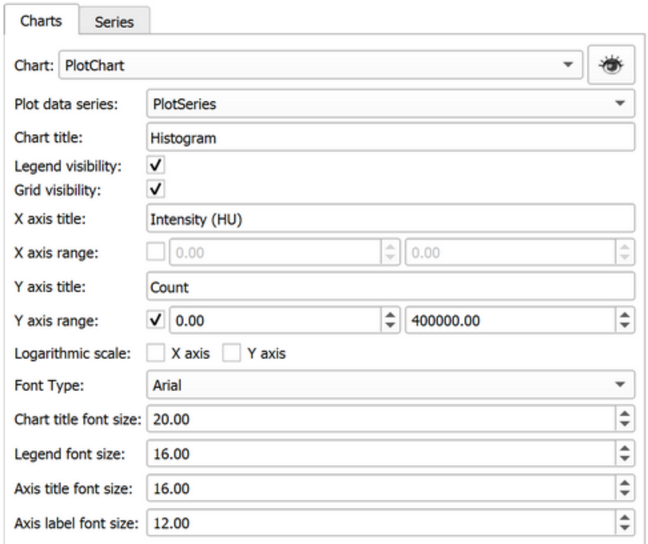
- Left mouse button: depends on interaction mode of the view
  - Pan view: pan view
  - Select points: select points using rectangular selection
  - Freehand select points: select points using free-form selection
  - Move points: move data points by drag-and-drop; this changes values in the referenced table node
- Middle mouse button: pan view (except in pan view mode; in that case it zooms to selected rectangular region)
- Right mouse button: zoom in/out along X and Y axes

Panels and their use

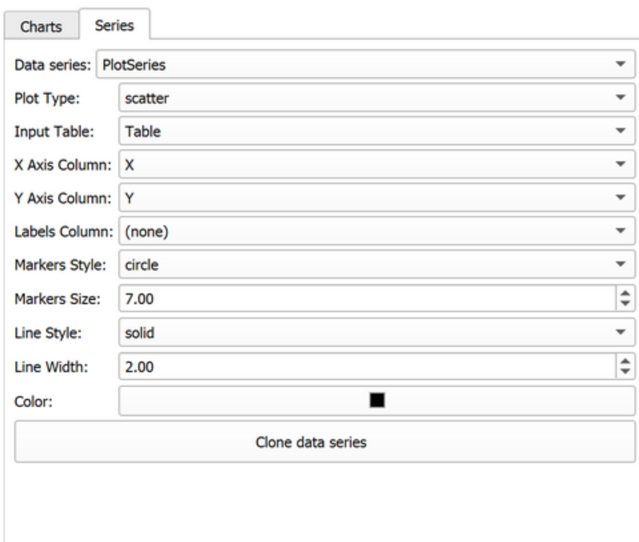
Plots module user interface has two sections, one for editing properties of charts and another for editing properties of plots.

Chart section

Choose one or more plot series in `Plot data series` and then adjust display settings.



Series section



- Plot type:
  - `line` or `bar`: displays a single numeric table column as Y values.
  - `scatter` and `scatter bar`: displays two numeric table columns, one is used as X values, the other as Y values.
- Labels column: a string column can be chosen to be used as labels on values, which are displayed when the mouse hovers over point in the plot.
- Logarithmic scale: this feature is not tested thoroughly, need to be used with caution.

Information for developers

See examples and other developer information in *Developer guide* and *Script repository*.

Related modules

This module replaced the Charts module, which was a very limited module for plotting `DoubleArray` nodes.

Contributors

Authors:

- Davide Punzo (Kapteyn Astronomical Institute, University of Groningen)
- Andras Lasso (PerkLab, Queen's University)

Acknowledgements

This work was supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement nr. 291-531.

9.14.3 Sample Data

Overview

This module provides data sets that can be used for testing 3D Slicer. Data sets are downloaded via network connection and recently used data sets are cached for faster access.

Panels and their use

`BuiltIn` category contains typical clinical images. `Development` category contains special data sets that developers can use for testing.

Extensions can add more data sets in additional custom categories.

Contributors

- Steve Pieper (Isomics)
- Benjamin Long (Kitware)
- Jean-Christophe Fillion-Robin (Kitware)
- Andras Lasso (PerkLab)

Acknowledgements

This work was was funded in part by Cancer Care Ontario and the Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO)

9.14.4 Tables

Overview

The Tables module allows displaying and editing of spreadsheets.

Panels and their use

- **Input**:
  - **Active table**: Select the table node to edit/view.
  - **Lock button**: Allows locking the table node to read-only. Only applies to the user interface. The node can eb still modified programmatically.
- **Edit**:
  - **Copy button**: Copy contents of selected cells to clipboard.
  - **Paste button**: Paste contents of clipboard at the current position. Data that does not fit into the table is ignored.
  - **Add column button**: Add an empty column at the end of the table.
  - **Delete column button**: Delete all selected columns. If any cell is selected in a column the entire column will be removed.
  - **Lock first column button**: Use the first column as row header. Header cannot be edited and is not copied to clipboard but still saved to file.
  - **Add row button**: Add an empty row at the end of the table.
  - **Delete row button**: Delete all selected rows. If any cell is selected in a row the entire row will be removed.
  - **Lock first row button**: Use the first row as column header. Header cannot be edited and is not copied to clipboard but still saved to file.

Contributors

Andras Lasso (PerkLab), Kevin Wang (PMH)

Acknowledgements

This work was was partially funded by OCAIRO, the Applied Cancer Research Unit program of Cancer Care Ontario, and Department of Anesthesia and Critical Care Medicine, Children's Hospital of Philadelphia.

9.14.5 Terminologies

Overview

The Terminologies module provides a coding system for specifying anatomy, clinical findings, or other clinical terms. For example, the coding system can be used to specify what anatomical parts are stored in a segmentation node. Using standard codes reduces the chance of data-entry errors and improves interoperability. The coding system in Slicer is compatible with coding used in DICOM information objects and allows storing SNOMED-CT (Systematized Nomenclature of Human and Veterinary Medicine - Clinical Terms) or any other terminologies.

The module stores multiple lists of terminology items, such a list is called a **context**. Terminologies module comes with two contexts for segmentations (*Segmentation category and type - DICOM master list* and *Segmentation category and type - 3D Slicer General Anatomy list*) and one for specifying region (*Anatomic codes - DICOM master list*). The *3D Slicer General Anatomy list* is a subset of the DICOM master list, containing items corresponding to labels in Slicer's *GenericAnatomyColors* color table. Custom contexts can also be defined. These contexts were created as part of the QIICR project, in the dcmqi toolkit.

Each item in the segmentation context specifies **category** (such as "Tissue", "Physical Object", "Body Substance", . . .), **type** (such as "Artery", "Bone", "Amygdala", . . .), an optional **type modifier** (such as "left" and "right"), recommended **display color**. For items that are not anatomic regions themselves, **region** can be specified as well, using an item from a region context.

Each item in the region context specifies the **region** (such as "Anterior Tibial Artery", "Bladder", . . .) and an optional **region modifier** (such as "left" and "right"). Region most commonly specifies an anatomic region, but in non-clinical applications the location may be defined at a much smaller or larger scale, not necessarily at anatomy level.

The terminology module can display user interface (called terminology navigator) for choosing a segmentation terminology item. This navigator appears when selecting the "color" of certain types of data nodes (such as models and markups) in the *Data* module, or when selecting the "color" for a segment in *Segment Editor*. The terminology navigator can also be used to select terminology information from color tables.

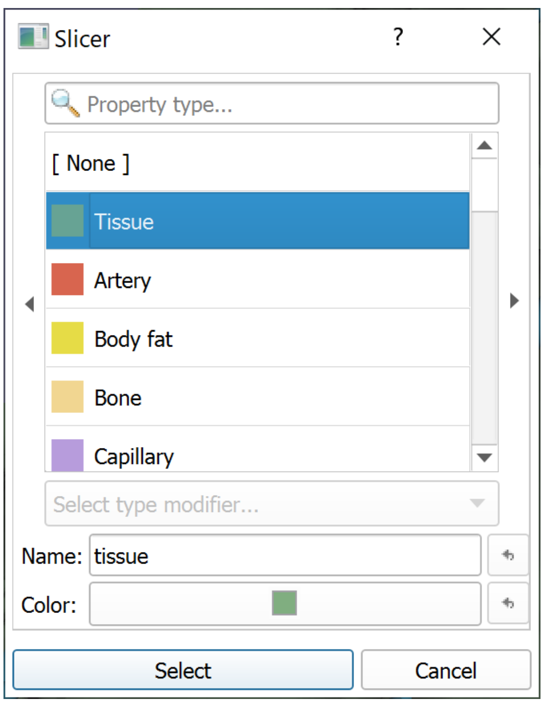
Use cases

Define category, type, type modifier and region of a data object, such as model node or markup node, or a segment in a segmentation node, so that it can be identified unambiguously. The associated standardized codes are also saved into DICOM files when the segmentation is saved as a DICOM Segmentation information object.

Panels and their use

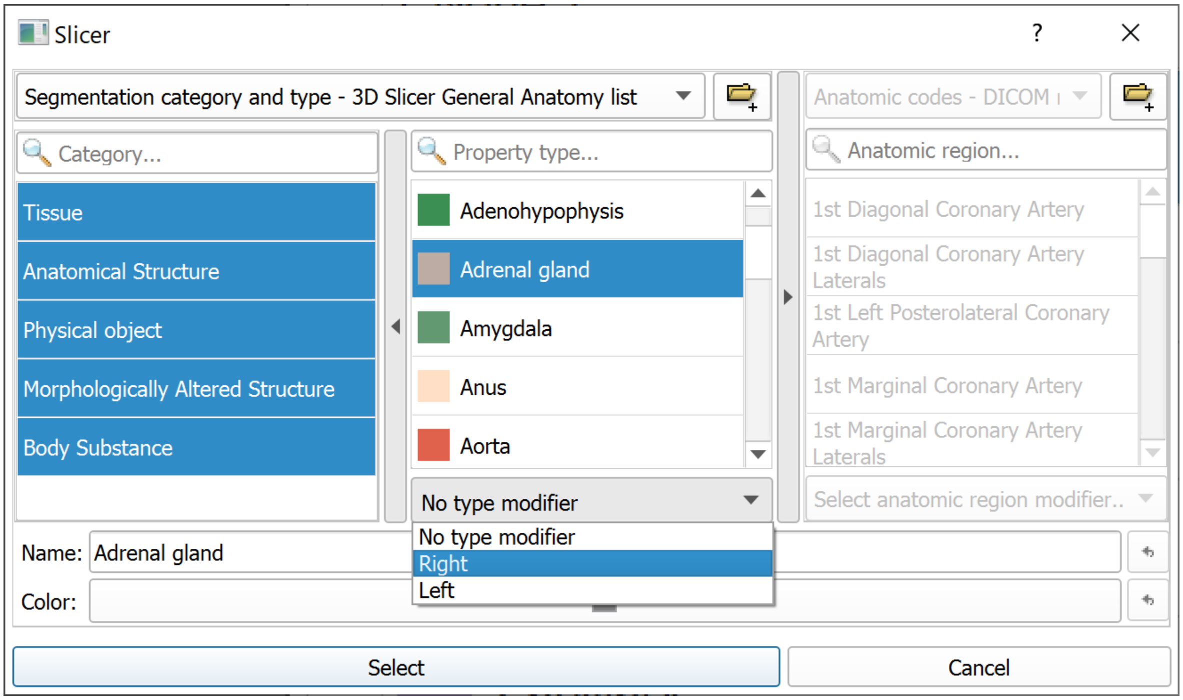
The terminology navigator dialog is displayed when clicking the color selector box in data trees and lists where the color of the objects is shown in a column as a color swatch. Such selectors are there for data nodes in the *Data*, *Models*, *Markups* modules, and for the segments within segmentation nodes in the *Segmentations* and *Segment Editor* modules.

After clicking, the basic selector window opens:



It shows the item types in all categories in a searchable list. Custom name and color can be assigned while keeping the selected terminology entry.

Opening up the panes on the left and right, additional options become visible:



- Left pane
  - Allows selection of a subset of categories (all categories selected by default).
  - On the top, the terminology context can be changed, as well as new ones loaded from `.json` files.
- Right pane
  - Region can be selected for items that can be located in multiple parts of the body, such as "Blood clot", "Mass", or "Cyst".

How to

- Find items: Start typing the name of the category/type/region in the search box above the column.
- Load new terminology/region context: click the Load button next to the context drop-down and select JSON from local storage.
- Create custom terminology/region context: Start from an existing JSON file, such as the DICOM master list for terminologies or region contexts. Remove the entries you do not need. Validate the JSON file with the validator online tool.

References

- DCMQI wiki

9.14.6 Texts

Overview

A module to create, edit and manage text data in the scene, such as short strings or text files (txt, xml, json).

Panels and their use

- **Contents**: Text can be viewed and edited in this section.
- **Advanced**
  - **Auto-save edits to the text node**: If enabled then content in the scene is immediately updated on each keypress. If disabled then **Edit** button must be pressed to enable editing the text and clicking **Save** or **Cancel** button finishes editing.
  - **Enable word wrapping**: Switch between wrapping long lines / displaying a horizontal scrollbar.

Contributors

- Kyle Sunderland (PerkLab, Queen's University)
- Andras Lasso (PerkLab, Queen's University)

Acknowledgements

This work was supported through CANARIE's Research Software Program, and Cancer Care Ontario.

9.14.7 DataProbe

Overview

The DataProbe module shows information about what is visible at the current mouse pointer position, and displays corner annotation (patient name, id, series date, description, etc.) in slice views.

Panels and their use

- **Enable slice view annotations**: Enable showing corner annotations in slice views.
- **Active corners**: What information is displayed in corner annotations.
- **Annotation display level**: Controls the amount of details displayed in corner annotations.
- **Background DICOM annotations persistence**: Enables displaying DICOM metadata if showing non-DICOM foreground image DICOM background image.

Contributors

- Steve Pieper (Isomics)

Acknowledgements

This work is supported by NA-MIC, NAC, NCIGT, NIH U24 CA180918 (PIs Kikinis and Fedorov) and the Slicer Community.

9.15 Registration

9.15.1 General Registration (BRAINS)

Overview

Register a three-dimensional volume to a reference volume (Mattes Mutual Information by default). Method described in BRAINSFit: Mutual Information Registrations of Whole-Brain 3D Images, Using the Insight Toolkit, Johnson H.J., Harris G., Williams K., The Insight Journal, 2007. https://hdl.handle.net/1926/1291

Panels and their use

Input Images:

- **Fixed Image Volume** (*fixedVolume*): Input fixed image (the moving image will be transformed into this image space).
- **Moving Image Volume** (*movingVolume*): Input moving image (this image will be transformed into the fixed image space).
- **Percentage Of Samples** (*samplingPercentage*): Fraction of voxels of the fixed image that will be used for registration. The number has to be larger than zero and less or equal to one. Higher values increase the computation time but may give more accurate results. You can also limit the sampling focus with ROI masks and ROIAUTO mask generation. The default is 0.002 (use approximately 0.2% of voxels, resulting in 100000 samples in a 512x512x192 volume) to provide a very fast registration in most cases. Typical values range from 0.01 (1%) for low detail images to 0.2 (20%) for high detail images.
- **B-Spline Grid Size** (*splineGridSize*): Number of BSpline grid subdivisions along each axis of the fixed image, centered on the image space. Values must be 3 or higher for the BSpline to be correctly computed.

Output Settings (At least one output must be specified):

- **Slicer Linear Transform** (*linearTransform*): (optional) Output estimated transform - in case the computed transform is not BSpline. NOTE: You must set at least one output object (transform and/or output volume).
- **Slicer BSpline Transform** (*bsplineTransform*): (optional) Output estimated transform - in case the computed transform is BSpline. NOTE: You must set at least one output object (transform and/or output volume).
- **Output Image Volume** (*outputVolume*): (optional) Output image: the moving image warped to the fixed image space. NOTE: You must set at least one output object (transform and/or output volume).

Transform Initialization Settings: Options for initializing transform parameters.

- **Initialization transform** (*initialTransform*): Transform to be applied to the moving image to initialize the registration. This can only be used if Initialize Transform Mode is Off.
- **Initialize Transform Mode** (*initializeTransformMode*): Determine how to initialize the transform center. useMomentsAlign assumes that the center of mass of the images represent similar structures. useCenterOfHeadAlign attempts to use the top of head and shape of neck to drive a center of mass estimate. useGeometryAlign on assumes that the center of the voxel lattice of the images represent similar structures. Off assumes that the physical space of the images are close. This flag is mutually exclusive with the Initialization transform.

Registration Phases (Check one or more, executed in order listed): Each of the registration phases will be used to initialize the next phase

- **Rigid (6 DOF)** (*useRigid*): Perform a rigid registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
- **Rigid+Scale(7 DOF)** (*useScaleVersor3D*): Perform a ScaleVersor3D registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
- **Rigid+Scale+Skew(10 DOF)** (*useScaleSkewVersor3D*): Perform a ScaleSkewVersor3D registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
- **Affine(12 DOF)** (*useAffine*): Perform an Affine registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
- **BSpline (>27 DOF)** (*useBSpline*): Perform a BSpline registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.

- **SyN** (*useSyN*): Perform a SyN registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.
- **Composite (many DOF)** (*useComposite*): Perform a Composite registration as part of the sequential registration steps. This family of options overrides the use of transformType if any of them are set.

Image Mask and Pre-Processing:

- **Masking Option** (*maskProcessingMode*): Specifies a mask to only consider a certain image region for the registration. If ROIAUTO is chosen, then the mask is computed using Otsu thresholding and hole filling. If ROI is chosen then the mask has to be specified as in input.
- **(ROI) Masking input fixed** (*fixedBinaryVolume*): Fixed Image binary mask volume, required if Masking Option is ROI. Image areas where the mask volume has zero value are ignored during the registration.
- **(ROI) Masking input moving** (*movingBinaryVolume*): Moving Image binary mask volume, required if Masking Option is ROI. Image areas where the mask volume has zero value are ignored during the registration.
- **(ROIAUTO) Output fixed mask** (*outputFixedVolumeROI*): ROI that is automatically computed from the fixed image. Only available if Masking Option is ROIAUTO. Image areas where the mask volume has zero value are ignored during the registration.
- **(ROIAUTO) Output moving mask** (*outputMovingVolumeROI*): ROI that is automatically computed from the moving image. Only available if Masking Option is ROIAUTO. Image areas where the mask volume has zero value are ignored during the registration.
- **Define BSpline grid over the ROI bounding box** (*useROIBSpline*): If enabled then the bounding box of the input ROIs defines the BSpline grid support region. Otherwise the BSpline grid support region is the whole fixed image.
- **Histogram Match** (*histogramMatch*): Apply histogram matching operation for the input images to make them more similar. This is suitable for images of the same modality that may have different brightness or contrast, but the same overall intensity profile. Do NOT use if registering images from different modalities.
- **Median Filter Size** (*medianFilterSize*): Apply median filtering to reduce noise in the input volumes. The 3 values specify the radius for the optional MedianImageFilter preprocessing in all 3 directions (in voxels).
- **Remove Intensity Outliers value at one tail** (*removeIntensityOutliers*): Remove very high and very low intensity voxels from the input volumes. The parameter specifies the half percentage to decide outliers of image intensities. The default value is zero, which means no outlier removal. If the value of 0.005 is given, the 0.005% of both tails will be thrown away, so 0.01% of intensities in total would be ignored in the statistic calculation.

Advanced Output Settings:

- **Fixed Image Volume 2** (*fixedVolume2*): Input fixed image that will be used for multimodal registration. (the moving image will be transformed into this image space).
- **Moving Image Volume2** (*movingVolume2*): Input moving image that will be used for multimodal registration(this image will be transformed into the fixed image space).
- **Output Image Pixel Type** (*outputVolumePixelType*): Data type for representing a voxel of the Output Volume.
- **Background Fill Value** (*backgroundFillValue*): This value will be used for filling those areas of the output image that have no corresponding voxels in the input moving image.
- **Scale Output Values** (*scaleOutputValues*): If true, and the voxel values do not fit within the minimum and maximum values of the desired outputVolumePixelType, then linearly scale the min/max output image voxel values to fit within the min/max range of the outputVolumePixelType.
- **Interpolation Mode** (*interpolationMode*): Type of interpolation to be used when applying transform to moving volume. Options are Linear, NearestNeighbor, BSpline, WindowedSinc, Hamming, Cosine, Welch, Lanczos, or

ResampleInPlace. The ResampleInPlace option will create an image with the same discrete voxel values and will adjust the origin and direction of the physical space interpretation.

Advanced Optimization Settings:

- **Max Iterations** (*numberOfIterations*): The maximum number of iterations to try before stopping the optimization. When using a lower value (500-1000) then the registration is forced to terminate earlier but there is a higher risk of stopping before an optimal solution is reached.
- **Maximum Step Length** (*maximumStepLength*): Starting step length of the optimizer. In general, higher values allow for recovering larger initial misalignments but there is an increased chance that the registration will not converge.
- **Minimum Step Length** (*minimumStepLength*): Each step in the optimization takes steps at least this big. When none are possible, registration is complete. Smaller values allows the optimizer to make smaller adjustments, but the registration time may increase.
- **Relaxation Factor** (*relaxationFactor*): Specifies how quickly the optimization step length is decreased during registration. The value must be larger than 0 and smaller than 1. Larger values result in slower step size decrease, which allow for recovering larger initial misalignments but it increases the registration time and the chance that the registration will not converge.
- **Transform Scale** (*translationScale*): How much to scale up changes in position (in mm) compared to unit rotational changes (in radians) – decrease this to allow for more rotation in the search pattern.
- **Reproportion Scale** (*reproportionScale*): ScaleVersor3D 'Scale' compensation factor. Increase this to allow for more rescaling in a ScaleVersor3D or ScaleSkewVersor3D search pattern. 1.0 works well with a translationScale of 1000.0
- **Skew Scale** (*skewScale*): ScaleSkewVersor3D Skew compensation factor. Increase this to allow for more skew in a ScaleSkewVersor3D search pattern. 1.0 works well with a translationScale of 1000.0
- **Maximum B-Spline Displacement** (*maxBSplineDisplacement*): Maximum allowed displacements in image physical coordinates (mm) for BSpline control grid along each axis. A value of 0.0 indicates that the problem should be unbounded. NOTE: This only constrains the BSpline portion, and does not limit the displacement from the associated bulk transform. This can lead to a substantial reduction in computation time in the BSpline optimizer.

Expert-only Parameters:

- **Fixed Image Time Index** (*fixedVolumeTimeIndex*): The index in the time series for the 3D fixed image to fit. Only allowed if the fixed input volume is 4-dimensional.
- **Moving Image Time Index** (*movingVolumeTimeIndex*): The index in the time series for the 3D moving image to fit. Only allowed if the moving input volume is 4-dimensional
- **Histogram bin count** (*numberOfHistogramBins*): The number of histogram levels used for mutual information metric estimation.
- **Histogram match point count** (*numberOfMatchPoints*): Number of histogram match points used for mutual information metric estimation.
- **Cost Metric** (*costMetric*): The cost metric to be used during fitting. Defaults to MMI. Options are MMI (Mattes Mutual Information), MSE (Mean Square Error), NC (Normalized Correlation), MC (Match Cardinality for binary images)
- **Inferior Cut Off From Center** (*maskInferiorCutOffFromCenter*): If Initialize Transform Mode is set to useCenterOfHeadAlign or Masking Option is ROIAUTO then this value defines the how much is cut of from the inferior part of the image. The cut-off distance is specified in millimeters, relative to the image center. If the value is 1000 or larger then no cut-off performed.

- **ROIAuto Dilate Size** (*ROIAutoDilateSize*): This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the final dilation size to capture a bit of background outside the tissue region. A setting of 10mm has been shown to help regularize a BSpline registration type so that there is some background constraints to match the edges of the head better.
- **ROIAuto Closing Size** (*ROIAutoClosingSize*): This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the hole closing size in mm. It is rounded up to the nearest whole pixel size in each direction. The default is to use a closing size of 9mm. For mouse data this value may need to be reset to 0.9 or smaller.
- **Number Of Samples** (*numberOfSamples*): The number of voxels sampled for mutual information computation. Increase this for higher accuracy, at the cost of longer computation time.\nNOTE that it is suggested to use samplingPercentage instead of this option. However, if set to non-zero, numberOfSamples overwrites the samplingPercentage option.
- **Stripped Output Transform** (*strippedOutputTransform*): Rigid component of the estimated affine transform. Can be used to rigidly register the moving image to the fixed image. NOTE: This value is overridden if either bsplineTransform or linearTransform is set.
- **Transform Type** (*transformType*): Specifies a list of registration types to be used. The valid types are, Rigid, ScaleVersor3D, ScaleSkewVersor3D, Affine, BSpline and SyN. Specifying more than one in a comma separated list will initialize the next stage with the previous results. If registrationClass flag is used, it overrides this parameter setting.
- **Output Transform** (*outputTransform*): (optional) Filename to which save the (optional) estimated transform. NOTE: You must select either the outputTransform or the outputVolume option.
- **Pass warped moving image to BSpline registration filter** (*initializeRegistrationByCurrentGenericTransform*): If this flag is ON, the current generic composite transform, resulted from the linear registration stages, is set to initialize the follow nonlinear registration process. However, by the default behavior, the moving image is first warped based on the existent transform before it is passed to the BSpline registration filter. It is done to speed up the BSpline registration by reducing the computations of composite transform Jacobian.
- **writes the output registration transforms in single precision** (*writeOutputTransformInFloat*): By default, the output registration transforms (either the output composite transform or each transform component) are written to the disk in double precision. If this flag is ON, the output transforms will be written in single (float) precision. It is especially important if the output transform is a displacement field transform, or it is a composite transform that includes several displacement fields.

Debugging Parameters:

- **Failure Exit Code** (*failureExitCode*): If the fit fails, exit with this status code. (It can be used to force a successfult exit status of (0) if the registration fails due to reaching the maximum number of iterations.
- **Write Transform On Failure** (*writeTransformOnFailure*): Flag to save the final transform even if the numberOfIterations are reached without convergence. (Intended for use when –failureExitCode 0)
- **Number Of Threads** (*numberOfThreads*): Explicitly specify the maximum number of threads to use. (default is auto-detected)
- **Debug option** (*debugLevel*): Display debug messages, and produce debug intermediate results. 0=OFF, 1=Minimal, 10=Maximum debugging.
- **Set Sampling Strategy** (*metricSamplingStrategy*): It defines the method that registration filter uses to sample the input fixed image. Only Random is supported for now.
- **Log File Report** (*logFileReport*): A file to write out final information report in CSV file: MetricName,MetricValue,FixedImageName,FixedMaskName,MovingImageName,MovingMaskName

Contributors

Hans J. Johnson (hans-johnson -at- uiowa.edu, https://www.psychiatry.uiowa.edu), Ali Ghayoor

Acknowledgements

Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5); Ali Ghayoor(4) 1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard

Use cases

Most frequently used for these scenarios and recommended registration settings.

Same Subject: Longitudinal

For this case we're registering a baseline T1 scan with a follow-up T1 scan on the same subject a year later.

First, set the fixed and moving volumes:

```
--fixedVolume test.nii.gz \
--movingVolume test2.nii.gz \
```

Next, set the output transform and volume:

```
--outputVolume testT1LongRegFixed.nii.gz \
--outputTransform longToBase.xform \
```

Since the input scans are of the same subject we can assume very little has changed in the last year, so we'll use a Rigid registration.

```
--transformType Rigid \
```

Note

If the registration is poor or there are reasons to expect anatomical changes (tumor growth, rapid disease progression, etc.) additional transforms may be needed. In that case they can be added in a comma separated list, such as "Rigid,ScaleVersor3D,ScaleSkewVersor3D,Affine,BSpline". Available methods are:

- Rigid
- ScaleVersor3D
- ScaleSkewVersor3D
- Affine
- BSpline

Example: multiple registration methods

```
--transformType Rigid,ScaleVersor3D,ScaleSkewVersor3D,Affine,BSpline \
```

The scans are the same modality so we'll use –histogramMatch to match the intensity profiles as this tends to help the registration. If there are lesions or tumors that vary between images this may not be a good idea, since it will make it harder to detect differences between the images.

```
--histogramMatch \
```

To start with the best possible initial alignment we use –initializeTransformMode. The available transform modes are:

- useCenterOfHeadAlign
- useCenterOfROIAlign
- useMomentsAlign
- useGeometryAlign

We're working with human heads so we pick useCenterOfHeadAlign, which detects the center of head even with varying amounts of neck or shoulders present.

```
--initializeTransformMode useCenterOfHeadAlign \
```

ROI masks normally improve registration but we haven't generated any so we turn on –maskProcessingMode ROIAUTO (other options are NOMASK and ROI).

```
--maskProcessingMode ROIAUTO \
```

The registration generally performs better if we include some background in the mask to make the tissue boundary very clear. The parameter that expands the mask outside the brain is ROIAutoDilateSize (under "Registration Debugging Parameters" if using the GUI). These values are in millimeters and a good starting value is 3.

```
--ROIAutoDilateSize 3 \
```

Lastly, we set the interpolation mode to be Linear, which is a decent tradeoff between quality and speed. If the best possible interpolation is needed regardless of processing time, select WindowedSync instead.

```
--interpolationMode Linear
```

The full command is:

```
BRAINSFit --fixedVolume test.nii.gz \
    --movingVolume test2.nii.gz \
    --outputVolume testT1LongRegFixed.nii.gz \
    --outputTransform longToBase.xform \
    --transformType Rigid \
    --histogramMatch \
    --initializeTransformMode useCenterOfHeadAlign \
    --maskProcessingMode ROIAUTO \
    --ROIAutoDilateSize 3 \
    --interpolationMode Linear
```

Same Subject: MultiModal

For this use case we're registering a T1 scan with a T2 scan collected in the same session. The two images are again available on the Midas site as test.nii.gz and standard.nii.gz

First we set the fixed and moving volumes as well as the output transform and output volume names.

```
--fixedVolume test.nii.gz \
--movingVolume standard.nii.gz \
--outputVolume testT2RegT1.nii.gz \
--outputTransform T2ToT1.xform \
```

Since these are the same subject, same session we'll use a Rigid registration.

```
--transformType Rigid \
```

The scans are different modalities so we absolutely DO NOT want to use –histogramMatch to match the intensity profiles! This would try to map T2 intensities into T1 intensities resulting in an image that is neither, and hence useless.

To start with the best possible initial alignment we use –initializeTransformMode. We're working with human heads so we pick useCenterOfHeadAlign, which detects the center of head even with varying amounts of neck or shoulders present.

```
--initializeTransformMode useCenterOfHeadAlign \
```

ROI masks normally improve registration but we haven't generated any so we turn on –maskProcessingMode ROIAUTO (other options are NOMASK and ROI).

```
--maskProcessingMode ROIAUTO \
```

The registration generally performs better if we include some background in the mask to make the tissue boundary very clear. The parameter that expands the mask outside the brain is ROIAutoDilateSize (under "Registration Debugging Parameters" if using the GUI). These values are in millimeters and a good starting value is 3.

```
--ROIAutoDilateSize 3 \
```

Lastly, we set the interpolation mode to be Linear, which is a decent tradeoff between quality and speed. If the best possible interpolation is needed regardless of processing time, select WindowedSync instead.

```
--interpolationMode Linear
```

The full command is:

```
BRAINSFit --fixedVolume test.nii.gz \
    --movingVolume standard.nii.gz \
    --outputVolume testT2RegT1.nii.gz \
    --outputTransform T2ToT1.xform \
    --transformType Rigid \
    --initializeTransformMode useCenterOfHeadAlign \
    --maskProcessingMode ROIAUTO \
    --ROIAutoDilateSize 3 \
    --interpolationMode Linear
```

Mouse Registration

Here we'll register brains from two different mice together. The fixed and moving mouse brains used in this example are available on the Midas site as mouseFixed.nii.gz and mouseMoving.nii.gz.

First we set the fixed and moving volumes as well as the output transform and output volume names.

```
--fixedVolume mouseFixed.nii.gz \
--movingVolume mouseMoving.nii.gz \
--outputVolume movingRegFixed.nii.gz \
--outputTransform movingToFixed.xform \
```

Since the subjects are different we are going to use transforms all the way through BSpline.

Note

Building up transforms one type at a time can't hurt and might help, so we're including all transforms from Rigid through BSpline in the transformType parameter.

```
--transformType Rigid,ScaleVersor3D,ScaleSkewVersor3D,Affine,BSpline \
```

The scans are the same modality so we'll use –histogramMatch.

```
--histogramMatch \
```

To start with the best possible initial alignment we use –initializeTransformMode but we aren't working with human heads this time, so we can't pick useCenterOfHeadAlign! Instead we pick useMomentsAlign which does a reasonable job of selecting the centers of mass.

```
--initializeTransformMode useMomentsAlign \
```

ROI masks normally improve registration but we haven't generated any so we turn on –maskProcessingMode ROIAUTO.

```
--maskProcessingMode ROIAUTO \
```

Since the mouse brains are much smaller than human brains there are a few advanced parameters we'll need to tweak, ROIAutoClosingSize and ROIAutoDilateSize (both under Registration Debugging Parameters if using the GUI). These values are in millimeters and a good starting value for mice is 0.9.

```
--ROIAutoClosingSize 0.9 \
--ROIAutoDilateSize 0.9 \
```

Lastly, we set the interpolation mode to be Linear, which is a decent tradeoff between quality and speed. If the best possible interpolation is needed regardless of processing time, select WindowedSync instead.

```
--interpolationMode Linear
```

The full command is:

```
BRAINSFit --fixedVolume mouseFixed.nii.gz \
    --movingVolume mouseMoving.nii.gz \
    --outputVolume movingRegFixed.nii.gz \
    --outputTransform movingToFixed.xform \
    --transformType Rigid,ScaleVersor3D,ScaleSkewVersor3D,Affine,BSpline \
    --histogramMatch \
    --initializeTransformMode useMomentsAlign \
    --maskProcessingMode ROIAUTO \
    --ROIAutoClosingSize 0.9 \
    --ROIAutoDilateSize 0.9 \
    --interpolationMode Linear
```

References

- BRAINSFit: Mutual Information Registrations of Whole-Brain 3D Images, Using the Insight Toolkit, Johnson H.J., Harris G., Williams K., The Insight Journal, 2007.
- Source code on github

9.15.2 Resample Image (BRAINS)

Overview

```
This program collects together three common image processing tasks that all involve␣
↪resampling an image volume: Resampling to a new resolution and spacing, applying a␣
↪transformation (using an ITK transform IO mechanisms) and Warping (using a vector␣
↪image deformation field).
```

Use cases

Most frequently used for these scenarios:

- Change an image's resolution and spacing.
- Apply a transformation to an image (using an ITK transform IO mechanisms)
- Warping an image (using a vector image deformation field).

Interpolation types

- NearestNeighbor: The value of the nearest voxel is copied into the new voxel
- Linear: The average of the voxels in the input image occupying the new voxel volume is used
- ResampleInPlace: Detailed information can be found here.
- BSpline: Detailed information can be found here.
- WindowedSinc: Detailed information can be found here.

Panels and their use

Inputs: Parameters for specifying the image to warp and resulting image space

- **Image To Warp** (*inputVolume*): Image To Warp
- **Reference Image** (*referenceVolume*): Reference image used only to define the output space. If not specified, the warping is done in the same space as the image to warp.

Outputs: Resulting deformed image parameters

- **Output Image** (*outputVolume*): Resulting deformed image
- **Pixel Type** (*pixelType*): Specifies the pixel type for the input/output images. If the type is "input", then infer from the input image. The "binary" pixel type uses a modified algorithm whereby the image is read in as unsigned char, a signed distance map is created, signed distance map is resampled, and then a thresholded image of type unsigned char is written to disk.

Warping Parameters: Parameters used to define home the image is warped

- **Displacement Field (deprecated)** (*deformationVolume*): Displacement Field to be used to warp the image (ITKv3 or earlier)
- **Transform file** (*warpTransform*): Filename for the BRAINSFit transform (ITKv3 or earlier) or composite transform file (ITKv4)
- **Interpolation Mode** (*interpolationMode*): Type of interpolation to be used when applying transform to moving volume. Options are Linear, ResampleInPlace, NearestNeighbor, BSpline, or WindowedSinc

- **Compute inverse transform of given transformation?** (*inverseTransform*): True/False is to compute inverse of given transformation. Default is false
- **Default Value** (*defaultValue*): Default voxel value

Advanced Options:

- **Add Grids** (*gridSpacing*): Add warped grid to output image to help show the deformation that occurred with specified spacing. A spacing of 0 in a dimension indicates that grid lines should be rendered to fall exactly (i.e. do not allow displacements off that plane). This is useful for making a 2D image of grid lines from the 3D space

Multiprocessing Control:

- **Number Of Threads** (*numberOfThreads*): Explicitly specify the maximum number of threads to use.

Contributors

This tool was developed by Vincent Magnotta, Greg Harris, and Hans Johnson.

Acknowledgements

The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Similar modules

- *Resample Scalar/Vector/DWI Volume*
- *Resample Scalar Volume*
- *Resample DTI Volume*

9.15.3 Resize Image (BRAINS)

Overview

This program is useful for downsampling an image by a constant scale factor.

Panels and their use

Inputs: Parameters for specifying the image to warp and resulting image space

- **Image To Warp** (*inputVolume*): Image To Scale

Outputs: Resulting scaled image parameters

- **Output Image** (*outputVolume*): Resulting scaled image
- **Pixel Type** (*pixelType*): Specifies the pixel type for the input/output images. The "binary" pixel type uses a modified algorithm whereby the image is read in as unsigned char, a signed distance map is created, signed distance map is resampled, and then a thresholded image of type unsigned char is written to disk.

Scaling Parameters: Parameters used to define the scaling of the output image

- **Scale Factor** (*scaleFactor*): The scale factor for the image spacing.

Contributors

This tool was developed by Hans Johnson.

Acknowledgements

The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

9.15.4 Fiducial Registration

Overview

Computes a rigid, similarity or affine transform from a matched list of fiducials

Panels and their use

IO: Input/output parameters

- **Fixed landmarks** (*fixedLandmarks*): Ordered list of landmarks in the fixed image
- **Moving landmarks** (*movingLandmarks*): Ordered list of landmarks in the moving image
- **Save transform** (*saveTransform*): Save the transform that results from registration
- **Transform Type** (*transformType*): Type of transform to produce
- **RMS Error** (*rms*): Display RMS Error.
- **Output Message** (*outputMessage*): Provides more information on the output

Contributors

Casey B Goodlett (Kitware), Dominik Meier (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.

9.15.5 Landmark Registration

Overview

This module aligns two images based on a set of corresponding landmarks (paired points).

How to

Linear registration

- Load the two images to register
- Enter the Landmark Registration module
- Select the two images as fixed and moving volumes (do not select transformed volume)
- Scroll to the `Registration` area and select `Affine registration`
- Pick Axi/Sag/Cor in the Visualization box (this will create a custom layout with fixed on top, moving in the middle, and fixed + transformed on the bottom)
- Place a point on either the fixed or moving volumes (a corresponding one will be created on the other volume)

- Drag the points in the fixed and moving volumes until they are on the same anatomical location. The blended view will update automatically on mouse release.
- Place and adjust points until registration is good.
- Optional: set `Local Refinement Method` to `Local SimpleITK` (it tends to be more robust than `Local BRAINSFit`) and click on the `Refine landmark ...` button.

Chose registration type

Similarity mode is Rigid + Scale and can be good for some cross-subject registration.

Affine mode requires more landmarks but should work.

Thin-Plate spline mode works but does not automatically update (click Apply to calculate). It overwrites the transformed volume so you can't go back to Linear mode from Thin-Plate mode.

Panels and their use

- **Fixed volume** and **Moving volume**: the computed transformation will be computed to transform the moving image into the fixed image.
- **Transformed volume**: output image, obtained by applying the computed transform to the moving volume and resampled to the fixed volume.
- **Target transform**: computed transform that aligns the moving volume with the fixed volume.
- **Visualization**: view layout and view mode to visualize the alignment.
- **Landmarks**: list of corresponding landmark points.
- **Local refinement**: automatically adjust the position of a placed landmark point in the moving image. `Local SimpleITK` method tends to be more robust.
- **Registration type**: choose between linear transform (`Affine Registration`) and warping transform (`ThinPlate Registration`).

Contributors

- Steve Pieper (Isomics)

Acknowledgements

This file was originally developed by Steve Pieper, Isomics, Inc. It was partially funded by NIH grant 3P41RR013218-12S1 and P41 EB015902 the Neuroimage Analysis Center (NAC) a Biomedical Technology Resource Center supported by the National Institute of Biomedical Imaging and Bioengineering (NIBIB). And 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. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics. This work is also supported by NIH grant 1R01DE024450-01A1 "Quantification of 3D Bony Changes in Temporomandibular Joint Osteoarthritis" (TMJ-OA).

9.15.6 Registration Metric Test (BRAINS)

Overview

Compare Mattes/MSQ metric value for two input images and a possible input BSpline transform.

Panels and their use

IO: Input parameters

- **Transform File Name** (*inputBSplineTransform*): Input transform that is use to warp moving image before metric comparison.
- **Fixed image** (*inputFixedImage*):
- **Moving image** (*inputMovingImage*):

Input variables: Metric type and input parameters.

- **Metric type** (*metricType*): Comparison metric type
- **Number Of Samples** (*numberOfSamples*): The number of voxels sampled for metric evaluation.
- **Number Of Histogram Bins** (*numberOfHistogramBins*): The number of histogram bins when MMI (Mattes) is metric type.

Contributors

Ali Ghayoor

Acknowledgements

9.15.7 Reformat

Overview

This module is used for changing the slice properties.

Panels and their use

- **Slice**: Select the slice to operate on – the module's Display interface will change to show slice parameters.
- **Display - Edit**: Information about the selected slice. Fields can be edited to precisely set values to the slice.
  - **Offset**: See and Set the current distance from the origin to the slice plane
  - **Origin**: The location of the center of the slice. It is also related to the reformat widget origin associated to the selected slice.
    - **Center**: This button will adjust the slice Origin so that the entire slice is centered around 0,0,0 in the volume space.
  - **Orientation**
    - **Reset to**: Reset the slice to transformation to the corresponding orientation preset, such as "Axial, "Sagittal", "Coronal" or "Reformat".
    - **Rotate to volume plane**: Rotates the slice view to be aligned with the axes of the displayed volume.
    - **Flip H** and **Flip V**: Flip the image slice horizontally or vertically.
    - **Rotate CW** and **Rotate CCW**: Rotate the slice in-plane by 90 degrees in clockwise or counterclockwise direction.
    - **Normal**: Allow to set the slice plane normal direction.
      - **Normal to LR**: Set the normal to left-right anatomical direction.
      - **Normal to PA**: Set the normal to posterior-anterior anatomical direction.
      - **Normal to IS**: Set the normal to inferior-superior anatomical direction.

    * **Normal to camera**: Align the slice normal to the camera view direction.
    * **Rotation**
        - **Horizontal**: Free rotation of the slice around its horizontal axis.
        - **Vertical**: Free rotation of the slice around its vertical axis.
        - **In-Plane**: Free in-plane rotation of the slice (around its normal).

Contributors

Michael Jeulin-Lagarrigue (Kitware)

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.

9.16 Segmentation

9.16.1 Foreground masking (BRAINS)

Overview

This program is used to create a mask over the most prominent foreground region in an image. This is accomplished via a combination of otsu thresholding and a closing operation.

Panels and their use

IO: Input/output parameters

- **Input Image Volume** (*inputVolume*): The input image for finding the largest region filled mask.
- **Output Mask** (*outputROIMaskVolume*): The ROI automatically found from the input image.
- **Output Image** (*outputVolume*): The inputVolume with optional [maskOutput|cropOutput] to the region of the brain mask.
- **Mask Output** (*maskOutput*): The inputVolume multiplied by the ROI mask.
- **Output Image Clipped by ROI** (*cropOutput*): The inputVolume cropped to the region of the ROI mask.

Configuration Parameters:

- **Otsu Percentile Threshold** (*otsuPercentileThreshold*): Parameter to the Otsu threshold algorithm.
- **Otsu Correction Factor** (*thresholdCorrectionFactor*): A factor to scale the Otsu algorithm's result threshold, in case clipping mangles the image.
- **Closing Size** (*closingSize*): The Closing Size (in millimeters) for largest connected filled mask. This value is divided by image spacing and rounded to the next largest voxel number.
- **ROIAuto Dilate Size** (*ROIAutoDilateSize*): This flag is only relevant when using ROIAUTO mode for initializing masks. It defines the final dilation size to capture a bit of background outside the tissue region. At setting of 10mm has been shown to help regularize a BSpline registration type so that there is some background constraints to match the edges of the head better.
- **Output Image Pixel Type** (*outputVolumePixelType*): The output image Pixel Type is the scalar datatype for representation of the Output Volume.

- **Number Of Threads** (*numberOfThreads*): Explicitly specify the maximum number of threads to use.

Contributors

Hans J. Johnson, hans-johnson -at- uiowa.edu, https://www.psychiatry.uiowa.edu

Acknowledgements

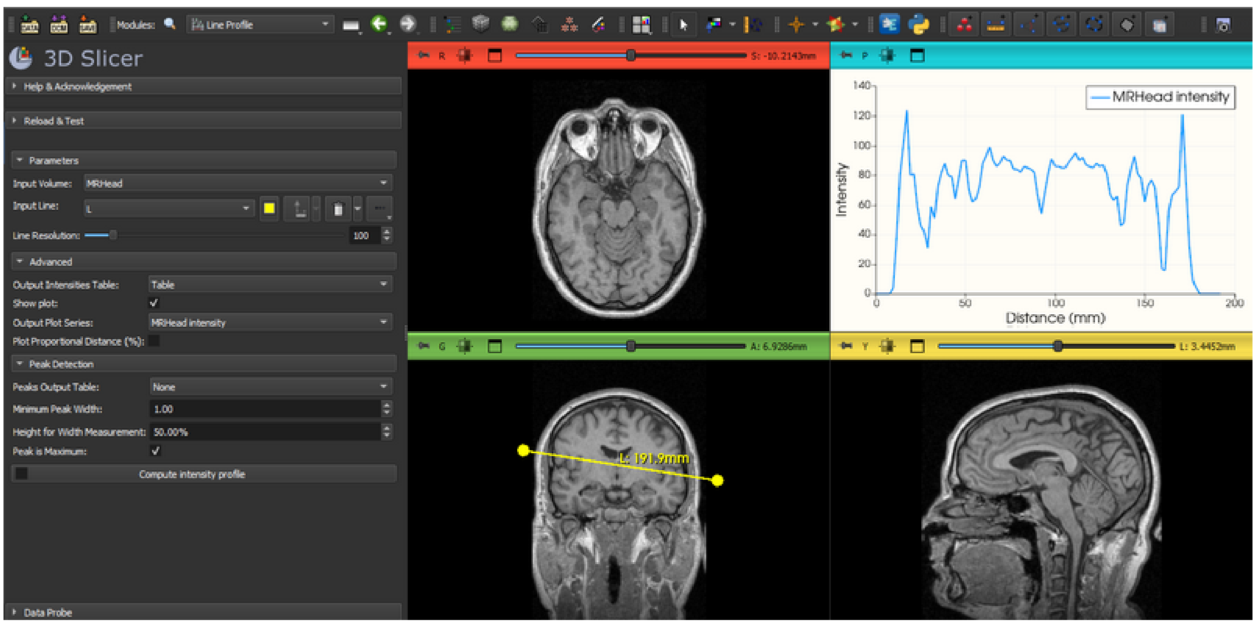
Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5), fedorov -at- bwh.harvard.edu (Slicer integration); (1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard)

9.17 Quantification

9.17.1 Line Profile

Overview

The Line Profile module can compute intensity profile from an image along a line or curve.



Panels and their use

- **Parameters**:
  - **Input volume**: The intensity profile will be computed from voxel values of this image.
  - **Input line**: The image will be sampled along this line or curve. Click and select `Create new line` or `Create new curve` and the click in the views.
  - **Line resolution**: Specifies how many samples will be taken from the image along the input line.
- **Advanced**:
  - **Output intensities table**: Intensity values will be stored in this table.
  - **Show plot**: If this option is enabled then any updates will automatically display the intensity plot in the view layout.

- **Output plot series**: This series will be set up to plot the intensity values.
- **Plot proportional distance (%)**: If this option is enabled then distance values are relative (in percentage) along the line.

- **Peak detection**:
  - **Peaks output table**: Peak detection results will be stored in this table.
  - **Minimum peak width**: Minimum width of the peak. Use higher values to reject small peaks detected due to image noise.
  - **Height for width measurement**: Height at which the peak width is measured. 50% computes full width at half maximum. Larger value means that the height is measured near the top of the peak.
  - **Peak is maximum**: If checked then peaks are local maximum values. If unchecked then peaks are local minimum values.
- **Compute intensity profile**: Click the button to compute the intensity profile. Check the checkbox in the button to enable auto-update, i.e., automatically update the intensity profile when the line or any parameter is changed.

Contributors

Andras Lasso (PerkLab), Natalie Johnston (Revvity), James Butler (Revvity), Mike Bindschadler, Adam Aji, chir.set, Csaba Pinter (Ebatinca)

Acknowledgements

This work was was partially funded by Revvity and Children's Hospital of Philadelphia.

9.17.2 PET Standard Uptake Value Computation

Overview

Computes the standardized uptake value based on body weight. Takes an input PET image in DICOM and NRRD format (DICOM header must contain Radiopharmaceutical parameters). Produces a CSV file that contains patientID, studyDate, dose, labelID, suvmin, suvmax, suvmean, labelName for each volume of interest. It also displays some of the information as output strings in the GUI, the CSV file is optional in that case. The CSV file is appended to on each execution of the CLI.

Panels and their use

Image and Information: Input parameters

- **PET DICOM volume path** (*PETDICOMPath*): Input path to a directory containing a PET volume containing DICOM header information for SUV computation
- **Input PET Volume** (*PETVolume*): Input PET volume for SUVbw computation (must be the same volume as pointed to by the DICOM path!).
- **Input VOI Volume** (*VOIVolume*): Input label volume containing the volumes of interest

Output: The Output file collects the information on disk from the output label, suv max/mean/min output stringsin the gui, plus some extra information from the DICOM header.

- **Output table** (*OutputCSV*): A table holding the output SUV values in comma separated lines, one per label. Optional.
- **Output Label** (*OutputLabel*): List of labels for which SUV values were computed
- **Output Label Value** (*OutputLabelValue*): List of label values for which SUV values were computed

- **SUV Max** (*SUVMax*): SUV max for each label
- **SUV Mean** (*SUVMean*): SUV mean for each label
- **SUV Minimum** (*SUVMin*): SUV minimum for each label

Contributors

Wendy Plesniak (SPL, BWH), Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)

Acknowledgements

This work is funded by the Harvard Catalyst, and 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.

9.17.3 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.

Computed metrics

Labelmap statistics

The values are computed from 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 surface area 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: a measure of how flat the segment is. Calculated as the square root of the ratio of the second-smallest principal moment to the smallest. A value of 1 represents a non-flat structure (e.g., sphere or cigar); larger values indicate a flatter structure. See detailed definition here.
- Elongation: a measure of how stretched the segment is along its longest axis relative to its second-longest. Calculated as the square root of the ratio of the largest principal moment to the second-largest. A value of 1 means the segment has equal extent along its two longest axes (e.g., a sphere or a rod with circular cross-section); larger values indicate a more elongated shape. Note that this metric does not compare the longest axis to the shortest — for that, see Flatness. 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. Principal axis directions are used to orient the bounding box.

Scalar volume statistics

The values are computed from the binary labelmap representation of the segment, for the part that overlaps with the chosen scalar volume.

- Voxel count: the number of voxels in the segment
- Volume mm3: volume of that part of the segment that overlaps with the chosen scalar volume, in mm3
- Volume cm3: volume of that part of the segment that overlaps with the chosen scalar volume, in cm3
- Minimum: the minimum scalar value in the segment
- Maximum: the maximum scalar value in the segment
- Mean: the mean scalar value in the segment
- Median: the median scalar value in the segment
- Standard deviation: the standard deviation of scalar values in the segment (computed using *corrected sample standard deviation* formula)

Closed surface statistics

The values are computed from the closed surface representation of the segment.

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

Frequently asked questions

What is the difference between the surface and volume values computed by various plugins?

There are several ways of computing volume and surface of a segment. The main difference relates to the representation being used (3D binary image or surface mesh) and if the entire segment is used or only the part that overlaps with the selected scalar volume. Difference between the values should be very small, less than one percent, so it usually does not matter which one is used. Usually it does not matter which plugin is used, but to minimize variance between values, it is recommended to consistently use the same plugin within a study.

How to choose between plugins:

- The `Labelmap` plugin is a good choice for computing the volume of a segment for most cases. The volume is computed from the number of voxels multiplied by the volume of a singlevoxel. By default, surface computation in `Labelmap` plugin is disabled - either enable `Surface mm2` measurement in advanced settings to make this plugin compute surface area; or click `Show 3D` button to create closed surface representation for the segment and then get surface area from `Closed surface` plugin.
- If primarily the closed surface representation of the segmentation is used (e.g., 3D visualization, 3D printing) then it may be more appropriate to use the `Closed surface` plugin to compute both the volume and surface of the segment. The values are computed from the closed surface representation of the segmentation that is shown in 3D views.
- If the scalar volume input is set in the module then the `Scalar volume` plugin computes image intensity statistics for each segment. In this case, using values provided by the `Scalar volume` plugin makes sense. Surface and volume values are computed by the same method as in `Labelmap` plugin, the only difference is that the values are computed for only that part of the segments that overlap with the scalar volume.

How to get cross-sectional area of a segment along a certain axis?

There are several modules in various extensions to compute cross-sectional areas in segmentations:

- **in axial slices:** you can use the `Segment Cross-Section Area` module in Sandbox extension.
- **along a line in arbitrary direction:** you can use `SegmentGeometry` module in SlicerBiomech extension.
- **along a curve:** you can segment the region and use `Cross-section analysis` or `Stenosis measurement` modules in VMTK extension.
- **in a single slice (either axial oblique):** you can use a closed curve markup (enable area measurement in `Measurements` section in `Markups` module).

Related Modules

- *Segmentations* module allows changing conversion options, such as decimation and smoothing when converting from labelmap to closed surface representations, which are mainly for visualization, but can have an impact on some statistics such as volume and surface area. The Segmentations module can also be used for 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

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

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 (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

9.18 Sequences

9.18.1 Sequences

Overview

The Sequences module can create and visualize sequences of nodes, for example time 4D CT, cine-MRI, 4D ultrasound, or navigated 2D ultrasound. The module is not limited to just image sequences as it works for sequences of any other nodes (transforms, markups, …) including their display properties. Multiple sequences can be replayed in real-time, optionally synchronized, to visualize the contents in 2D and 3D.

Use cases

Explore Sequences module using sample data

Option A. Load sample data sets using "Sample Data" module

- Go to the Sample Data module and click on one of these data sets:
  - CTPCardioSeq: cardiac 4D perfusion CT
  - CTCardioSeq: cardiac 4D coronary CT
- Use the toolbar to start replay or browse time points

Option B. Download scene files and load them into the application

- Download one of these scene files:
  - Deformation of a 3D model
  - Ultrasound-guided needle insertion: moving tracked ultrasound image and tools, synchronized replay of image ad transforms
- Load the downloaded .mrb scene file into Slicer by drag-and-dropping the file to the application window then clicking OK
- Use the toolbar to start replay or browse time points

Recording node changes into a sequence node

- Go to Sequences module.
- If a node is not selected in the *Sequence browser* selector then click on it and choose *Create new SequenceBrowser*.
- Click the green + button next to *(new sequence)*. This creates a new sequence that will store the segmentation for each timepoint.
- In the *Proxy node* column and in the last row of the table, choose the node that you want to record changes. This indicates that this sequence will store states of the chosen segmentation node.
- Check the *Save changes* checkbox to allow modifying the sequence by editing the segmentation node.- Click the record button (red dot) in the sequence browser toolbar and start modifying the node to record changes.
- Click the record button (red dot) in the sequence browser toolbar and start modifying the node to record changes.
- Click the stop button in the sequence browser toolbar to stop recording.
- Move the slider or click the play button in the sequence browser toolbar to review recorded data.

Creating sequences from a set of nodes

- Load all your nodes (volumes, models, etc.) into Slicer - these will be referred to as *data nodes*
- Open the Sequences module
- Switch to the Sequences tab
- Click *Select a Sequence*, choose *Create new Sequence*
- In the *Add/remove data nodes* section select your first data node in the list and click the left arrow button
- Repeat this for all data nodes: select the next data node and click the left arrow button (Slicer will automatically jump to the next data node of the same type, so you may need to keep clicking the arrow button)
- To replay the sequence that you have just created: switch to *Sequence browser* tab
- Click on *Select a Sequence* in the Master node list and select your sequence node (it is called *Sequence* by default)
- Press the play button to start replay of the data
- Go to the Data module and select all input *data nodes*. Right-click and choose to delete them, to prevent them from occluding the view.
- To visualize a volume in 2D: drag-and-drop the *Sequence [time=…]* node into a slice view
- To visualize a volume in 3D viewer using volume rendering: drag-and-drop the *Sequence [time=…]* node into a 3D view

Create a segmentation node sequence

To allow segmenting each time point of the image, you need to create a segmentation sequence:

- Create a new Segmentation node (e.g., by segmenting the image at one timepoint)
- Go to the Sequences module
- Click the green + button next to *(new sequence)*. This creates a new sequence that will store the segmentation for each timepoint.
- Choose your segmentation node in the *Proxy node* column and in the last row of the table. This indicates that this sequence will store states of the chosen segmentation node.
- Check the *Save changes* checkbox to allow modifying the sequence by editing the segmentation node.

Convert MultiVolume node to Sequence node

If your data is in a *MultiVolume* node, you can convert it to a *Sequence* node by following these steps:

- Save your 4D volume (to a .nrrd file)
- Load the saved .nrrd file as a sequence node: in the Add data dialog, select *Volume Sequence* in the Description column

Load DICOM file as Sequence node

In Application settings / DICOM / MultiVolumeImporterPlugin / Preferred multi-volume import format, select "volume sequence". After this, volume sequences will be loaded as Sequence nodes by default.

It is also possible to choose the import format for each loaded DICOM data set, by following these steps:

- Open the DICOM browser, select data set to load
- Check the "Advanced" checkbox
- Click "Examine"
- In the populated table, check the checkbox in the row that contains "… frames Volume Sequence by …" (to load data set as multi-volume node, select row "… frames Multivolume by …")

Definitions

- Sequence: Contains an ordered array of data nodes, each data node is tagged with an index value.
- Data node: A regular MRML node, one item in the sequence. Data nodes are stored privately inside the sequence, therefore they are not visible in the main scene (where the sequence node is located). Singleton nodes are not allowed to be stored as data nodes. Sequence nodes can be data nodes, therefore a sequence of sequence nodes can be used to represent higher-dimensional data sets.
- Sequence index: The index describes the dimension of the data node sequence. The index name (such as "time"), unit (such as "s"), and type (such as "numeric" or "text") is the same for the whole sequence. The index value is specified for each data node. The index type information is used for sorting (numerical or string sorting) and for matching the index values (in the case of a numerical index we can find the closest data node even if there is no perfectly matching index value).
- Sequence browsing: A sequence node only contains the data nodes, but does not store any node relationships, such as parent transform, display properties, etc. These relationships can be only defined for the virtual output nodes that are generated by the Sequence browser module. Several browser nodes can be created to visualize data from the same sequence to support comparing multiple different time points from the same sequence.
- Proxy node: The sequence browser node creates a copy of the selected privately stored data node in the main scene. This copy is the proxy node (formerly *virtual output node*).

Contributors

Authors:

- Andras Lasso (PerkLab, Queen's University)
- Matthew Holden (PerkLab, Queen's University)
- Kyle Sunderland (PerkLab, Queen's University)
- Kevin Wang (Princess Margaret Cancer Centre)
- Gabor Fichtinger (PerkLab, Queen's University)

Acknowledgements

This work is funded in part 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 a free, open-source toolset for radiotherapy and related image-guided interventions.

9.18.2 Crop volume sequence

Overview

This module can crop&resample all the volumes of a volume sequence using the same region.

Panels and their use

- **Parameters**:
  - **Input volume sequence**: sequence node that contain volume nodes that will be cropped
  - **Output volume sequence**: sequence node to store the cropped input volume sequence
  - **Crop volume settings**: crop&resample settings. Click the green arrow button to go to the *Crop volume* module to edit these settings.

Related extensions and modules

This module internally uses *Crop volume* module to apply crop&resample operation on each volume of the sequence.

Contributors

- Andras Lasso (PerkLab)

Acknowledgements

This work was partially supported by Department of Anesthesia and Critical Care at The Children's Hospital of Philadelphia (CHOP).

9.18.3 MultiVolumeImporter

Overview

This module can load multiple images - referred to as "frames" - as a multi-volume. Each frame can be a 2D or 3D image, must have the same geometry (origin, spacing, axis directions, extents), and have only a single scalar component.

> **Note**
>
> This module is being phased out. It will be replaced by the more generic *Sequences* module, which can handle any type of images, geometry does not need to be the same for all frames, and can store not only images, but any other node types (images, transforms, markups, etc.). Therefore, it is generally recommended to use Sequences module instead.

Use cases

Most frequently used for these scenarios:

- Import multiple frames from files in a folder. Each frame must be stored as a separate NRRD, NIfTI, or any other image format supported by 3D Slicer.
- Import multiple frames from DICOM. This module registers a reader plugin in the DICOM module. The plugin automatically recognizes image sequences and loads them as a multivolume data set. In the application settings -> DICOM -> MultiVolumeImporterPlugin section, `Preferred multi-volume import format` setting specifies if DICOM image sequences will be read as `multi-volume` or a `volume sequence`. It is generally recommended to use `volume sequence`, as multi-volumes will be phased out.

Tutorials

Sample datasets are available:

- CTCardioMultiVolume.zip: ECG-gated contrast-enhanced cardiac CT, 10 frames, each saved as a nrrd file - *the file must be renamed after downloading*

Panels and their use

- **Basic settings**
  - **Input directory**: location of the input data as a collection of frames.
    - Frames can be in independent volume files or a single 4D NIfTI file with the selected directory.

> **Warning**
>
> The only files contained in the directory from which you are trying to import should be image volumes. The module will attempt to read each of these files.
>
> If you use non-DICOM input data type, the frames will be sorted based on the **alphanumerical order** of the frame filenames. If you have more than 10 frames, you should name them as follows to make sure they are ordered correctly, for example:
>
> - Correct naming: frame001.nrrd, frame002.nrrd, . . . , frame023.nrrd, . . . , frame912.nrrd
> - Incorrect naming: frame1.nrrd, frame2.nrrd, . . . , frame14.nrrd, . . . , frame1045.nrrrd.

  - **Output node**: MultiVolume node that will keep the loaded data. You need to create a new node or select and existing one when importing the data.
- **Advanced settings**: contains elements that can be changed by the user. These items will be associated with the resulting multivolume, and will be available in case they are needed for the subsequent post-processing of the data (e.g., for pharmacokinetic modeling)
  - **Frame identifying DICOM tag**: in all modes, shows the DICOM tag that will be used to separate individual frames/volumes in the DICOM series. This field does not have meaning when the input data type is non-DICOM.
  - **Frame identifying units**: automatically populated for pre-defined tags. Needs to be defined for other input data types.
  - **Initial value** and **Step**: specify values of the frame-identifying units for non-DICOM data type.
  - **Import button**: once the panels are populated with the appropriate settings, hit this button to import the dataset into Slicer. Note that depending on the size of the data this operation can take significant time, so be patient.

Related modules

- *MultiVolumeExplorer* module can be used for browsing the multi-volume data set after it is imported using MultiVolumeImporter module.
- PkModeling extension can be used for pharmacokinetic analysis of the DCE MRI data.
- *Sequences* is a more generic version of MultiVolumeImporter/MultiVolumeExplorer module. Eventually, multi-volume modules will be deprecated and removed from 3D Slicer and only Sequences module will remain.

Information for developers

Development of this module was initiated at the 2012 NA-MIC Project week in Salt Lake City, UT.

This module is an Slicer module stored in a separate repository, but bundled in the Slicer installation package. The source code is available on Github at https://github.com/fedorov/MultiVolumeImporter.

Contributors

- Andrey Fedorov (SPL, BWH)
- Jean-Cristophe Fillion Robin (Kitware)
- Julien Finet (Kitware)
- Steve Pieper (Isomics)
- Ron Kikinis (SPL, BWH)

Acknowledgements

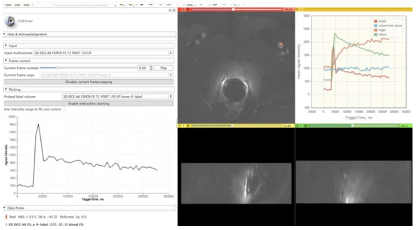
This work is supported by NA-MIC, NAC, NCIGT, and the Slicer Community. This work is partially supported by the following grants: P41EB015898, P41RR019703, R01CA111288 and U01CA151261.

9.18.4 MultiVolumeExplorer

Overview

This module allows browsing of frames of multi-volume data, such as a time sequence of images.

> **Note**
>
> This module is being phased out. It will be replaced by the more generic *Sequences* module, which can handle any type of images, geometry does not need to be the same for all frames, and can store not only images, but any other node types (images, transforms, markups, etc.). Therefore, it is generally recommended to use Sequences module instead.



Use cases

Most frequently used for these scenarios:

- Visualization of a DICOM dataset that contains multiple frames that can be separated based on some tag (e.g., DCE MRI data, where individual temporally resolved frames are identified by Trigger Time tag (0018,1060)
- Visualization of multiple frames defined in the same coordinate frame, saved as individual volumes in NRRD, NIfTI, or any other image format supported by 3D Slicer.
- Exploration of the multivolume data (cine mode visualization, plotting volume rendering).

Tutorials

- Exploration and Study of MultiVolume Image Data using 3D Slicer (Meysam Torabi and Andrey Fedorov)
- Video tutorial

Sample datasets are available:

- CTCardioMultiVolume.zip: ECG-gated contrast-enhanced cardiac CT, 10 frames, each saved as a nrrd file - *the file must be renamed after downloading*
- Prostate DCE-MRI series
  - Native DICOM format: DCE_series.zip - *the file must be renamed after downloading*
  - MultiVolume 4D NRRD format: DCE_series.nrrd - *the file must be renamed after downloading*

Panels and their use

> **Note**
>
> Before the module can be used, you should import the data into a MultiVolume node that you can choose as input in this module. There are two options to do this
>
> - If your data is in DICOM format: import it into Slicer DICOM database using DICOM module. Once imported, double-click on the series containing the multi-frame data in the DICOM browser. If DICOM module detects multi-frame dataset in the series (and default image sequence loading format is set to multi-volume) then the series will be loaded as a multi-volume.
> - If your data is in non-DICOM format (stored as a collection of NRRD/NIFTI/etc. volumes per time-point): import the images using *MultiVolumeImporter* module to first create a multi-volume node from your file collection, and then use that as input in MultiVolumeExplorer module.

- **Input multivolume**: select the multi-volume node you would like to explore.
- **Input secondary multivolume**: select an additional multi-volume node you would like to explore.
- Frame control:
  - **Current frame number**: you can use the slider or spin-box to select the currently shown frame.

    - **Play** button can be used to activate 'cine' view mode, with the frames being shown in a continuous mode.
    - **Current frame copy**: If **Enable copying while sliding** is enabled then each time the currently shown frame is changed, it will be copied to a scalar volume node.
    - **Current frame click-to-copy**: The current frame is copied into the selected volume node when the **Copy frame** button is clicked. This is useful in situations when you want to do processing of an individual frame (e.g., segmentation), or if you want to show volume rendering of a selected frame.
- Plotting:
    - Interactive plotting: When enabled, the chart will display the intensity values at the spatial location defined by the current cursor position, which can be set by moving the mouse cursor in a slice view, while holding down the `Shift` key. The range of the Y axis can be either fixed to the maximum intensity over all of the voxels/all frames of the dataset (if the fixed axis extent checkbox is selected), or otherwise will be adjusted dynamically to the signal range of the probed voxel curve.
    - Static plotting: This mode was developed for plotting intensity changes within designated regions. This mode is no longer available, but a new module will be added that provides this feature for volume sequences (using *Sequences module*).

Related modules

- *MultiVolumeImporter* module can be used for creating multi-volume data from separate volume files or load from a multi-frame DICOM series.
- PkModeling extension can be used for pharmacokinetic analysis of the DCE MRI data.
- *Sequences* is a more generic version of MultiVolumeImporter/MultiVolumeExplorer module. Eventually, multi-volume modules will be deprecated and removed from 3D Slicer and only Sequences module will remain.

Information for developers

Development of this module was initiated at the 2012 NA-MIC Project week in Salt Lake City, UT.

This module is an Slicer module stored in a separate repository, but bundled in the Slicer installation package. The source code is available on Github at https://github.com/fedorov/MultiVolumeExplorer.

Contributors

- Andrey Fedorov (SPL, BWH)
- Jean-Cristophe Fillion Robin (Kitware)
- Julien Finet (Kitware)
- Steve Pieper (Isomics)
- Ron Kikinis (SPL, BWH)

Acknowledgements

This work is supported by NA-MIC, NAC, NCIGT, and the Slicer Community. This work is partially supported by the following grants: P41EB015898, P41RR019703, R01CA111288 and U01CA151261.

9.19 Diffusion

9.19.1 Diffusion-weighted DICOM Import (DWIConvert)

Overview

Converts diffusion weighted MR images in DICOM series into NRRD format for analysis in Slicer. This program has been tested on only a limited subset of DTI DICOM formats available from Siemens, GE, and Philips scanners. Work in progress to support DICOM multi-frame data. The program parses DICOM header to extract necessary information about measurement frame, diffusion weighting directions, b-values, etc, and write out a NRRD image. For non-diffusion weighted DICOM images, it loads in an entire DICOM series and writes out a single dicom volume in a .nhdr/.raw pair.

Use cases

- Loading DICOM diffusion MRI data into Slicer.
- Conversion of diffusion weighted images (DWIs) from DICOM format to nrrd or nifti formats.

Tutorials

- Tutorials: https://dmri.slicer.org/docs/

Panels and their use

Conversion Options: Options that are used for all conversion modes

- **Input DWI Volume file** (*inputVolume*): Input DWI volume – not used for DicomToNrrd mode.
- **Output DWI Volume file** (*outputVolume*): Output filename (.nhdr or .nrrd)

Dicom To Nrrd Conversion Parameters: Parameters for Dicom to NRRD Conversion

- **Input Dicom Data Directory** (*inputDicomDirectory*): Directory holding Dicom series

NiftiFSL To Nrrd Conversion Parameters: NiftiFSL To Nrrd Conversion Parameters

- **FSL NIfTI File** (*fslNIFTIFile*): 4D NIfTI file containing gradient volumes
- **Input bval file** (*inputBValues*): The B Values are stored in FSL .bval text file format
- **Input bvec file** (*inputBVectors*): The Gradient Vectors are stored in FSL .bvec text file format

Nrrd To NiftiFSL Conversion Parameters: Nrrd To NiftiFSL (NrrdToFSL) Conversion Parameters

- **Output nii file** (*outputNiftiFile*): Nifti output filename (for Slicer GUI use).
- **Output bval file** (*outputBValues*): The B Values are stored in FSL .bval text file format (defaults to .bval)
- **Output bvec file** (*outputBVectors*): The Gradient Vectors are stored in FSL .bvec text file format (defaults to .bvec)

Advanced Conversion Parameters: Options to control the output.

- **Write Protocol Gradients File** (*writeProtocolGradientsFile*): Write the protocol gradients to a file suffixed by ".txt" as they were specified in the procol by multiplying each diffusion gradient direction by the measurement frame. This file is for debugging purposes only, the format is not fixed, and will likely change as debugging of new dicom formats is necessary.
- **Use Identity Measurement Frame** (*useIdentityMeaseurementFrame*): Adjust all the gradients so that the measurement frame is an identity matrix.
- **Use BMatrix Gradient Directions** (*useBMatrixGradientDirections*): Fill the nhdr header with the gradient directions and bvalues computed out of the BMatrix. Only changes behavior for Siemens data. In some cases the standard public gradients are not properly computed. The gradients can be empirically computed from the private BMatrix fields. In some cases the private BMatrix is consistent with the public grandients, but not in all cases, when it exists BMatrix is usually most robust.
- **Output Directory** (*outputDirectory*): Directory holding the output NRRD file
- **Small Gradient Threshold** (*smallGradientThreshold*): If a gradient magnitude is greater than 0 and less than smallGradientThreshold, then DWIConvert will display an error message and quit, unless the useBMatrixGradientDirections option is set.
- **Transpose Input BVectors** (*transpose*): FSL input BVectors are expected to be encoded in the input file as one vector per line. If it is not the case, use this option to transpose the file as it is read
- **Allow lossy image conversion** (*allowLossyConversion*): The only supported output type is 'short'. Conversion from images of a different type may cause data loss due to rounding or truncation. Use with caution!

DEPRECATED THESE DO NOT WORK:

- **Gradient Vector File** (*gradientVectorFile*): DEPRECATED: Use –inputBVector –inputBValue files Text file giving gradient vectors
- **Output fMRI file** (*fMRIOutput*): DEPRECATED: No support or testing. Output a NRRD file, but without gradients

Contributors

Hans Johnson (UIowa), Vince Magnotta (UIowa) Joy Matsui (UIowa), Kent Williams (UIowa), Mark Scully (Uiowa), Xiaodong Tao (GE)

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. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Similar modules

SlicerDMRI extension

Information for Developers

- DICOM for DWI and DTI
- Source code on github

9.19.2 DWI Cleanup (BRAINS)

Overview

Remove bad gradients/volumes from DWI NRRD file.

Panels and their use

Input Parameters:

- **Input Image Volume** (*inputVolume*): Required: input image is a 4D NRRD image.
- **NRRD File with bad gradients removed.** (*outputVolume*): given a list of
- **list of bad gradient volumes** (*badGradients*):

Contributors

This tool was developed by Kent Williams.

Acknowledgements

9.19.3 Resample DTI Volume

Overview

Resampling an image is a very important task in image analysis. It is especially important in the frame of image registration. This module implements DT image resampling through the use of itk Transforms. The resampling is controlled by the Output Spacing. "Resampling" is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions.

Panels and their use

Input/Output: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be resampled
- **Output Volume** (*outputVolume*): Resampled Volume
- **Reference Volume (To Set Output Parameters)** (*referenceVolume*): Reference Volume (spacing,size,orientation,origin)

Transform Parameters: Parameters used to transform the input image into the output image

- **Transform Node** (*transformationFile*):
- **Deformation Field Volume** (*deffield*): File containing the deformation field (3D vector image containing vectors with 3 components)
- **Displacement or h-Field** (*typeOfField*): Set if the deformation field is an -Field

Processing Options:

- **Interpolation** (*interpolationType*): Sampling algorithm (linear , nn (nearest neighbor), ws (WindowedSinc), bs (BSpline))
- **No Measurement Frame** (*noMeasurementFrame*): Do not use the measurement frame that is in the input image to transform the tensors. Uses the image orientation instead
- **Tensors Correction** (*correction*): Correct the tensors if computed tensor is not semi-definite positive

Tensor Transform Type:

- **Finite Strain (FS) or \nPreservation of the Principal Direction (PPD)** (*ppd*): Chooses between 2 methods to transform the tensors: Finite Strain (FS), faster but less accurate, or Preservation of the Principal Direction (PPD)

Advanced Transform Parameters: Those parameters should normally not be modified

- **Transforms Order** (*transformsOrder*): Select in what order the transforms are read
- **Not a Bulk Transform** (*notbulk*): The transform following the BSpline transform is not set as a bulk transform for the BSpline transform
- **Space Orientation inconsistency (between transform and image)** (*space*): Space Orientation between transform and image is different (RAS/LPS) (warning: if the transform is a Transform Node in Slicer3, do not select)

Advanced Rigid/Affine Parameters:

- **Rotation Center** (*rotationPoint*): Center of rotation (only for rigid and affine transforms)
- **Centered Transform** (*centeredTransform*): Set the center of the transformation to the center of the input image (only for rigid and affine transforms)
- **Image Center** (*imageCenter*): Image to use to center the transform (used only if "Centered Transform" is selected)
- **Inverse Transformation** (*inverseITKTransformation*): Inverse the transformation before applying it from output image to input image (only for rigid and affine transforms)

Manual Output Parameters: Parameters of the output image

- **Spacing** (*outputImageSpacing*): Spacing along each dimension (0 means use input spacing)
- **Size** (*outputImageSize*): Size along each dimension (0 means use input size)
- **Origin** (*outputImageOrigin*): Origin of the output Image
- **Direction Matrix** (*directionMatrix*): 9 parameters of the direction matrix by rows (ijk to LPS if LPS transform, ijk to RAS if RAS transform)

Advanced Resampling Parameters: Parameters used for resampling

- **Number Of Threads** (*numberOfThread*): Number of thread used to compute the output image
- **Default Pixel Value** (*defaultPixelValue*): Default pixel value for samples falling outside of the input region

Windowed Sinc Interpolate Function Parameters: Parameters used for the Windowed Sinc interpolation

- **Window Function** (*windowFunction*): Window Function \nh = Hamming \nc = Cosine \nw = Welch \nl = Lanczos \nb = Blackman

BSpline Interpolate Function Parameters: Parameters used for the BSpline interpolation

- **Spline Order** (*splineOrder*): Spline Order (Spline order may be from 0 to 5)

Manual Transform (Used only if no transform node set):

- **Transform Matrix** (*transformMatrix*): 12 parameters of the transform matrix by rows (–last 3 being translation–)
- **Transform Type** (*transformType*): Transform algorithm\nrt = Rigid Transform\na = Affine Transform

Contributors

Francois Budin (UNC)

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. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics

9.19.4 Resample Scalar/Vector/DWI Volume

Overview

This module implements image and vector-image resampling through the use of itk Transforms. It can also handle diffusion weighted MRI image resampling. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions. \n\nWarning: To resample DWMR Images, use nrrd input and output files. \n\nWarning: Do not use to resample Diffusion Tensor Images, tensors would not be reoriented

Panels and their use

Input/Output: Input/output parameters

- **Input Volume** (*inputVolume*): Input Volume to be resampled
- **Output Volume** (*outputVolume*): Resampled Volume
- **Reference Volume (To Set Output Parameters)** (*referenceVolume*): Reference Volume (spacing,size,orientation,origin)

Transform Parameters: Parameters used to transform the input image into the output image

- **Transform Node** (*transformationFile*):
- **Deformation Field Volume** (*deffield*): File containing the deformation field (3D vector image containing vectors with 3 components)
- **Displacement or H-Field** (*typeOfField*): Set if the deformation field is an h-Field

Interpolation Type:

- **Interpolation** (*interpolationType*): Sampling algorithm (linear or nn (nearest neighbor), ws (WindowedSinc), bs (BSpline))

Advanced Transform Parameters: Those parameters should normally not be modified

- **Transforms Order** (*transformsOrder*): Select in what order the transforms are read
- **Not a Bulk Transform** (*notbulk*): The transform following the BSpline transform is not set as a bulk transform for the BSpline transform
- **Space Orientation inconsistency (between transform and image)** (*space*): Space Orientation between transform and image is different (RAS/LPS) (warning: if the transform is a Transform Node in Slicer3, do not select)

Rigid/Affine Parameters:

- **Rotation Point** (*rotationPoint*): Rotation Point in case of rotation around a point (otherwise useless)
- **Centered Transform** (*centeredTransform*): Set the center of the transformation to the center of the input image
- **Image Center** (*imageCenter*): Image to use to center the transform (used only if "Centered Transform" is selected)
- **Inverse ITK Transformation** (*inverseITKTransformation*): Inverse the transformation before applying it from output image to input image

Manual Output Parameters: Parameters of the output image

- **Spacing** (*outputImageSpacing*): Spacing along each dimension (0 means use input spacing)
- **Size** (*outputImageSize*): Size along each dimension (0 means use input size)
- **Origin** (*outputImageOrigin*): Origin of the output Image
- **Direction Matrix** (*directionMatrix*): 9 parameters of the direction matrix by rows (ijk to LPS if LPS transform, ijk to RAS if RAS transform)

Advanced Resampling Parameters: Parameters used for resampling

- **Number Of Thread** (*numberOfThread*): Number of thread used to compute the output image
- **Default Pixel Value** (*defaultPixelValue*): Default pixel value for samples falling outside of the input region

Windowed Sinc Interpolate Function Parameters: Parameters used for the Windowed Sinc interpolation

- **Window Function** (*windowFunction*): Window Function \nh = Hamming \nc = Cosine \nw = Welch \nl = Lanczos \nb = Blackman

BSpline Interpolate Function Parameters: Parameters used for the BSpline interpolation

- **Spline Order** (*splineOrder*): Spline Order

Manual Transform (Only used if no transform node set):

- **Transform Matrix** (*transformMatrix*): 12 parameters of the transform matrix by rows (–last 3 being translation–)
- **Transform** (*transformType*): Transform algorithm\nrt = Rigid Transform\na = Affine Transform

Contributors

Francois Budin (UNC)

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. Information on the National Centers for Biomedical Computing can be obtained from http://nihroadmap.nih.gov/bioinformatics

9.20 Filtering

9.20.1 Add Scalar Volumes

Overview

Adds two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

Panels and their use

IO: Input/output parameters

- **Input Volume 1** (*inputVolume1*): Input volume 1
- **Input Volume 2** (*inputVolume2*): Input volume 2
- **Output Volume** (*outputVolume*): Volume1 + Volume2

Advanced: Advanced parameters for fine-tune the computation.

- **Interpolation order** (*order*): Order of the polynomial interpolation that is used if two images have different geometry (origin, spacing, axis directions, or extents): 0 = nearest neighbor, 1 = linear, 2 = quadratic, 3 = cubic interpolation.

Contributors

Bill Lorensen (GE)

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.

9.20.2 Cast Scalar Volume

Overview

Cast a volume to a given data type.\nUse at your own risk when casting an input volume into a lower precision type!\nAllows casting to the same type as the input volume.

Panels and their use

IO: Input/output parameters

- **Input Volume** (*InputVolume*): Input volume, the volume to cast.
- **Output Volume** (*OutputVolume*): Output volume, cast to the new type.

Filter Settings:

- **Output Type** (*Type*): Scalar data type for the new output volume.

Contributors

Nicole Aucoin (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.

9.20.3 Curvature Anisotropic Diffusion

Overview

Performs anisotropic diffusion on an image using a modified curvature diffusion equation (MCDE).\n\nMCDE does not exhibit the edge enhancing properties of classic anisotropic diffusion, which can under certain conditions undergo a 'negative' diffusion, which enhances the contrast of edges. Equations of the form of MCDE always undergo positive diffusion, with the conductance term only varying the strength of that diffusion. \n\n Qualitatively, MCDE compares well with other non-linear diffusion techniques. It is less sensitive to contrast than classic Perona-Malik style diffusion, and preserves finer detailed structures in images. There is a potential speed trade-off for using this function in place of Gradient Anisotropic Diffusion. Each iteration of the solution takes roughly twice as long. Fewer iterations, however, may be required to reach an acceptable solution.

Panels and their use

Anisotropic Diffusion Parameters: Parameters for the anisotropic diffusion algorithm

- **Conductance** (*conductance*): Conductance controls the sensitivity of the conductance term. As a general rule, the lower the value, the more strongly the filter preserves edges. A high value will cause diffusion (smoothing) across edges. Note that the number of iterations controls how much smoothing is done within regions bounded by edges.
- **Iterations** (*numberOfIterations*): The more iterations, the more smoothing. Each iteration takes the same amount of time. If it takes 10 seconds for one iteration, then it will take 100 seconds for 10 iterations. Note that the conductance controls how much each iteration smooths across edges.
- **Time Step** (*timeStep*): The time step depends on the dimensionality of the image. In Slicer the images are 3D and the default (.0625) time step will provide a stable solution.

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Bill Lorensen (GE)

Acknowledgements

This command module was derived from Insight/Examples (copyright) Insight Software Consortium

9.20.4 Gaussian Blur Image Filter

Overview

Apply a gaussian blur to an image

Panels and their use

IO: Input/output parameters

- **Sigma** (*sigma*): Sigma value in physical units (e.g., mm) of the Gaussian kernel
- **Input Volume** (*inputVolume*): Input volume
- **Output Volume** (*outputVolume*): Blurred Volume

Contributors

Julien Jomier (Kitware), Stephen Aylward (Kitware)

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.

9.20.5 Gradient Anisotropic Diffusion

Overview

Runs gradient anisotropic diffusion on a volume.\n\nAnisotropic diffusion methods reduce noise (or unwanted detail) in images while preserving specific image features, like edges. For many applications, there is an assumption that light-dark transitions (edges) are interesting. Standard isotropic diffusion methods move and blur light-dark boundaries. Anisotropic diffusion methods are formulated to specifically preserve edges. The conductance term for this implementation is a function of the gradient magnitude of the image at each point, reducing the strength of diffusion at edges. The numerical implementation of this equation is similar to that described in the Perona-Malik paper, but uses a more robust technique for gradient magnitude estimation and has been generalized to N-dimensions.

Panels and their use

Anisotropic Diffusion Parameters: Parameters for the anisotropic diffusion algorithm

- **Conductance** (*conductance*): Conductance controls the sensitivity of the conductance term. As a general rule, the lower the value, the more strongly the filter preserves edges. A high value will cause diffusion (smoothing) across edges. Note that the number of iterations controls how much smoothing is done within regions bounded by edges.
- **Iterations** (*numberOfIterations*): The more iterations, the more smoothing. Each iteration takes the same amount of time. If it takes 10 seconds for one iteration, then it will take 100 seconds for 10 iterations. Note that the conductance controls how much each iteration smooths across edges.

- **Time Step** (*timeStep*): The time step depends on the dimensionality of the image. In Slicer the images are 3D and the default (.0625) time step will provide a stable solution.

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Advanced: Advanced parameters for the anisotropic diffusion algorithm

- **Use image spacing** (*useImageSpacing*): ![CDATA[Take into account image spacing in the computation. It is advisable to turn this option on, especially when the pixel size is different in different dimensions. However, to produce results consistent with Slicer4.2 and earlier, this option should be turned off.]]

Contributors

Bill Lorensen (GE)

Acknowledgements

This command module was derived from Insight/Examples (copyright) Insight Software Consortium

9.20.6 Grayscale Fill Hole Image Filter

Overview

GrayscaleFillholeImageFilter fills holes in a grayscale image. Holes are local minima in the grayscale topography that are not connected to boundaries of the image. Gray level values adjacent to a hole are extrapolated across the hole.\n\nThis filter is used to smooth over local minima without affecting the values of local maxima. If you take the difference between the output of this filter and the original image (and perhaps threshold the difference above a small value), you'll obtain a map of the local minima.\n\nThis filter uses the itkGrayscaleGeodesicErodeImageFilter. It provides its own input as the "mask" input to the geodesic erosion. The "marker" image for the geodesic erosion is constructed such that boundary pixels match the boundary pixels of the input image and the interior pixels are set to the maximum pixel value in the input image.\n\n Geodesic morphology and the Fillhole algorithm is described in Chapter 6 of Pierre Soille's book "Morphological Image Analysis: Principles and Applications", Second Edition, Springer, 2003. \n\n A companion filter, Grayscale Grind Peak, removes peaks in grayscale images.

Panels and their use

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Bill Lorensen (GE)

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.

9.20.7 Grayscale Grind Peak Image Filter

Overview

GrayscaleGrindPeakImageFilter removes peaks in a grayscale image. Peaks are local maxima in the grayscale topography that are not connected to boundaries of the image. Gray level values adjacent to a peak are extrapolated through the peak.\n\nThis filter is used to smooth over local maxima without affecting the values of local minima. If you take the difference between the output of this filter and the original image (and perhaps threshold the difference above a small value), you'll obtain a map of the local maxima.\n\nThis filter uses the GrayscaleGeodesicDilateImageFilter. It provides its own input as the "mask" input to the geodesic erosion. The "marker" image for the geodesic erosion is constructed such that boundary pixels match the boundary pixels of the input image and the interior pixels are set to the minimum pixel value in the input image.\n\nThis filter is the dual to the GrayscaleFillholeImageFilter which implements the Fillhole algorithm. Since it is a dual, it is somewhat superfluous but is provided as a convenience.\n\nGeodesic morphology and the Fillhole algorithm is described in Chapter 6 of Pierre Soille's book "Morphological Image Analysis: Principles and Applications", Second Edition, Springer, 2003.\n\nA companion filter, Grayscale Fill Hole, fills holes in grayscale images.

Panels and their use

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Bill Lorensen (GE)

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.

9.20.8 Mask Scalar Volume

Overview

Masks two images. The output image is set to 0 everywhere except where the chosen label from the mask volume is present, at which point it will retain it's original values. The two images do not have to have the same dimensions.

Panels and their use

Input and Output: Input/output parameters

- **Input Volume** (*InputVolume*): Input volume to be masked
- **Mask Volume** (*MaskVolume*): Label volume containing the mask
- **Masked Volume** (*OutputVolume*): Output volume: Input Volume masked by label value from Mask Volume

Settings: Filter settings

- **Label value** (*Label*): Label value in the Mask Volume to use as the mask
- **Replace value** (*Replace*): Value to use for the output volume outside of the mask

Contributors

Nicole Aucoin (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.

9.20.9 Median Image Filter

Overview

The MedianImageFilter is commonly used as a robust approach for noise reduction. This filter is particularly efficient against "salt-and-pepper" noise. In other words, it is robust to the presence of gray-level outliers. MedianImageFilter computes the value of each output pixel as the statistical median of the neighborhood of values around the corresponding input pixel.

Panels and their use

Median Filter Parameters: Parameters for the median filter

- **Neighborhood Size** (*neighborhood*): The size of the neighborhood in each dimension

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Bill Lorensen (GE)

Acknowledgements

This command module was derived from Insight/Examples/Filtering/MedianImageFilter (copyright) Insight Software Consortium

9.20.10 Multiply Scalar Volumes

Overview

Multiplies two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

Panels and their use

IO: Input/output parameters

- **Input Volume 1** (*inputVolume1*): Input volume 1
- **Input Volume 2** (*inputVolume2*): Input volume 2
- **Output Volume** (*outputVolume*): Volume1 * Volume2

Controls: Control how the module operates

- **Interpolation order** (*order*): Interpolation order if two images are in different coordinate frames or have different sampling.

Contributors

Bill Lorensen (GE)

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.

9.20.11 N4ITK MRI Bias correction

Overview

Performs image bias correction using N4 algorithm. This module is based on the ITK filters contributed in the following publication: Tustison N, Gee J "N4ITK: Nick's N3 ITK Implementation For MRI Bias Field Correction", The Insight Journal 2009 January-June, https://hdl.handle.net/10380/3053

Panels and their use

IO: Input/output parameters

- **Input Image** (*inputImageName*): Input image where you observe signal inhomegeneity
- **Mask Image** (*maskImageName*): Binary mask that defines the structure of your interest. NOTE: This parameter is OPTIONAL. If the mask is not specified, the module will use internally Otsu thresholding to define this mask. Better processing results can often be obtained when a meaningful mask is defined.
- **Output Volume** (*outputImageName*): Result of processing
- **Output bias field image** (*outputBiasFieldName*): Recovered bias field (OPTIONAL)

N4 Parameters:

- **BSpline grid resolution** (*initialMeshResolution*): Resolution of the initial bspline grid defined as a sequence of three numbers. The actual resolution will be defined by adding the bspline order (default is 3) to the resolution in each dimension specified here. For example, 1,1,1 will result in a 4x4x4 grid of control points. This parameter may need to be adjusted based on your input image. In the multi-resolution N4 framework, the resolution of the bspline grid at subsequent iterations will be doubled. The number of resolutions is implicitly defined by Number of iterations parameter (the size of this list is the number of resolutions)
- **Spline distance** (*splineDistance*): An alternative means to define the spline grid, by setting the distance between the control points. This parameter is used only if the grid resolution is not specified.
- **Bias field Full Width at Half Maximum** (*bfFWHM*): Bias field Full Width at Half Maximum. Zero implies use of the default value.

Advanced N4 Parameters: Advanced parameters of the algorithm

- **Number of iterations** (*numberOfIterations*): Maximum number of iterations at each level of resolution. Larger values will increase execution time, but may lead to better results.
- **Convergence threshold** (*convergenceThreshold*): Stopping criterion for the iterative bias estimation. Larger values will lead to smaller execution time.

- **BSpline order** (*bsplineOrder*): Order of B-spline used in the approximation. Larger values will lead to longer execution times, may result in overfitting and poor result.
- **Shrink factor** (*shrinkFactor*): Defines how much the image should be upsampled before estimating the inhomogeneity field. Increase if you want to reduce the execution time. 1 corresponds to the original resolution. Larger values will significantly reduce the computation time.
- **Weight Image** (*weightImageName*): Weight Image
- **Wiener filter noise** (*wienerFilterNoise*): Wiener filter noise. Zero implies use of the default value.
- **Number of histogram bins** (*nHistogramBins*): Number of histogram bins. Zero implies use of the default value.

Contributors

Nick Tustison (UPenn), Andrey Fedorov (SPL, BWH), Ron Kikinis (SPL, BWH)

Acknowledgements

The development of this module was partially supported by NIH grants R01 AA016748-01, R01 CA111288 and U01 CA151261 as well as by NA-MIC, NAC, NCIGT and the Slicer community.

9.20.12 CheckerBoard Filter

Overview

Create a checkerboard volume of two volumes. The output volume will show the two inputs alternating according to the user supplied checkerPattern. This filter is often used to compare the results of image registration. Note that the second input is resampled to the same origin, spacing and direction before it is composed with the first input. The scalar type of the output volume will be the same as the input image scalar type.

Panels and their use

CheckerBoard Parameters: Parameters for the checkerboard

- **Checker Pattern** (*checkerPattern*): The pattern of input 1 and input 2 in the output image. The user can specify the number of checkers in each dimension. A checkerPattern of 2,2,1 means that images will alternate in every other checker in the first two dimensions. The same pattern will be used in the 3rd dimension.

IO: Input/output parameters

- **Input Volume 1** (*inputVolume1*): First Input volume
- **Input Volume 2** (*inputVolume2*): Second Input volume
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Bill Lorensen (GE)

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.

9.20.13 Extract Skeleton

Overview

Extract the skeleton of a binary object. The skeleton can be limited to being a 1D curve or allowed to be a full 2D manifold. The branches of the skeleton can be pruned so that only the maximal center skeleton is returned.

Panels and their use

IO: Input/output parameters

- **Input Image** (*InputImageFileName*): Input image
- **Output Image** (*OutputImageFileName*): Skeleton of the input image.

Skeleton: Skeleton parameters

- **Skeleton type** (*SkeletonType*): Type of skeleton to create. 1D extract centerline curve points. 2D extracts medial surface points.
- **Extract full tree** (*FullTree*): Include the full tree in the output, not just the longest branch.
- **Number Of Points** (*NumberOfPoints*): Number of points used to represent the skeleton
- **Output points list** (*OutputPointsFileName*): Name of the file to store the coordinates of the central (1D) skeleton points
- **Output markups curve** (*OutputCurveFileName*): Centerline points as markups curve

Contributors

Pierre Seroul (UNC), Martin Styner (UNC), Guido Gerig (UNC), Stephen Aylward (Kitware)

Acknowledgements

The original implementation of this method was provided by ETH Zurich, Image Analysis Laboratory of Profs Olaf Kuebler, Gabor Szekely and Guido Gerig. Martin Styner at UNC, Chapel Hill made enhancements. Wrapping for Slicer was provided by Pierre Seroul and Stephen Aylward at Kitware, Inc.

9.20.14 Histogram Matching

Overview

Normalizes the grayscale values of a source image based on the grayscale values of a reference image. This filter uses a histogram matching technique where the histograms of the two images are matched only at a specified number of quantile values.\n\nThe filter was originally designed to normalize MR images of the sameMR protocol and same body part. The algorithm works best if background pixels are excluded from both the source and reference histograms. A simple background exclusion method is to exclude all pixels whose grayscale values are smaller than the mean grayscale value. ThresholdAtMeanIntensity switches on this simple background exclusion method.\n\n Number of match points governs the number of quantile values to be matched.\n\nThe filter assumes that both the source and reference are of the same type and that the input and output image type have the same number of dimension and have scalar pixel types.

Panels and their use

Histogram Matching Parameters: Parameters for Histogram Matching

- **Number of Histogram Levels** (*numberOfHistogramLevels*): The number of hisogram levels to use
- **Number of Match Points** (*numberOfMatchPoints*): The number of match points to use

- **Threshold at mean** (*thresholdAtMeanIntensity*): If on, only pixels above the mean in each volume are thresholded.

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Reference Volume** (*referenceVolume*): Input volume whose histogram will be matched
- **Output Volume** (*outputVolume*): Output volume. This is the input volume with intensities matched to the reference volume.

Contributors

Bill Lorensen (GE)

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.

9.20.15 Image Label Combine

Overview

Combine two label maps into one

Panels and their use

IO: Input/output parameters

- **Input Label Map A** (*InputLabelMap_A*): Label map image
- **Input Label Map B** (*InputLabelMap_B*): Label map image
- **Output Label Map** (*OutputLabelMap*): Resulting Label map image

Label combination options: Selection of how to combine label maps

- **First Label Overwrites Second** (*FirstOverwrites*): Use first or second label when both are present

Contributors

Alex Yarmarkovich (SPL, BWH)

Acknowledgements

9.20.16 Simple Filters

Overview

The SimpleFilters module provides a simple interface to hundreds of basic and advanced filters from ITK.

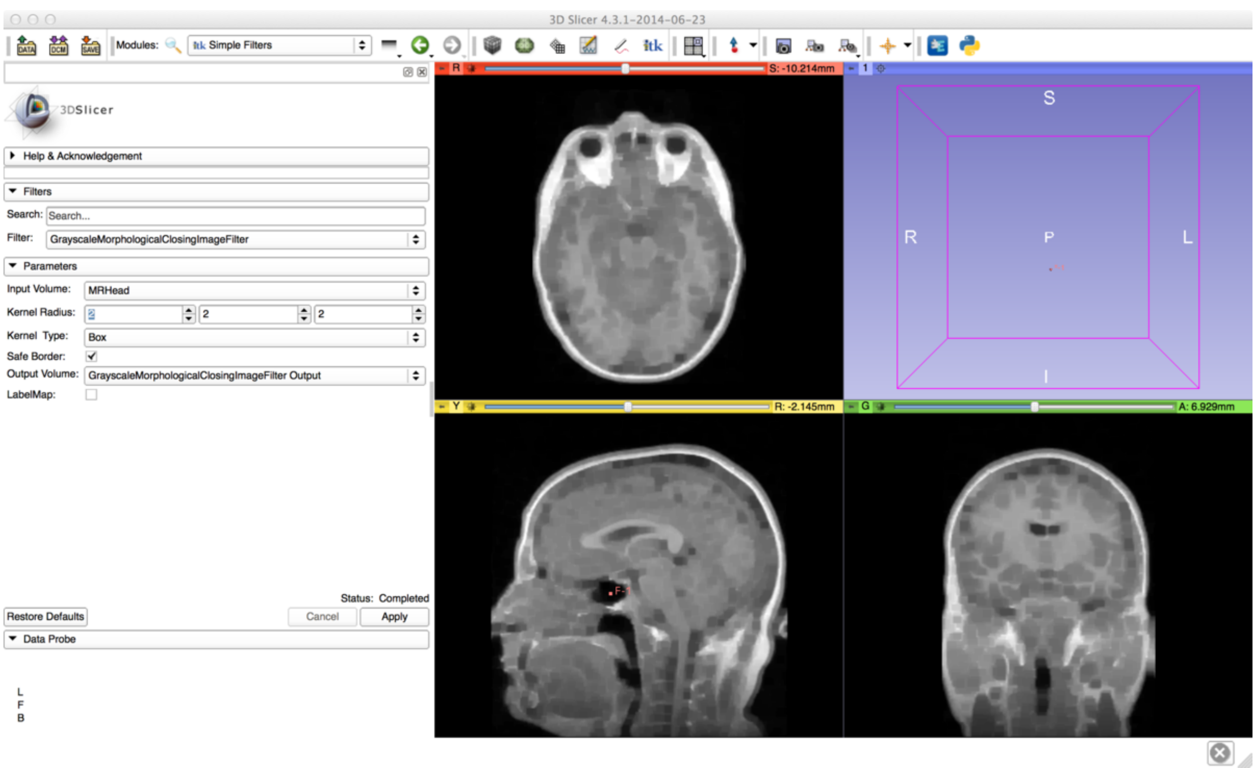
The algorithms available include binary morphology, grayscale morphology, denoising, thresholding, image intensity manipulation, region growing, FFT, and many advanced algorithms.

Panels and their use

- **Filters** section at the top enables the selection of one of the hundred of filters available. The **Search** text box is use quickly find a filter based on it's name.
- **Parameters** section dynamically changes based of the Filter selected above, it presents a list of input filter and parameters which the filter needs. Along with the output image for the filter.
- **Apply** button at the bottom runs the filter.
- **Cancel** button can be used to stop a running filter.
- **Restore Defaults** button revert the filter parameters to their initial settings.
- **LabelMap** checkbox indicates if the selected output volume is a labelmap. This can be used for verifying if the selected output volume is appropriate for storing the filtering result.

> **Note**
>
> Most filters which take more than one image as input expect that all the inputs are of the same pixel type, and the images occupy the same physical space.

> **Tip**
>
> Many filters only work on `float` pixel type. When an `integer` image is used as input, the `Exception thrown. .. Pixel type: ... integer is not supported...` error message is displayed. To use such filters, the pixel type can be converted to `float` using *Cast Scalar Volume* module.



Filters

Information for developers

Source code of the module is available at https://github.com/SimpleITK/SlicerSimpleFilters.

Contributors

- Bradley Lowekamp
- Steve Pieper
- Jean-Cristophe Fillion Robin

Acknowledgements

This work is supported by NLM, and the Slicer Community.

9.20.17 Subtract Scalar Volumes

Overview

Subtracts two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

Panels and their use

IO: Input/output parameters

- **Input Volume 1** (*inputVolume1*): Input volume 1
- **Input Volume 2** (*inputVolume2*): Input volume 2
- **Output Volume** (*outputVolume*): Volume1 - Volume2

Controls: Control how the module operates

- **Interpolation order** (*order*): Interpolation order if two images are in different coordinate frames or have different sampling.

Contributors

Bill Lorensen (GE)

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.

9.20.18 Threshold Scalar Volume

Overview

Panels and their use

IO: Input/output parameters

- **Input Volume** (*InputVolume*): Input volume
- **Output Volume** (*OutputVolume*): Thresholded input volume

Filter Settings:

- **Threshold Type** (*ThresholdType*): What kind of threshold to perform. If Outside is selected, uses Upper and Lower values. If Below is selected, uses the ThresholdValue, if Above is selected, uses the ThresholdValue.
- **Threshold Value** (*ThresholdValue*): Threshold value
- **Lower** (*Lower*): Lower threshold value
- **Upper** (*Upper*): Upper threshold value
- **Outside Value** (*OutsideValue*): Set the voxels to this value if they fall outside the threshold range
- **Negate Threshold** (*Negate*): Swap the outside value with the inside value.

Contributors

Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Julien Finet (Kitware)

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.

9.20.19 Voting Binary Hole Filling Image Filter

Overview

Applies a voting operation in order to fill-in cavities. This can be used for smoothing contours and for filling holes in binary images. This technique is used frequently when segmenting complete organs that may have ducts or vasculature that may not have been included in the initial segmentation, e.g. lungs, kidneys, liver.

Panels and their use

Binary Hole Filling Parameters: Parameters for Hole Filling

- **Maximum Radius** (*radius*): The radius of a hole to be filled
- **Majority Threshold** (*majorityThreshold*): The number of pixels over 50% that will decide whether an OFF pixel will become ON or not. For example, if the neighborhood of a pixel has 124 pixels (excluding itself), the 50% will be 62, and if you set a Majority threshold of 5, that means that the filter will require 67 or more neighbor pixels to be ON in order to switch the current OFF pixel to ON.
- **Background** (*background*): The value associated with the background (not object)
- **Foreground** (*foreground*): The value associated with the foreground (object)

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Bill Lorensen (GE)

Acknowledgements

This command module was derived from Insight/Examples/Filtering/VotingBinaryHoleFillingImageFilter (copyright) Insight Software Consortium

9.21 Utilities

9.21.1 Brain Deface from T1/T2 image (BRAINS)

Overview

This program: 1) will deface images from a set of images. Inputs must be ACPC aligned, and AC, PC, LE, RE provided.

Panels and their use

Input Images: First Image, Second Image and Mask Image

- **Landmarks File** (*inputLandmarks*): Input Landmark File with LE, and RE points defined in physical locations
- **Source Reference Image** (*inputVolume*): Input images, all images must be in the exact same physical space, ACPC aligned and consistent with landmarks.
- **Source Passive Images Image** (*passiveVolume*): Input images not used in generating masks, all images must be in the exact same physical space as inputVolumes, ACPC aligned and consistent with landmarks.
- **Optional Mask** (*inputMask*): Optional pre-generated mask to use.

Output Files: Outputs from both MUSH generation and brain volume mask creation

- **brain volume mask** (*outputMask*): The brain volume mask generated from the MUSH image
- **OutputDirectory** (*outputDirectory*): The output directory to writing the defaced input files

Run Mode: Modify the program to only generate a mask

- **No Mask Application** (*noMaskApplication*): Do not apply the mask to the input images used to generate the mask

Output Image Intensity Normalization: Parameters for normalizing the output images.

- **Upper Percentile** (*upperPercentile*): Upper Intensity Percentile (0.99 default)
- **Lower Percentile** (*lowerPercentile*): Lower Intensity Percentile (0.01 default)
- **Upper Output Intensity** (*upperOutputIntensity*): Upper Output Intensity
- **Lower Output Intensity** (*lowerOutputIntensity*): Lower Output Intensity
- **Upper Output Intensity** (*no_clip*): Do not clip Values outside of this range to be the "Outside Value"

- **Relative Scaling** (*no_relative*): Do not scale to the relative percentiles of the output scale
- **Debug Level** (*debugLevel*): Level of Debugging (0=None)

Contributors

This tool is created by Hans J. Johnson.

Acknowledgements

This work was developed by the University of Iowa Department of Electrical and Computer Engineering.

9.21.2 Strip Rotation (BRAINS)

Overview

Read an Image, write out same image with identity rotation matrix plus an ITK transform file

Panels and their use

Input Parameters:

- **Image To Warp** (*inputVolume*): Image To Warp
- **Output Image** (*outputVolume*): Resulting deformed image
- **Transform file** (*transform*): Filename for the transform file

Contributors

Kent Williams

Acknowledgements

9.21.3 Transform Convert (BRAINS)

Overview

Convert ITK transforms to higher order transforms

Panels and their use

IO: Input/output parameters

- **Transform File Name To Convert** (*inputTransform*):
- **Reference image** (*referenceVolume*):
- **Output displacement field** (*displacementVolume*):
- **Transform File Name To Save ConvertedTransform** (*outputTransform*):

Contributors

Hans J. Johnson,Kent Williams, Ali Ghayoor

Acknowledgements

9.21.4 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.

9.21.5 Endoscopy

Overview

Provides interactive animation of flythrough paths.

Basic use

Select the `Dual 3D` view in 3D Slicer so that you will be able to witness the flythrough from both first-person and third-person viewpoints, as well as in the displayed 2d-slice panes. Use the `Markups` module or toolbar to create a `curve` or `closed curve` in 3D space. You are now ready to use the `Endoscopy` module.

Panels and their use

Path

Selects the camera and curve to be manipulated.

- **Camera**: The camera used for the flythrough.
- **Curve to modify**: The curve defining the flythrough path control points.

Flythrough

Controls the animation.

- **Play flythrough / Stop flythrough**: Start or stop the flythrough animation.
- **Frame**: The current frame (step along the path).
- **Frame Skip**: Number of frames to skip (larger numbers make the animation go faster).
- **Frame Delay**: Time delay between animation frames (larger numbers make the animation go slower)
- **View Angle**: Field of view of the camera. The default value of 30 degrees approximates normal camera lenses. Larger numbers, such as 110 or 120 degrees approximate the wide angle lenses often used in endoscopy viewing systems.

- **Save keyframe orientation / Update keyframe orientation**: Press to indicate that a flythrough frame is a keyframe and that you have selected your desired camera orientation for this keyframe in the first-person, 3D viewing pane. If you wish to update the orientation for a keyframe, use the Frame, First, Back, Next, or Last buttons to go to the frame, adjust the camera orientation, and hit this button again.
- **Delete keyframe orientation**: Discard the camera orientation associated with the selected keyframe.
- **First**: Press to go to the lowest-numbered keyframe.
- **Back**: Press to move backwards through the flythrough to the nearest keyframe.
- **Next**: Press to move forwards through the flythrough to the nearest keyframe.
- **Last**: Press to go to the highest-numbered keyframe

By default, both the flythrough path and the cursor that indicates the current position are not shown in the first-person 3d-viewing pane, but are shown in the other panes. You can turn the visibility of the path or cursor on or off in any of these panes using the `Data` module.

Advanced

This functionality is retained to support older workflow requiring a model. It is expected that current users will not use the features in this section. Instead, a curve that is modified as above and then saved will retain its keyframe information and can be loaded and used at a later time.

- **Output model**: Select a name for the model to be exported
- **Export as model**: The flythrough will be exported as a model.

Contributors

Authors:

- Steve Pieper (Isomics)
- Jean-Christophe Fillion-Robin (Kitware)
- Harald Scheirich (Kitware)
- Lee Newberg (Kitware).

Acknowledgements

This work is supported by PAR-07-249: R01CA131718 NA-MIC Virtual Colonoscopy (See https://www.na-mic.org/Wiki/index.php/NA-MIC_NCBC_Collaboration:NA-MIC_virtual_colonoscopy) NA-MIC, NAC, BIRN, NCIGT, and the Slicer Community. See https://www.slicer.org for details.

9.21.6 Screen Capture

Overview

This module is for creating videos, image sequences, or lightbox image from 3D and slice view contents.

Panels and their use

Input

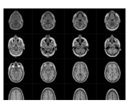
- **Main view:** This view is used for capturing (rotated, sweeped, etc.).
- **Capture all views:** When enabled, all views in the layout will be captured. Otherwise, only the main view is captured. By capturing all views, it is possible to see the animated view (such as a moving slice) in 3D and in other slice views.

- **Capture mode:** specifies how the main view will be modified during capture.
  - **single frame:** Acquire a single frame of selected main view or all views if "Capture all views" is enabled.
  - **3D rotation:** Acquire video of a rotating 3D view. For smooth repeated display of a 360-degree rotation it is recommended to choose 31 or 61 as "Number of images".
  - **slice sweep:** Acquire video while going through selected range of image frames (for slice viewer only).
  - **slice fade:** Acquire video while fading between the foreground and background image (for slice viewer only).).
  - **sequence:** sequence: Acquire video while going through items in the selected sequence browser.

3D rotation	Slice sweep	Slice fade	Sequence

Output

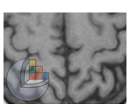
- **Output type:**
  - **image series:** Save screenshots as separate image files (in jpg or png file format).
  - **video:** Save animation as a compressed video file. Requires installation of *ffmpeg video encoder*.
  - **lightbox image:** Save screenshots as separate jpg or png files.

- **Number of images:** Defines how many frames are generated in the specified range. Higher number results in smoother animation but larger video file. If **single frame** capture mode is selected then a single image is captured (and counter in the filename is automatically incremented, therefore it can be used to acquire many screenshots manually).
- **Output directory:** Output image or video will be saved in this directory.
- **Output file name:** Output file name for video and lightbox.
- **Video format:**
  - **H264:** modern compressed video format, compatible with current video players.
  - **H264 (high-quality):** H264 with higher-quality setting, results in larger file.
  - **MPEG4:** commonly used compressed video format, mostly compatible with older video players.
  - **MPEG4 (high-quality):** MPEG4 with higher-quality setting, results in larger file.
  - **Animated GIF:** file format that provides lower quality images and large files, but it is more compatible with some legacy image viewers and websites.
  - **Animated GIF (grayscale):** animated GIF, saved as a grayscale image, resulting in slightly smaller files.
- **Video length:** Set total replay time of the video by adjusting the video frame rate.
- **Video frame rate:** Set replay frame rate of the video by adjusting video length.

Advanced

- **Forward-backward:** After generating images by animating in forward direction, adds animation in reverse direction as well. It removes the "jump" at the end of the animation when it is played repeatedly in a loop.
- **Repeat:** Repeat the entire animation the specified number of times. It is useful for making animations longer (e.g., for uploading to YouTube).
- **ffmpeg executable:** Path to ffmpeg executable. Only used if video export is selected. Requires installation of *ffmpeg video encoder*.
- **Video extra options:** Options for ffmpeg that controls video format and quality. Only used if video export is selected.
  - These parameters are already specified by the module and therefore should not be included in the extra options: `-i (input files) -y (overwrite without asking) -r (frame rate) -start_number`.
  - Information about available options:
    * https://trac.ffmpeg.org/wiki/Encode/H.264
    * https://trac.ffmpeg.org/wiki/Encode/MPEG-4
    * https://trac.ffmpeg.org/wiki/Encode/YouTube
    * https://ffmpeg.org/ffmpeg-all.html
- **Image file name pattern:** Defines image file naming pattern. `%05d` will be replaced by the image number (5 numbers, padded with zeros). This is only used if image series output is selected.
- **Lightbox image columns:** Number of columns in the generated lightbox image.
- **Maximum number of images:** Specifies the maximum range of the "number of images" slider. Useful for creating very long animations.
- **Output volume node:** If "single frame" capture mode is selected then the output can be saved into the selected volume node.
- **View controllers:** Show view controllers. If unchecked then view controllers will be temporarily hidden during screen capture.
- **Transparent background:** If checked then images will be captured with transparent background.
- **Watermark image:** Adds a watermark image to the captured images.

  - **Position:** Position of the watermark image over the captured image.
  - **Size:** Watermark image size, as percentage of the original size.
  - **Opacity:** Watermark image opacity, larger value makes the watermark more visible (less transparent).

Setting up ffmpeg

FFmpeg library is not packaged with 3D Slicer due to large package size and licensing requirements for some video compression methods (see https://ffmpeg.org/legal.html). The FFmpeg library has to be downloaded and set up only once and 3D Slicer will remember its location.

Windows setup instructions

On Windows, Screen Capture model can automatically download and install ffmpeg when needed. Follow these instructions for manual setup:

- Download ffmpeg from here: https://ffmpeg.org/download.html (click Windows icon, select a package, for example `Download FFmpeg64-bit static`)
- Extract downloaded package (for example, to `C:\Users\Public`)
- In Advanced section, ffmpeg executable: select path for ffmpeg.exe (for example, `C:\Users\Public\ffmpeg-20160912-bc7066f-win32-static\bin\ffmpeg.exe`)

MacOS setup instructions

- Install homebrew (from https://brew.sh/)
- Run:

```
brew install ffmpeg
```

- In Advanced section, ffmpeg executable: select `/usr/local/bin/ffmpeg`

Linux setup instructions

- Run these commands from the terminal:

```
git clone https://git.ffmpeg.org/ffmpeg.git ffmpeg
cd ffmpeg
sudo apt-get install libx264-dev
./configure --enable-gpl --enable-libx264 --prefix=${HOME}
make install
```

- In Advanced section, ffmpeg executable: select `${HOME}/bin/ffmpeg`

Related modules

- Animator module in SlicerMorph extension allows creating more complex animations, such as cutting through a volume (by changing region of interest), adjusting volume rendering transfer functions, or exploding view of complex model assembly.
- *Scene Views* module can create snapshot of the entire scene content along with a screenshot, which are all saved in the scene.
- Capture Toolbar allows creation simple screenshots that are saved in the scene. The feature may be removed in the future. Screen Capture module's "Output volume node" feature can be used for saving screenshots in the scene.

Information for developers

- This is a Python scripted module. Source code is available here.
- Examples of capturing images are available in the *Script Repository*

Contributors

Andras Lasso (PerkLab, Queen's University)

Acknowledgements

This work was was funded by Cancer Care Ontario and the Ontario Consortium for Adaptive Interventions in Radiation Oncology (OCAIRO)

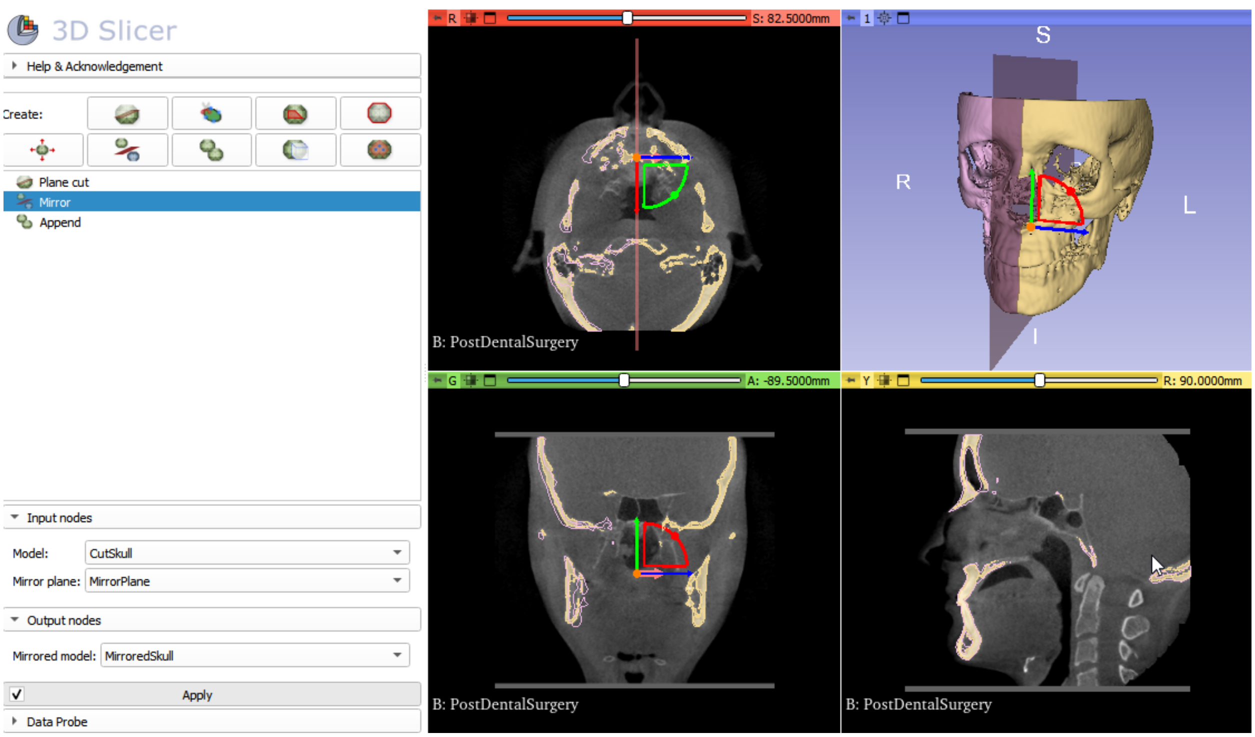
9.22 Surface Models

9.22.1 Dynamic Modeler

Overview

This is a module that provides an extensible framework for automatic processing of mesh nodes by executing "Tools" on the input to generate output mesh. Output of a tool can be used as input in another tool, which allows specification of complex editing operations. This is similar to "parametric editing" in engineering CAD software, but this module is specifically developed to work well on complex meshes used in biomedical applications (while most engineering CAD software does not directly support parametric editing of complex polygonal meshes).

See examples of potential applications using the Dynamic Modeler module here.



How to use

To create a new tool node, switch to the Dynamic Modeler module, and click on one of the tool buttons along the top of the module representing the tool that you would like to create.

Within the "Input nodes" section, select all required nodes. Note that some inputs can be selected multiple times, denoted by the `[#]` following the input name.

Within the "Parameters" section, you can adjust all parameters as needed.

Within the "Output nodes" section, select all desired outputs. At least one output must be selected.

Click on the Apply button to run the tool once, or check the checkbox on the Apply button to enable automatic update (so that outputs are automatically recomputed whenever inputs change). Tools cannot be run continuously if one of the input nodes is present in the output. The tool can still be run on demand by clicking the apply button.

Tools

Plane cut

Extracts a region of the surface by cutting it with one or more markup planes or slice view planes. The `Output model` contains what is on the negative side of the plane(s). If more than one plane is specified, the negative sides of the planes are combined using a union, intersection, or difference operation. The part of the `input model` that is not included in the `output model` is stored in the `complement output model`.

Input nodes

- `Model node`: The surface model to be cut.
- `Plane node (repeatable)`: The planes used to cut the model.

Parameters

- `Cap surface`: If enabled, creates a closed surface by triangulating the clipped region.
- `Operation type`: The method that will be used to combine multiple planes for cutting (union, intersection and difference boolean operations).

Output nodes

- `Output model (negative side)`: Portion of the cut model that is on the opposite side of the plane as the normal.
- `Complement output model (positive side)`: Portion of the cut model that is on the same side of the plane as the normal.

Curve cut

Extracts a region from the surface that is enclosed by a markup curve.

Input nodes

- `Model`: The surface model to be cut.
- `Curve`: The curve node (open or closed) that will be used to cut the model.
- `Inside point (optional)`: Points list node defining the region that will be considered the inside. If not defined, then the smallest region will be defined as the inside.

Parameters

- `Straight cut`: If enabled, the surface will be cut as close to the curve as possible. Otherwise the original edges will be preserved.

Output nodes

- `Inside model`: The cut region of the original surface model that represents the inside region.
- `Outside model`: The cut region of the original surface model that represents the outside region.

Boundary cut

Extracts a region from the surface that is enclosed by many markup curves and planes. In instances where there is ambiguity about which region should be extracted, a markup fiducial can be used to specify the region of interest.

Input nodes

- `Model node`: The surface model to be cut.
- `Border node (repeatable)`: Plane or curve nodes that define the boundaries of the regions to be cut.
- `Seed point node (optional)`: Points list node defining the regions that will be preserved in the output. If no points are defined, then the preserved region will be the region that is near the center of the defined boundaries.

Output nodes

- `Model node`: The region of the surface mesh that was cut by the specified boundaries.

Hollow

Create a shell from the surface of the model, effectively making it hollow.

Input nodes

- `Model`: The surface model to be converted to a shell.

Parameters

- `Shell thickness`: The thickness in millimeters of the resulting shell.

Output nodes

- `Hollowed model`: The output shell model.

Margin

Expands or shrinks a model by the specified margin.

Input nodes

- `Model`: Model to grow or shrink by the specified margin. Surface normals must be computed for the model, for example using Surface Toolbox module.

> **Note**
>
> Requires input models with precomputed surface normals.
>
> Normals can be computed using SurfaceToolbox module or in Python scripting using the `vtkPolyDataNormals` VTK filter.

May create self-intersecting mesh if the margin value is large or the model has sharp edges.

Parameters

- `Margin`: If positive value is specified then the model will be expanded by this much towards the surface normal, if negative then the model will be shrunken. Keep the absolute value low to avoid self-intersection.

Output nodes

- `Output model`: Model generated by growing or shrinking the input model.

Mirror

Reflects the points in a model across the input plane. Useful in conjunction with the plane cut tool to cut a model in half and then mirror the selected half across the cutting plane.

Input nodes

- `Model`: Surface model to be mirrored.
- `Mirror plane`: Plane that the input model will be mirrored across.

Output nodes

- `Mirrored model`: Model mirrored across the plane.

Append

Combine multiple models into a single output model node.

Input nodes

- `Model (repeatable)`: Models to be appended in the output.

Output nodes

- `Appended model`: Output model combining the input models.

ROI cut

Clips a vtkMRMLModelNode and returns the region of the model that is inside or outside the ROI. The tool can also add caps to the clipped regions to maintain a closed surface.

Input nodes

- `Model node`: Model node to be cut.
- `ROI node`: ROI node to cut the model node.

Parameters

- `Cap surface`: If enabled, create a closed surface by triangulating the clipped region.

Output nodes

- `Clipped output model (inside)`: Portion of the cut model that is inside the ROI.
- `Clipped output model (outside)`: Portion of the cut model that is outside the ROI.

Select by points

Allows selecting region(s) of a model node by specifying by markups fiducial points. Model points that are closer to the points than the specified selection distance are selected.

Input nodes

- `Model node`: Model node to select faces from.
- `Fiducials node`: Fiducials node to make the selection of model's faces.

Parameters

- `Selection distance`: Selection distance of model's points to input fiducials in millimeters.
- `Selection algorithm`: Method used to calculate points distance to seeds. "SphereRadius" method uses straight line distance. "GeodesicDistance" method uses distance on surface. Geodesic distance is computed using code from Krishnan K. "Geodesic Computations on Surfaces." article published in the VTK journal in June 2013.

Output nodes

- `Model with selection scalars`: All model points have a selected scalar value that is 0 or 1.
- `Model of the selected cells`: Model that only contains the selected faces of the input model.

Information for developers

See examples for using Dynamic Modeler tools in the *Script repository*.

Contributors

Authors:

- Kyle Sunderland (PerkLab, Queen's University)
- Andras Lasso (PerkLab, Queen's University)
- Jean-Christophe Fillion-Robin (Kitware)
- Mauro I. Dominguez
- Csaba Pinter (Ebatinca)

Acknowledgements

This module was originally developed with support from CANARIE's Research Software Program, OpenAnatomy, and Brigham and Women's Hospital through NIH grant R01MH112748.

9.22.2 Grayscale Model Maker

Overview

Create 3D surface models from grayscale data. This module uses Marching Cubes to create an isosurface at a given threshold. The resulting surface consists of triangles that separate a volume into regions below and above the threshold. The resulting surface can be smoothed and decimated. This model works on continuous data while the module Model Maker works on labeled (or discrete) data.

Panels and their use

IO: Input/output parameters

- **Input Volume** (*InputVolume*): Volume containing the input grayscale data.
- **Output Geometry** (*OutputGeometry*): Output that contains geometry model.

Grayscale Model Maker Parameters: Parameters used for making models.

- **Threshold** (*Threshold*): Grayscale threshold of isosurface. The resulting surface of triangles separates the volume into voxels that lie above (inside) and below (outside) the threshold.
- **Model Name** (*Name*): Name to use for this model.
- **Smooth** (*Smooth*): Number of smoothing iterations. If 0, no smoothing will be done.
- **Decimate** (*Decimate*): Target reduction during decimation, as a decimal percentage reduction in the number of polygons. If 0, no decimation will be done.
- **Split Normals?** (*SplitNormals*): Splitting normals is useful for visualizing sharp features. However it creates holes in surfaces which affect measurements
- **Compute Point Normals?** (*PointNormals*): Calculate the point normals? Calculated point normals make the surface appear smooth. Without point normals, the surface will appear faceted.

Advanced:

- **Debug** (*Debug*): Turn this flag on to log more details during execution.

Contributors

Nicole Aucoin (SPL, BWH), Bill Lorensen (GE)

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.

9.22.3 Label Map Smoothing

Overview

This filter smoothes a binary label map. With a label map as input, this filter runs an anti-alising algorithm followed by a Gaussian smoothing algorithm. The output is a smoothed label map.

Panels and their use

Label Selection Parameters: Parameters for selecting the label to smooth

- **Label to smooth** (*labelToSmooth*): The label to smooth. All others will be ignored. If no label is selected by the user, the maximum label in the image is chosen by default.

AntiAliasing Parameters: Parameters for the AntiAliasing algorithm

- **Number of Iterations** (*numberOfIterations*): The number of iterations of the level set AntiAliasing algorithm
- **Maximum RMS Error** (*maxRMSError*): The maximum RMS error.

Gaussian Smoothing Parameters: Parameters for Gaussian Smoothing

- **Sigma** (*gaussianSigma*): The standard deviation of the Gaussian kernel

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input label map to smooth
- **Output Volume** (*outputVolume*): Smoothed label map

Contributors

Dirk Padfield (GE), Josh Cates (Utah), Ross Whitaker (Utah)

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 filter is based on work developed at the University of Utah, and implemented at GE Research.

9.22.4 Merge Models

Overview

Merge the polydata from two input models and output a new model with the combined polydata. Uses the vtkAppend-PolyData filter. Works on .vtp and .vtk surface files.

Panels and their use

IO: Input/output

- **Model 1** (*Model1*): Input model 1
- **Model 2** (*Model2*): Input model 2
- **Output Model** (*ModelOutput*): Output model

Contributors

Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Daniel Haehn (SPL, BWH, UPenn)

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.

9.22.5 Model Maker

Overview

Create 3D surface models from segmented data.Models are imported into Slicer under a model hierarchy node in a MRML scene. The model colors are set by the color table associated with the input volume (these colors will only be visible if you load the model scene file).IO:Specify an Input Volume that is a segmented label map volume. Create a new Models hierarchy to provide a structure to contain the return models created from the input volume.Create Multiple:If you specify a list of Labels, it will over ride any start/end label settings.If you click Generate All it will over ride the list of labels and any start/end label settings.Model Maker Parameters:You can set the number of smoothing iterations, target reduction in number of polygons (decimal percentage). Use 0 and 0 if you wish no smoothing nor decimation.You can set the flags to split normals or generate point normals in this pane as well.You can save a copy of the models after intermediate steps (marching cubes, smoothing, and decimation if not joint smoothing, otherwise just after decimation); these models are not saved in the mrml file, turn off deleting temporary files first in the python window:slicer.modules.modelmaker.cliModuleLogic().DeleteTemporaryFilesOff()

Panels and their use

IO: Input/output parameters

- **Input Volume** (*InputVolume*): Input label map. The Input Volume drop down menu is populated with the label map volumes that are present in the scene, select one from which to generate models.
- **Models** (*ModelSceneFile*): Generated models, under a model hierarchy node. Models are imported into Slicer under a model hierarchy node, and their colors are set by the color table associated with the input label map volume. The model hierarchy node must be created before running the model maker, by selecting Create New ModelHierarchy from the Models drop down menu. If you're running from the command line, a model hierarchy node in a new mrml scene will be created for you.

Create Multiple: Create more than one model at the same time, used for continuous ranges of labels.

- **Model Name** (*Name*): Name to use for this model. Any text entered in the entry box will be the starting string for the created model file names. The label number and the color name will also be part of the file name. If making multiple models, use this as a prefix to the label and color name.
- **Generate All Models** (*GenerateAll*): Generate models for all labels in the input volume. select this option if you want to create all models that correspond to all values in a labelmap volume (using the Joint Smoothing option below is useful with this option). Ignores Labels, Start Label, End Label settings. Skips label 0.

Model Maker Parameters: Parameters used for making models.

- **Labels** (*Labels*): A comma separated list of label values from which to make models. f you specify a list of Labels, it will override any start/end label settings. If you click Generate All Models it will override the list of labels and any start/end label settings.
- **Start Label** (*StartLabel*): If you want to specify a continuous range of labels from which to generate models, enter the lower label here. Voxel value from which to start making models. Used instead of the label list to specify a range (make sure the label list is empty or it will over ride this).
- **End Label** (*EndLabel*): If you want to specify a continuous range of labels from which to generate models, enter the higher label here. Voxel value up to which to continue making models. Skip any values with zero voxels.

- **Skip Un-Named Labels** (*SkipUnNamed*): Select this to not generate models from labels that do not have names defined in the color look up table associated with the input label map. If true, only models which have an entry in the color table will be generated. If false, generate all models that exist within the label range.
- **Joint Smoothing** (*JointSmoothing*): This will ensure that all resulting models fit together smoothly, like jigsaw puzzle pieces. Otherwise the models will be smoothed independently and may overlap.
- **Smooth** (*Smooth*): Here you can set the number of smoothing iterations for Laplacian smoothing, or the degree of the polynomial approximating the windowed Sinc function. Use 0 if you wish no smoothing.
- **Filter Type** (*FilterType*): You can control the type of smoothing done on the models by selecting a filter type of either Sinc or Laplacian.
- **Decimate** (*Decimate*): Chose the target reduction in number of polygons as a decimal percentage (between 0 and 1) of the number of polygons. Specifies the percentage of triangles to be removed. For example, 0.1 means 10% reduction and 0.9 means 90% reduction.
- **Split Normals** (*SplitNormals*): Splitting normals is useful for visualizing sharp features. However it creates holes in surfaces which affects measurements.
- **Point Normals** (*PointNormals*): Turn this flag on if you wish to calculate the normal vectors for the points.
- **Pad** (*Pad*): Pad the input volume with zero value voxels on all 6 faces in order to ensure the production of closed surfaces. Sets the origin translation and extent translation so that the models still line up with the unpadded input volume.

Debug:

- **Color Hierarchy** (*ModelHierarchyFile*): A mrml file that contains a template model hierarchy tree with a hierarchy node per color used in the input volume's color table. Color names used for the models are matched to template hierarchy names to create a multi level output tree. Create a hierarchy in the Models GUI and save a scene, then clean it up to remove everything but the model hierarchy and display nodes.
- **Save Intermediate Models** (*SaveIntermediateModels*): You can save a copy of the models after each of the intermediate steps (marching cubes, smoothing, and decimation if not joint smoothing, otherwise just after decimation). These intermediate models are not saved in the mrml file, you have to load them manually after turning off deleting temporary files in they python console (View ->Python Interactor) using the following command slicer.modules.modelmaker.cliModuleLogic().DeleteTemporaryFilesOff().
- **Debug** (*debug*): turn this flag on in order to see debugging output (look in the Error Log window that is accessed via the View menu)

Contributors

Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Bill Lorensen (GE)

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.

9.22.6 Model To LabelMap

Overview

Intersects an input model with a reference volume and produces an output label map, filling voxels inside the model with the specified label value.

Panels and their use

Settings: Parameter settings

- **Label value** (*labelValue*): The unsigned char label value to use in the output label map.

IO: Input/output

- **Input Volume** (*InputVolume*): Output volume will have the same origin, spacing, axis directions, and extent as this volume.
- **Model** (*surface*): Input model
- **Output Volume** (*OutputVolume*): Unsigned char label map volume

Contributors

Andras Lasso (PerkLab), Csaba Pinter (PerkLab), Nicole Aucoin (SPL, BWH), Xiaodong Tao (GE)

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.

9.22.7 Probe Volume With Model

Overview

Paint a model by a volume (using vtkProbeFilter).

Panels and their use

IO: Input/output parameters

- **Input volume** (*InputVolume*): Volume to use to "paint" the model
- **Input model** (*InputModel*): Input model
- **Output model** (*OutputModel*): Output "painted" model
- **Output array name** (*OutputArrayName*): Name of the array that will contain the voxel values.

Contributors

Lauren O'Donnell (SPL, BWH)

Acknowledgements

BWH, NCIGT/LMI

9.22.8 Surface Toolbox

Overview

This module supports various cleanup and optimization processes on surface models.

Panels and their use

Select the input and output models, and then enable the stages of the pipeline by selecting the tool buttons.

Stages that include parameters will open up when they are enabled.

Click apply to activate the pipeline and then click the Toggle button to compare the model before and after the operation.

The module includes tools for:

- *Surface model cleaning*
- *Surface model uniform remeshing*
- *Surface model decimation* (reduction of the number of triangles).
- *Surface model smoothing*
- *Surface model holes filling*
- *Surface model normal computation*
- *Surface model mirroring*
- Surface model geometric transformations: *Scale* and *Translate*
- *Edge extraction*
- *Extraction of the largest component in the surface model*

These tools can be combined for multiple surface model processing effects.

Clean

This tool can merge coincident points, remove unused points (i.e. not used by any cell) and treat degenerate cells.

Uniform remeshing

Uniformly remesh the surface using ACVD algorithm.

This resampling typically provides higher quality meshes than decimation, with similar computation time.

- **Number of points:** Number of desired points in the output mesh. Use higher number to preserve more details.
- **Subdivide:** Number of subdivision to perform before remeshing. Each subdivision creates 4 triangles for each input triangle. This is needed if the required number of desired points is higher than the number of points in the input mesh, or there are some too large cells in the input mesh.

Note

It requires `pyacvd` Python package for which the user is required to confirm the installation.

Decimate

Perform a topology-preserving reduction of surface triangles. The user can modify the following parameters:

- **Reduction:** Target reduction factor during decimation. Ratio of triangles that are requested to be eliminated. 0.8 means that the surface model size is requested to be reduced by 80%.
- **Boundary deletion:** If enabled then *FastQuadric* method is used (it provides more even element sizes but cannot be forced to preserve boundary), otherwise *DecimatePro* method is used (that can preserve boundary edges but tend to create more ill-shaped triangles).

Smooth

Performs surface model smoothing on a surface model according to the following parameters:

- **Method:** Selects the smoothing method: either a *Laplacian* filter or *Taubin's non-shrinking* algorithm.
- **Iterations:** Number of smoothing iterations.
- **Pass band:** The pass-band value for the windowed sinc filter in the Taubin's non-shrinking algorithm. This should be between 0 and 2, where lower values cause more smoothing.
- **Boundary smoothing:** If enabled, boundary edges will be smoothed. Otherwise, the edges will remain fixed.

Fill holes

Fills up a hole in a open surface model.

- **Maximum hole size:** Specifies the maximum size of holes that will be filled. This is represented as a radius to the bounding circumsphere containing the hole. Note that this is an approximate area; the actual area cannot be computed without first triangulating the hole.

Compute surface normals

Generate surface normals for geometry algorithms or for improving visualization through lighting.

- **Auto orient normals:** Orient normals outwards from a closed surface.
- **Flip normals:** Flip normal direction from its current or auto-oriented state.
- **Splitting:** Allow sharp change in normals where angle between neighbor faces is above a threshold (indicated feature angle).
  - **Feature angle for splitting:** Normals will be split only along those edges where angle is larger than this value.

Mirror

Mirror the surface model along one or more axes.

- **X axis:** Enable/disable the use of the X axis for the mirroring operation.
- **Y axis:** Enable/disable the use of the Y axis for the mirroring operation.
- **Z axis:** Enable/disable the use of the Z axis for the mirroring operation.

Scale

Performs a scaling transformation on the surface. This transformation can be non-uniform (different scaling factors for different axes).

- **Scale X:** Scaling factor along the X axis.
- **Scale Y:** Scaling factor along the Y axis.
- **Scale Z:** Scaling factor along the Z axis.

Translate

Performs a translation transformation on the surface.

- **Center:** enables/disables centering the model in the origin (this is done by translating the center of the object's bounding box to the origin) before performing the translation.

- **Translate X:** specifies the translation along the X axis.
- **Translate Y:** specifies the translation along the Y axis.
- **Translate Z:** specifies the translation along the Z axis.

Extract edges

Extract the edges of the surface model. The extraction can be controlled with the following parameters:

- **Boundary edges:** Enable/disable the extraction of boundary edges (edges used only by one polygon or a line cell).
- **Feature edges:** Enable/disable the extraction of feature edges (edges used by two triangles and whose dihedral angle is larger than the specified feature angle).
- **Feature angle:** Minimum angle to consider an edge to be a feature edge.
- **Manifold edges:** Enable/disable the extraction of manifold edges (edges used exactly by two polygons).
- **Non-manifold edges:** Enable/disable the extraction of non-manifold edges (edges used by three or more polygons).

Extract largest component

Enable the extraction of the largest connected component.

Contributors

- Luca Antiga (Orobix)
- Steve Pieper (BWH)
- Beatriz Paniagua (Kitware)
- Martin Styner (UNC)
- Hans Johnson (University of Iowa)
- Ben Wilson (Kitware)
- Jean-Christophe Fillion-Robin (Kitware)
- Andras Lasso (PerkLab, Queen's University)
- Sara Rolfe (Seattle Children's Research Institute)

Acknowledgments

Development was funded in part by NIH National Institute of Biomedical Imaging Bioengineering R01EB021391 (Shape Analysis Toolbox for Medical Image Computing Projects) and by the NSF National Science Foundation under Grant DBI-1759883 (An Integrated Platform for Retrieval, Visualization and Analysis of 3D Morphology From Digital Biological Collections).

9.23 Converters

9.23.1 Create a DICOM Series

Overview

Create a DICOM Series from a Slicer volume. User can specify values for selected DICOM tags in the UI. Given the number of tags DICOM series have, it is impossible to expose all tags in UI. So only important tags can be set by the user.

Panels and their use

Input: Input parameters

- **Input Volume** (*inputVolume*): Input volume to be resampled

Output: Output parameters

- **DICOM Directory** (*dicomDirectory*): The directory to contain the DICOM series.
- **DICOM filename prefix** (*dicomPrefix*): The prefix of the DICOM filename.
- **DICOM file number format** (*dicomNumberFormat*): The printf-style format to be used when creating the per-slice DICOM filename. The leading % sign can be omitted.
- **Reverse Slices** (*reverseImages*): Reverse the slices.
- **Use Compression** (*useCompression*): Compress the output pixel data.
- **Output Type:** (*Type*): Type for the new output volume.

Patient Parameters: Parameters that apply to a patient

- **Patient Name** (*patientName*): The name of the patient (0010,0010)
- **Patient ID** (*patientID*): The patient ID (0010,0020). If set to [random] then a random ID will be generated.
- **Patient Birth Date** (*patientBirthDate*): Patient birth date (0010,0030) in the format YYYYMMDD.
- **Patient Sex** (*patientSex*): Patient sex (0010,0040). M=male, F=female, O=other, [unknown]=not specified
- **Patient Comments** (*patientComments*): Patient comments (0010,4000)

Study Parameters: Parameters that apply to a study

- **Study ID** (*studyID*): The study ID (0020,0010)
- **Study Date** (*studyDate*): The date of the study (0008,0020) in the format YYYYMMDD. If set to "[now]" then the current date is used.
- **Study Time** (*studyTime*): The time of the study (0008,0030) in the format HHMMSS. If set to "[now]" then the current time is used.
- **Study Comments** (*studyComments*): Study comments (0032,4000)
- **Study Description** (*studyDescription*): Study description (0008,1030)
- **Modality** (*modality*): Modality (0008,0060)
- **Manufacturer** (*manufacturer*): Manufacturer (0008,0070)
- **Model** (*model*): model (0008,1090)

Series Parameters: Parameters that apply to a series

- **Series Number** (*seriesNumber*): The series number (0020,0011)
- **Series Description** (*seriesDescription*): Series description (0008,103E)
- **Series Date** (*seriesDate*): The date of the series (0008,0021) in the format YYYYMMDD. If set to "[now]" then the current date is used.
- **Series Time** (*seriesTime*): The time of the series (0008,0031) in the format HHMMSS. If set to "[now]" then the current time is used.
- **Patient Position:** (*patientPosition*): Patient position descriptor relative to the equipment.

Image Parameters: Parameters that apply to the images and data in each image

- **Window center** (*windowCenter*): Window center (0028,1050). Specify a linear conversion from stored pixel values (after Rescale Slope and Intercept have been applied) to values to be displayed. Window Center contains the input value that is the center of the window. If either window center or width is undefined then the window is set to the full intensity range of the image.
- **Window width** (*windowWidth*): Window width (0028,1051). Specify a linear conversion from stored pixel values (after Rescale Slope and Intercept have been applied) to values to be displayed. Window Width contains the width of the window. If either window center or width is undefined then the window is set to the full intensity range of the image.
- **Rescale intercept** (*rescaleIntercept*): Rescale intercept (0028,1052). Converts pixel values on disk to pixel values in memory. (Pixel value in memory) = (Pixel value on disk) * rescaleSlope + rescaleIntercept. Default is 0.0. Data values are converted on write (the data is scaled and shifted so that the slope and interscept will bring it back to the current intensity range).
- **Rescale slope** (*rescaleSlope*): Rescale slope (0028,1053). Converts pixel values on disk to pixel values in memory. (Pixel value in memory) = (Pixel value on disk) * rescaleSlope + rescaleInterscept. Default is 1.0. Data values are converted on write (the data is scaled and shifted so that the slope and interscept will bring it back to the current intensity range).
- **Rescale Type** (*rescaleType*): Specifies the output units of the rescaled image (0008,1054). Leave it blank to set it automatically (Hounsfield unit for CT, unspecified for others).
- **Content Date** (*contentDate*): The date of the image content (0008,0023) in the format YYYYMMDD. If set to "[now]" then the current date is used.
- **Content Time** (*contentTime*): The time of the image content (0008,0033) in the format HHMMSS. If set to "[now]" then the current time is used.

Unique Identifiers (UIDs): Unique identifiers (UIDs) that allow appending frames to existing studies or series. To generate UIDs automatically, leave all of them blank.

- **Study Instance UID** (*studyInstanceUID*): The study instance UID (0020,000d). Leave it blank to generate UIDs automatically.
- **Series Instance UID** (*seriesInstanceUID*): The series instance UID (0020,000e). Leave it blank to generate UIDs automatically.
- **Frame of Reference UID** (*frameOfReferenceUID*): The frame of reference UID (0020,0052). Leave it blank to generate UIDs automatically.

Contributors

Bill Lorensen (GE)

Acknowledgements

This command module was derived from Insight/Examples (copyright) Insight Software Consortium

9.23.2 Crop Volume

Overview

This module is used to crop (i.e. remove unwanted regions) from scalar volumes. The removal process takes place with the help of a region of interest (ROI).

This region of interest can be manipulated either from the 3D View (possibly, with the help of volume rendering or 3D surface models) or from any of the Slice views (2D). In addition, the module includes advance functionality to fill the region outside the original volume, and control the spacing and the interpolation scheme used inside the cropped region.

You can use this module to crop a DTI or DWI volume.

Use Cases

Most frequently used for these scenarios:

- **Segmentation of an object that occupies small portion of the image**. Cropping the ROI with that object can simplify your segmentation task and reduce both the memory requirements and processing time.
- **Segmentation of an image that has very different resolution along different axes** (for example, 0.2mm with in an image slice, 1.0mm between slices). Isotropic spacing option ensures that the output volume has equal spacing on all axes. Doing this as part of cropping is useful, because forcing the same (high) resolution in all axes would increase the total number of voxels (and therefore memory need, processing times, etc.), but removing irrelevant parts of the volume can compensate for this increase in size.
- **Registration of two objects that occupy smaller portions of the image**. In this scenario, cropping will allow you to focus the processing at the region of interest, and simplify registration initialization.
- **Fix orientation of the volume**: If the orientation of the input volume is not correct (i.e., does not correspond to the anatomical axes) then the "Reorient volume" section can be used for adjusting the orientation by clicking on `Initialize`, use the rotation handles in views to fix the orientation, then click `Apply`. After that it is recommended to change the ROI fitting mode (dropdown menu of `Fit to Volume` button) to `Align to world axes + Resize` so that the cropped volume's axes are aligned to anatomical directions.
- **Definition of new axis directions** for your image: *Interpolated cropping options* allows the output volume to have different axis directions than the original volume. The output volume's axis directions will be aligned with the ROI widget's axes. ROI widget axes can be rotated using the *Transforms* module.
- **Cropping of oblique sub-volumes**: This can be done by placing either or both of input volume and ROI under transform(s). These transforms will be taken into account while preparing the output.

Panels and their use

Crop Volume

- **Parameter set:** Save/retrieve cropping presets.

IO

- **Input volume:** The scalar volume to crop.
  - **Reorient volume:** Useful for correcting orientation of the volume before cropping. Click on `Initialize` to show rotation handles, adjust the orientation, then click `Apply` to permanently change the volume orientation and hide the rotation handles.
- **Input ROI:** The region of interest driving the cropping process. ROIs have a box-like shape in which the interior of the box is the region to preserve and the exterior is the region to exclude. This combobox widget will allow the user to select an already existing ROI or create a new one (in addition, this widget will provide additional options such as rename or edit an existing ROI).
  - **Display ROI:** Turn on/off the display of the ROI representation. If this is on, a representation of the ROI will be visible in the Slice views (2D) or in the 3D view. The resulting ROI can be manipulated interactively in any of the Slice views (2D) or the 3D view.
  - **Fit to Volume:** This will resize the ROI to fit the extent of the Input volume specified. The button has a dropdown menu to tune its behavior, with the following options:
    * **Align to volume axes + Resize:** rotate the ROI axes to align the ROI box orientation (and thus the cropped volume axes) to match the input volume axes directions. This is useful if the goal is to minimize changes compared to the input volume.
    * **Align to world axes + Resize:** rotate the ROI axes to align the ROI box orientation (and thus the cropped volume axes) to the world coordinate system. This is useful to make the cropped volume axes to be anatomically aligned.
    * **Resize only:** only resize the ROI and keep its current orientation. This is useful if the ROI orientation is already set to the desired value and only the size of the ROI needs to be adjusted to fully enclose the input volume.
- **Output volume:** The output volume that represents the result of the cropping operation. This widget allows for the selection of an already existing volume (e.g., the input volume itself) or the creation of a new output volume node to store the results. By default, if no node is selected, output volume is created automatically.

After setting these parameters the user can click the Apply button to perform the operation.

> **Warning**
>
> The user should be careful when selecting the input and output volumes: selecting the same output volume as the input may render the input volume unusable for other operations.

Advanced

- **Fill value:** The value to fill the voxels of the ROI that fall outside the input volume. If the ROI remains inside the input volume, this parameter has no effect.
- **Interpolated cropping:** This enables/disables the selection of the *Interpolated cropping options* for the cropped output volume. If disabled, then only the extent of the volume is changed with spacing and axes directions remaining exactly the same. These parameters are controlled by the following widgets:

Interpolated cropping options

- **Spacing scale:** voxel spacing for the resulting cropped output volume. This parameter is a scale factor. The output spacing is defined by multiplying the input spacing in each dimension by the user-specified coefficient. If the value is greater than 1.0, then the cropped volume will have lower resolution than the input volume; if the value is smaller than 1.0, then the cropped volume will have higher resolution than the input volume. For example, if the input spacing is 1x1x1.4, and the scaling coefficient is 0.5, the output volume will have spacing 0.5x0.5x0.7, effectively doubling the resolution of the output image.

- **Isotropic spacing:** Enable/disable isotropic spacing. When this is enabled, the voxel spacing of the output volume will be set to the lowest spacing in any of the axes present in the input volume.
- **Interpolator:** Type of interpolation used: `Nearest neighbor`, `Linear`, `Windowed Sinc`, `B-spline` (see *Resample Scalar/Vector/DWI Volume* module for details). For subvolumes being extracted from a label volume, you should use `Nearest Neighbor` interpolator. Otherwise `B-spline` is the preferred choice. `Linear` interpolator requires less computation, which may be important for very large ROIs.

Volume Information

This section shows preview of spacing and dimensions of the cropped **output volume** and, for comparison, the **input volume**. Average dimensions of a volume is about 200-300 voxels along each axis. If the resolution is increased using **Spacing scale** and **Isotropic spacing** then it is recommended to reduce the region of interest so that the dimensions are not increased significantly.

Related modules

- The cropping functionality of this module is enabled by *Resample Scalar/Vector/DWI Volume* module, which is called internally.

Contributors

- Andrey Fedorov (BWH, SPL)
- Ron Kikinis (BWH, SPL).

Acknowledgments

This work was supported by NIH grants R01CA111288 and U01CA151261, NA-MIC, NAC and the Slicer Community.

9.23.3 Orient Scalar Volume

Overview

Orients an output volume. Rearranges the slices in a volume according to the selected orientation. The slices are not interpolated. They are just reordered and/or permuted. The resulting volume will cover the original volume. NOTE: since Slicer takes into account the orientation of a volume, the re-oriented volume will not show any difference from the original volume, To see the difference, save the volume and display it with a system that either ignores the orientation of the image (e.g. Paraview) or displays individual images.

Panels and their use

IO: Input/output parameters

- **Input Volume 1** (*inputVolume1*): Input volume 1
- **Output Volume** (*outputVolume*): The oriented volume

Orientation Parameters: Orientation of output. Note that these orientation code use ITK convention: each letters refer to where each axis originates FROM. This is opposite of commonly used orientation conventions, for example to get image with image axes corresponding to LPS coordinate system axis directions, this parameter has to be set to RAI.

- **Orientation** (*orientation*): Orientation choices

Contributors

Bill Lorensen (GE)

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.

9.23.4 Vector to Scalar Volume

Overview

This module is useful for converting RGB (vector) volumes to one component (scalar) grayscale volumes.

Vector volumes are typically loaded from a series of `jpg`, `png`, `bmp`, or other classic 2D image formats.

While vector volumes can be visualized in the slice viewers, many slicer operations only operate on scalar volumes (such as segmentation and registration modules).

Panels and their use

- **Parameters**:
  - **Input vector volume**
  - **Conversion Method**:
    - **Luminance**: convert RGB images to scalar using luminance as implemented in vtkImageLuminance (scalar = 0.30*R* + *0.59*G + 0.11*B).
    - **Average**: computes the mean of all the components.
    - **Single Component Extraction**: extract single components from any vector image.
  - **Output scalar volume**

Related extensions and modules

- The SlicerMorph extension's ImageStacks module supports reading and other operations on volumetric vector (color) image formats.

Contributors

- Steve Pieper (Isomics)
- Pablo Hernandez-Cerdan (Kitware)
- Jean-Christophe Fillion-Robin (Kitware)
- Andras Lasso (PerkLab)

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.

9.24 Developer Tools

These modules are intended for module developers.

9.24.1 Cameras

Overview

Bring core support for multiple cameras and 3D views in Slicer.

The multi views and cameras framework is available from the "Camera" modules. This module displays two pull-down menus. The first one, "View", lists all available views and lets the user create new 3D views. The second one, "Camera", lists all available cameras, and lets the user create new cameras.

Only one camera is used by a view at a time. When a view is selected from the pull-down, the camera it is currently using is automatically selected in the second pull-down.

- **Create a new camera:** In the "Camera" pull-down menu, select "Create New Camera": a new camera node is created (most likely named "Camera1", as opposed to the default "Camera" node), and automatically assigned to the currently selected view node (named "View" by default). Try interacting with the 3D view.
- **Assign a camera to a view:** Select any camera node from the "Camera" pull-down menu: the camera is assigned to the currently selected view. For example, try selecting back the "Camera" node if you have created a "Camera1" node in the previous step, and you should notice the 3D view on the right update itself to reflect the different point of view. Note: a camera can not be shared between two views: selecting a camera already used by a view will effectively swap the cameras between the two views.
- **Create a new view:** Select the "Tabbed 3D Layout" from the layout button in the toolbar. In the "View" pull-down menu, select "Create New View": a new view node is created (most likely named "View1", as opposed to the default "View" node), and displayed on the right under a new tab. You can select which view to display by clicking on the corresponding tab in the "Tabbed 3D Layout". Interacting in that view will automatically mark it

as "active"; there can only be one "active" view at a time in Slicer, and it always feature a thin dark blue border. Since a view can not exist without a camera, a new camera node is automatically created and assigned to this new view (most likely named "Camera2" if you have created "Camera1" in the previous steps). At this point, you can assign any of the previously created cameras to this new view (say, the "Camera" or "Camera1" nodes).

Panels and their use

- **Cameras**: List the 3D views and cameras in the scene.
  - **View**: Select a 3D view. "View" is the main 3D view, additional 3D views are named "View_1", "View_2" etc. On selection, the view active camera is automatically selected in the Camera pull-down.
  - **Camera**: Active camera of the current view.

Contributors

Julien Finet (Kitware), Sebastien Barré (Kitware)

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.

9.24.2 Event Broker

Overview

Profiling tool for the developers. The event broker manages observations between VTK objects in Slicer. Observed objects are populated in the tree view, their observers are listed under each observed object events. Selecting a VTK object in the list, dumps (PrintSelf()) all its information.

Panels and their use

- **Actions**: Actions in the view.
  - **Refresh**: Rescan the event broker and populate the tree view with the observers of the MRML scene, nodes, logics.
  - **Reset Times**: Reset the time spent in all callbacks to 0 second. To be used with "Show observations with Elapsed Times
  - **Show observations with Elapsed Times >0**: Filter the tree view to show only objects for which time has been spent in at least one of its callback.

Contributors

Julien Finet (Kitware)

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.

9.24.3 Execution Model Tour

Overview

Shows one of each type of parameter.

Panels and their use

Scalar Parameters (árvíztűrő tükörfúrógép): Variations on scalar parameters

- **Integer Parameter** (*integerVariable*): An integer without constraints
- **Double Parameter** (*doubleVariable*): A double with constraints

Vector Parameters: Variations on vector parameters

- **Float Vector Parameter** (*floatVector*): A vector of floats
- **String Vector Parameter** (*stringVector*): A vector of strings

Enumeration Parameters: Variations on enumeration parameters

- **String Enumeration Parameter** (*stringChoice*): An enumeration of strings

Boolean Parameters: Variations on boolean parameters

- **Boolean Default true** (*boolean1*): A boolean default true
- **Boolean Default false** (*boolean2*): A boolean default false
- **Boolean No Default** (*boolean3*): A boolean with no default, should be defaulting to false

File, Directory and Image Parameters: Parameters that describe files and directories.

- **Input file**: An input file
- **Input Files**: Multiple input files
- **Output file**: An output file
- **Input directory**: An input directory. If no default is specified, the current directory is used,
- **Input image**: An input image
- **Input 4D Image**: Input 4D Image (txyz)
- **Output image**: An output image

Transform Parameters: Parameters that describe transforms.

- **Input transform**: A generic input transform
- **Input transform linear**: A linear input transform
- **Input transform nonlinear**: A nonlinear input transform
- **Input transform bspline**: A bspline input transform
- **Output transform**: A generic output transform
- **Output transform linear**: A linear output transform
- **Output transform nonlinear**: A nonlinear output transform
- **Output transform bspline**: A bspline output transform

Point Parameters: Parameters that describe point sets.

- **Seeds** (*seed*): Lists of points in the CLI correspond to slicer fiducial lists
- **Seeds file** (*seedsFile*): Test file of input fiducials, compared to seeds
- **Output seeds file** (*seedsOutFile*): Output file to read back in, compare to seeds with flipped settings on first fiducial

Geometry Parameters: Parameters that describe models.

- **Input Model** (*InputModel*): Input model
- **Output Model** (*OutputModel*): Output model
- **Models** (*ModelSceneFile*): Generated models, under a model hierarchy node. Models are imported into Slicer under a model hierarchy node. The model hierarchy node must be created before running the model maker, by selecting Create New ModelHierarchy from the Models drop down menu.

Index Parameters: Variations on parameters that use index rather than flags.

- **First index argument** (*arg0*): First index argument is an image
- **Second index argument** (*arg1*): Second index argument is an image

Regions of interest:

- **Input region list** (*regions*): List of regions to process

Measurements:

- **Input FA measurements** (*inputFA*): Array of FA values to process
- **Output FA measurements** (*outputFA*): Array of processed (output) FA values

Generic Tables:

- **Input Table** (*inputDT*): Array of Table values to process
- **Output Table** (*outputDT*): Array of processed (output) Table values

Simple return types:

- **An integer return value** (*anintegerreturn*): An example of an integer return type
- **A boolean return value** (*abooleanreturn*): An example of a boolean return type
- **A floating point return value** (*afloatreturn*): An example of a float return type
- **A double point return value** (*adoublereturn*): An example of a double return type
- **A string point return value** (*astringreturn*): An example of a string return type
- **An integer vector return value** (*anintegervectorreturn*): An example of an integer vector return type
- **A string enumeration return value** (*astringchoicereturn*): An enumeration of strings as a return type

Contributors

Daniel Blezek (GE), Bill Lorensen (GE)

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.

9.24.4 Extension Wizard

Overview

The Extension Wizard modules provides a graphical interface within Slicer to aid in the creation of Slicer extensions.

Panels and their use

- **Extension Tools**:
  - **Create Extension**: Create a new extension from a specified template, given a name and destination. The newly created template is automatically selected for editing.
  - **Select Extension**: Choose an existing extension to edit. If the extension provides scripted modules that are not already loaded, an option to load such modules is provided.
- **Extension Editor**:
  - **Name**: The name of the currently selected extension.
  - **Location**: The location (on disk) of the currently selected extension.
  - **Repository**: If available, the upstream URL of the repository hosting the extension.
  - **Contents**: A tree view showing the file contents of the currently selected extension.
  - **Add Module to Extension**: Create a new module from a specified template, given a name, and add it to the selected extension.
  - **Edit Extension Metadata**: Edit metadata associated with the extension (such as the name, contributors, etc.).

Settings

The Extension Wizard module provides a settings page, which is accessible via the *Application Settings*.

- **Built-in template path**: If found, displays the location of the built-in templates.
- **Additional template paths**: A list of additional locations containing categorized templates.
- **Additional template paths for <category>**: A list of additional locations containing templates for a particular category (e.g. extensions, modules).

A "template" is a directory containing a collection of files which comprise that template. Additional paths should point to directories which *contain* such template directories, not the directory of the template itself. A categorized template directory should have directories for one or more categories, which in turn contain templates.

The built-in templates provide an example of the correct layout for a categorized template directory. (Each category directory is in turn an example of a template collection for that category.)

Contributors

- Matthew Woehlke (Kitware)
- Jean-Christophe Fillion-Robin (Kitware)

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.

9.24.5 WebServer

Overview

Creates a fairly simple but powerful web server that can respond to http(s) requests with data from the current application state or modify the application state.

This module is meant to be the basis for implementing web applications that use Slicer as a remote render / computation engine or for controlling your Slicer instance for interaction with other system code like shell scripts or other applications.

There are three types of endpoints:

> **Note**
>
> The web server is integrated with the Qt event loop so it can be used together with the interactive session.

> **Warning**
>
> This module should be considered somewhat experimental and a likely security risk. Do not expose web server endpoints on the public internet without careful consideration.
>
> Because the web server uses standard http, there are many off-the-shelf security options, such as firewalls, ssh-tunneling, and authenticating proxies, that can be used to improve security of installations.

Panels and their use

- Start server: Launches web server listening on port `2016`. If the default port is in use, other ports are checked sequentially until an open port is found, allowing more than one Slicer web server to run concurrently.
- Stop server: Stop web server.
- Open static page in external browser: Display `docroot` using default operating system web browser found using `qt.QDesktopServices`.

- Open static page in internal browser: Display `docroot` using Slicer built-in web browser instantiated using `slicer.qSlicerWebWidget()`.
- Log output: If `Log to GUI` is enabled, access log and execution results are logged. Logs are cleared periodically.
- Clear Log: Clear log output displayed in the module panel.
- Advanced:
  - CORS: Enable/disable Cross-Origin Resource Sharing.
  - Slicer API: Enable/disable use of *Slicer endpoints* associated with the `/slicer` path.
  - Slicer API exec: Enable/disable remote execution of python code through `/slicer/exec` endpoint. See [Remote Control][#remote-control].
  - DICOMweb API: Enable/disable support of *DICOMWeb endpoints* associated with the `/dicom` path.
  - Static pages: Enable/disable serving of static files found in the `docroot` associated with the `/` path.
  - Log to Console: Enable/disable the logging of messages in the console.
  - Log to GUI: Enable/disable the logging of messages in the module panel.

Note

Logging to the console and/or the GUI is useful for learning about the software and for debugging, but slows down request handling and should be disabled for routine use.

Warning

The `Slicer API exec` option exposes the full python interface of Slicer running with the same permissions as the Slicer app itself. This means that users of that API can install arbitrary code on the system and execute it with the user's rights. In practice this means that the user of the API can perform actions such as deleting files, sending emails, or installing system software. Exposing these capabilities is intentional and aligned with the design of the module, but users should be aware that enabling this feature is effectively the same as giving the user of the API the password to whatever account is running Slicer.

Note also that even with the `Slicer API exec` disabled, it is possible that other endpoints expose vulnerabilities such as buffer overruns that could lead to server exploits. It is suggested that only trusted users be granted access to any of the API endpoints.

Warning

Cross-Origin Resource Sharing allows browser-based code hosted from any origin to access the Slicer API. That is, any javascript in a site opened in a browser on the machine running the server would have access to the API. While this feature is useful for some development or specific scenarios it should be used with caution. Note that CORS is enforced by the user's web browser, so even with CORS turned off it's possible for other software running with access to the port to access the API even if CORS is turned off.

Static endpoints

Hosts files out of the module's `docroot` like any standard http server.

Currently this is used just for examples, but note that this server can be used to host web applications of significant complexity with the option of interacting with the Slicer API.

OHIF DICOM viewer is included as an example (available at http://localhost:2016/browse). If the DICOMweb endpoint is enabled then this viewer can be used to quickly share content of the Slicer DICOM database with other computers on the same network. All users that can access the computer are assumed to be trusted - if the server is accessible to non-trusted people then it is recommended to restrict access by setting up a firewall or proxy server.

DICOMweb endpoints

Exposes the Slicer dicom database as a DICOMweb endpoint.

This version implements a subset of the `QIDO-RS` and `WADO-RS` specifications allowing to host a web app such as the OHIF Viewer.

Supported QIDO requests:

- `/dicom/studies`: get list of studies as json, optional query parameters: `offset`, `limit`, `PatientID`
- `/dicom/studies/<studyuid>/metadata`: get DICOM tags of the specified study as json
- `/dicom/studies/<studyuid>/series`: get list of series for a study as json
- `/dicom/studies/<studyuid>/series/<seriesuid>/metadata`: get DICOM tags of the specified series as json
- `/dicom/studies/<studyuid>/series/<seriesuid>/instances`: get list of instances for a series as json
- `/dicom/studies/<studyuid>/series/<seriesuid>/instances/<sopinstanceuid>`: download the instance
- `/dicom/studies/<studyuid>/series/<seriesuid>/instances/<sopinstanceuid>/metadata`: get DICOM tags of the specified instance as json

Supported WADO requests:

- `/dicom?object=<sopinstanceuid>`: downloads the specified instance

For OHIF version 2, change the `platform/viewer/public/config/default.js`, set the `servers` configuration key as follows.

```
servers: {
  dicomWeb: [
    {
      name: 'DCM4CHEE',
      wadoUriRoot: 'http://localhost:2016/dicom',
      qidoRoot: 'http://localhost:2016/dicom',
      wadoRoot: 'http://localhost:2016/dicom',
      qidoSupportsIncludeField: true,
      imageRendering: 'wadouri',
      thumbnailRendering: 'wadouri',
      enableStudyLazyLoad: true,
      supportsFuzzyMatching: true,
    },
  ],
},
```

Slicer endpoints

Full specification of the Slicer API is available at the bottom of this page. The API is subject to change.

Remote Control

The web server can also be accessed via other commands such as `curl`. A `dict` can be returned as a json object by setting it in the `__execResult` variable and enabling the `Slicer API exec` feature in the `Advanced` section.

For example, these commands may be used to download the MRHead sample data, change the screen layout and return a dictionary including the ID of the loaded volume:

```
curl -X POST localhost:2016/slicer/exec --data "import SampleData; volumeNode =
↪SampleData.SampleDataLogic().downloadMRHead(); slicer.app.layoutManager().
↪setLayout(slicer.vtkMRMLLayoutNode.SlicerLayoutOneUpRedSliceView); __execResult = {
↪'volumeNodeID': volumeNode.GetID()}"
```

Note

See the *Script Repository* for other examples of python code you could remotely execute.

Remote Rendering

There are several endpoints to get png images from slice views and 3D views. These endpoints allow control of slice offset or 3D camera view (see method doc strings in the source code for options since there is currently no auto-generated api documentation). These endpoints can be used as the `src` or `href` for html `img` or `a` tags or as WebGL textures as shown in the demo scripts.

Data Access

Give read/write access to features in Slicer's MRML scene and GUI.

`http GET`, `POST`, `DELETE` operations can be used to query, load, save, delete data in the scene.

Usage via slicerio Python package

The API can be conveniently used via the slicerio Python package **in any Python environment**. For example, to open an image file and a segmentation file in a single Slicer instance (without restarting a new instance for each file), run this code snippet:

```
import slicerio.server
slicerio.server.file_load("c:/tmp/MRHead.nrrd")
slicerio.server.file_load("c:/tmp/Segmentation.nrrd", "SegmentationFile")
```

note: If Slicer application is not running (or the server API is disabled) then `slicerio.server.file_load` will start a new Slicer instance by launching the Slicer executable specified either in the `slicer_executable` function argument or in the `SLICER_EXECUTABLE` environment variable.

Data nodes can be retrieved from Slicer by saving into local file:

```
slicerio.server.file_save("c:/tmp/MRHeadOutput.nrrd", name="MRHead", properties={
↪'useCompression': False})
```

Properties of nodes can be queried using `slicerio.server.node_properties`:

```
properties = slicerio.server.node_properties(name="Segmentation")[0]
segments = properties["Segmentation"]["Segments"]
for segmentId in segments:
```

(continues on next page)

(continued from previous page)

```
    segment = segments[segmentId]
    print(f"{segment['Name']} color: {segment['Color']}")
```

Full specification and more examples are available in the `slicerio` package documentation.

Usage via curl

For example, to save a nrrd version of a volume in Slicer, you can use:

```
curl -v http://localhost:2016/slicer/volume\&id='MRHead' -o /tmp/local.nrrd
```

Currently only limited forms are supported (scalar volumes and grid transforms).

Other endpoints allow get/set of transforms and fiducials.

Slicer REST API

Remote control (exec)

GET /exec

Run script in Slicer's Python interpreter, as if it was run in the application's Python console. It can be used to implement a Read Eval Print Loop (REPL).

Parameters:

- `source`: Python code to run

Return:

- 200 (application/json): Result of code running as json string. The result must be set in a `__execResult` variable (`dict` object)
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

Example:

```
curl -X POST localhost:2016/slicer/exec --data "slicer.app.layoutManager().
↪setLayout(slicer.vtkMRMLLayoutNode.SlicerLayoutOneUpRedSliceView)"
```

MRML data access

Get information on MRML nodes in the scene or load/save nodes.

Common parameters for data selection for all methods:

- `id`: select node that has this id.
- `class`: select nodes of this class (select nodes of this class (e.g., vtkMRMLVolumeNode, vtkMRMLSegmentationNode)
- `name`: select nodes of this name

If `id` is specified then `class` and `name` parameters are ignored. If both `class` and `name` are specified then those nodes will be selected that have fulfill both selection criteria.

GET /mrml and GET /mrml/names

Get names of the selected nodes.

Parameters:

- `id`: as described above
- `class`: as described above
- `name`: as described above

Return:

- 200 (application/json): list of node names.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /mrml/ids

Get ids of the selected nodes.

Parameters:

- `id`: as described above
- `class`: as described above
- `name`: as described above

Return:

- 200 (application/json): list of node ids.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /mrml/properties

Get properties of the selected nodes as a json object.

Parameters:

- `id`: as described above
- `class`: as described above
- `name`: as described above

Return:

- 200 (application/json): dictionary object, key is the node id, value is the node properties object.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /mrml/file

Save node to local file. Query parameters must be specified so that only a single node is selected.

Parameters:

- `id`: as described above
- `class`: as described above
- `name`: as described above
- `localfile`: filename to save the node to

- `useCompression`: specifies if the file should be saved using compression (`true` or `false`)

Return:

- 200 (application/json): JSON object, with `success` property set to true.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

POST /mrml/file

Load node from URL or local file into a node.

Parameters:

- `localfile`: Local filename to load the node from. If specified then `url` is ignored.
- `url`: Local or remote URL to load the file from.
- `filetype`: Specifies how to interpret the selected file. For example `VolumeFile`, `SegmentationFile`, `ModelFile`, `MarkupsFile`, `TransformFile`, `SceneFile`.

Return:

- 200 (application/json): JSON object, "success" property is set true and `loadedNodeIDs` property contains a list of loaded node ids.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

DELETE /mrml/file

Remove nodes from the scene. If no query parameters are specified then the whole scene is cleared.

Parameters:

- `id`: as described above
- `class`: as described above
- `name`: as described above
- `localfile`: filename to save the node to
- `useCompression`: specifies if the file should be saved using compression (`true` or `false`)

Return:

- 200 (application/json): JSON object, with `success` property set to true.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

PUT /mrml

Reload node that was originally loaded from a file, from the same file. This is useful if the input file is changed since it was last loaded into Slicer.

Parameters:

- `id`: as described above
- `class`: as described above
- `name`: as described above

Return:

- 200 (application/json): JSON object, "success" property is set true and `reloadedNodeIDs` property contains a list of reloaded node ids. "success" property is set to false if it was not possible to reload any nodes.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

User interface

PUT /gui

Shutdown the application.

Parameters:

- `contents`: show full application GUI (`full`) or viewers only (`viewers`)
- `viewersLayout`: set view layout, such as `fourup`, `oneup3d` (names derived from slicer.vtkMRMLLayoutNode constants - SlicerLayout…View)

Return:

- 200 (application/json): JSON object, with `success` property set to true.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /screenshot

Get screenshot of the application main window.

Return:

- 200 (image/png): screenshot image
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /slice

Get screenshot of a slice view after applying parameters.

Parameters:

- `view`: `red`, `yellow`, or `green`
- `scrollTo`: 0 to 1 for slice position within volume
- `offset`: mm offset relative to slice origin (position of slice slider)
- `size`: pixel size of output png
- `copySliceGeometryFrom`: view name of other slice to copy from
- `orientation`: `axial`, `sagittal`, `coronal`

Return:

- 200 (image/png): screenshot image
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /threeD

Get screenshot of the first 3D view after applying parameters.

Parameters:

- `lookFromAxis`: `L`, `R`, `A`, `P`, `I`, `S`

Return:

- 200 (image/png): screenshot image
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /timeimage

For timing and debugging - return an image with the current time rendered as text down to the hundredth of a second.

Parameters:

- `color`: hex encoded RGB of dashed border (default 333 for dark gray)

Return:

- 200 (image/png): rendered image
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

Other functions

GET /system/version

Get application version information as a json object.

Return:

- 200 (application/json): version information object.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

DELETE /system

Shutdown the application.

Return:

- 200 (application/json): JSON object, with `success` property set to true.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /tracking

Display/update position of a cursor (position marker cube) in the 3D view. This can be used to display position of a tracked object.

Parameters:

- `m`: 4x4 transformation matrix in column major order (position is last row). Matrix is overwritten if position or quaternion are provided
- `q`: quaternion in WXYZ order
- `p`: position (last column of transform)

Return:

- 200 (text/plain): plain text message for the user
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /sampledata

Load a sample data set into the scene.

Parameters:

- `name`: name of the sample data set (such as `MRHead`)

Return:

- 200 (text/plain): plain text message for the user
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /volumeSelection

Cycles through loaded volumes in the scene.

Parameters:

- `cmd`: either `next` or `previous` to indicate direction

Return:

- 200 (text/plain): plain text message for the user
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /volumes, GET /gridtransforms

Get a list of mrml volume or grid transform node names and ids.

Parameters:

- `cmd`: either `next` or `previous` to indicate direction

Return:

- 200 (application/json): list of json objects, with `name` and `id` attributes.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

Example of successful output:

```
[
    {"name": "Volume1", "id": "vtkMRMLScalarVolumeNode1"},
    {"name": "Volume2", "id": "vtkMRMLScalarVolumeNode2"},
]
```

GET /volume, GET /griddtransform

Retrieve the specified volume or grid transform as a .nrrd file.

Parameters:

- `id`: id of the node to get

Return:

- 200 (application/octet-stream): data stream of a nrrd file
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

POST /volume

Create or update a volume from a .nrrd file. Only uncompressed 3D volumes are accepted, in LPS coordinate system, with little endian short pixel type.

Parameters:

- `id`: id of the volume to create or update.

Return:

- 200 (application/json): data stream of a nrrd file
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

GET /fiducials

Retrieve basic information about all markup point lists (formerly called fiducial lists) in the scene.

Return:

- 200 (application/json): Basic information about all markup point lists.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

Example of successful output:


```
{
    "vtkMRMLMarkupsFiducialNode1": {
        "name": "F",
        "color": [1.0, 0.5000076295109483, 0.5000076295109483],
        "scale": 3.0,
        "markups": [
            {"label": "F-1", "position": [-35.422643698898014, 13.121414583492907, -10.
↪214302062988281]},
            {"label": "F-2", "position": [43.217879176918984, 41.565859027937364, -10.
↪214302062988281]},
            {"label": "F-3", "position": [39.8714739481608, -32.05505600474238, -10.
↪214302062988281]}]},
    "vtkMRMLMarkupsFiducialNode2": {
        "name": "F_1",
        "color": [1.0, 0.5000076295109483, 0.5000076295109483],
        "scale": 3.0,
        "markups": [
            {"label": "F_1-1", "position": [82.53814061482748, 13.121414583492907, -23.
↪599922978020956]},
            {"label": "F_1-2", "position": [-4.468395332884938, 13.121414583492907, 65.
↪07981558407056]}]}
}
```


PUT /fiducial

Set the location of a control point in a markups point list (formerly called fiducial list).

Parameters:

- `id`: id of the node to update.
- `r`: Right coordinate

- `a`: Anterior coordinate
- `s`: Superior coordinate

Return:

- 200 (application/json): JSON object, with `success` property set to true.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

POST /accessDICOMwebStudy

Access DICOMweb server to download requested study, add it to Slicer's dicom database, and load it into the scene.

Request body: json string with the following properties

- 'dicomWEBPrefix': is the start of the url
- 'dicomWEBStore': is the middle of the url
- 'studyUID': is the end of the url
- 'accessToken': is the authorization bearer token for the DICOMweb server

Return:

- 200 (application/json): JSON object, with `success` property set to true.
- 500 (application/json): In case of unexpected error. `message` attribute contains error message.

Related modules

- The OpenIGTLink Extension has some similar functionality customized for image guided therapy applications. It should be preferred for integration with imaging devices and use in a clinical setting or setting up continuous high-throughput image and transform streams.

Future work

Features have been added to this server based on the needs of demos and proof of concept prototypes. A more comprehensive mapping of the Slicer API to a web accessible API has not yet been performed. Similarly, the DICOMweb implementation is bare-bones and has only been implemented to the extent required to support a simple viewer scenario without performance optimizations. The existing framework could also be improved through the implementation of newer HTTP features and code refactoring.

History

The development of the first implementation was started by Steve Pieper in 2012 and has been developed over the years to include additional experiments. See https://github.com/pieper/SlicerWeb

Then, in November 2021, a stripped down version of the module addressing the most common expected use cases was proposed in pull request #5999.

In May 2022, the module was integrated into Slicer.

Contributors

- Steve Pieper, original author (Isomics)
- Andras Lasso, refactoring and Slicer integration (Queens)
- Jean-Christophe Fillion-Robin (Kitware)

Acknowledgements

This work was partially funded by NIH grant 3P41RR013218.

9.25 Testing

These modules are for testing correctness and performance of the application.

9.25.1 Performance Tests

Overview

Module to run interactive performance tests on the core of slicer.

Panels and their use

- **Get Sample Data**
- **Reslicing**: Go into a loop that stresses reslice by calling `sliceNode.SetSliceOffset()`. Average time is logged and time associated with each iteration are stored in a `vtkMRMLTableNode` named `Reslice performance`.
- **Crosshair Jump**: Go into a loop that stresses jumping to slices by moving crosshair using `slicer.util.clickAndDrag()`. Average time is logged.
- **Memory Check**: Run a periodic memory check in a window.

Contributors

- Steve Pieper (Isomics)
- Jean-Christophe Fillion-Robin (Kitware)
- Andras Lasso (PerkLab, Queen's)

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.

9.25.2 Self Tests

Overview

This module provides a built-in self-test (BIST) framework for Slicer modules.

Important features include:

- Tests are available as part of the binary distributions of Slicer, so users can confirm correct behavior on their systems.

- The same tests are run as part of the nightly test process and submitted to the Slicer dashboard.
- Developers can efficiently develop the tests by reloading python scripts without needing to exit Slicer.

The self-test framework can be used in a number of ways:

- When creating a new scripted module using the *Extension Wizard*, a `<moduleName>Test` class deriving from `slicer.ScriptedLoadableModule.ScriptedLoadableModuleTest` is generated. This way you can use the script to help you test the code as you develop it (by reloading and testing as you write the code without even exiting Slicer) and also verify that your code still works as you refactor and improve the code. Plus, you can easily test the code on multiple platforms without a lot of tedious clicking to reload data.
- Self Tests of the core Slicer functionality can be used equally in build-time and run-time scenarios.
- Any type of module or extension can also include self tests (and should!).

Information for developers

See examples and other developer information in *Developer guide*.

Contributors

- Steve Pieper (Isomics)
- Jean-Christophe Fillion-Robin (Kitware)

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.

9.26 Legacy and retired modules

Deprecated modules are not recommended to be used anymore, typically because other modules have replaced them, and they are planned to be removed in the future.

> **Warning**
>
> This module is deprecated and will be removed in the future. It is recommended to use the *Resample Scalar/Vector/DWI Volume* module or the *Crop Volume* module.

9.26.1 Resample Scalar Volume

Overview

Resampling an image is an important task in image analysis. It is especially important in the frame of image registration. This module implements image resampling through the use of itk Transforms. This module uses an Identity Transform. The resampling is controlled by the Output Spacing. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions. Several interpolators are available: linear, nearest neighbor, bspline and five flavors of sinc. The sinc interpolators, although more precise, are much slower than the linear and nearest neighbor interpolator. To resample label volumnes, nearest neighbor interpolation should be used exclusively.

Panels and their use

Resampling Parameters: Parameters used for resampling

- **Spacing** (*outputPixelSpacing*): Spacing along each dimension (0 means use input spacing)
- **Interpolation** (*interpolationType*): Sampling algorithm (linear, nearest neighbor, bspline(cubic) or windowed sinc). There are several sinc algorithms available as described in the following publication: Erik H. W. Meijering, Wiro J. Niessen, Josien P. W. Pluim, Max A. Viergever: Quantitative Comparison of Sinc-Approximating Kernels for Medical Image Interpolation. MICCAI 1999, pp. 210-217. Each window has a radius of 3;

IO: Input/output parameters

- **Input Volume** (*InputVolume*): Input volume to be resampled
- **Output Volume** (*OutputVolume*): Resampled Volume

Contributors

Bill Lorensen (GE)

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.

> **Warning**
>
> This module is deprecated and will be removed in the future. It is recommended to use the *Segment Editor* module.

9.26.2 Robust Statistics Segmenter

Overview

Active contour segmentation using robust statistic.

Panels and their use

Segmentation Parameters: Parameters for robust statistics segmentation

- **Approximate volume(mL)** (*expectedVolume*): The approximate volume of the object, in mL.

Auxiliary Parameters: Some auxiliary parameters to control the stop criteria.

- **Intensity Homogeneity[0-1.0]** (*intensityHomogeneity*): What is the homogeneity of intensity within the object? Given constant intensity at 1.0 score and extreme fluctuating intensity at 0.
- **Boundary Smoothness[0-1.0]** (*curvatureWeight*): Given sphere 1.0 score and extreme rough boundary/surface 0 score, what is the expected smoothness of the object?
- **Output Label Value** (*labelValue*): Label value of the output image
- **Max running time(min)** (*maxRunningTime*): The program will stop if this time is reached.

IO: Input/output parameters

- **Original Image** (*originalImageFileName*): Original image to be segmented
- **Label Image** (*labelImageFileName*): Label image for initialization
- **Output Volume** (*segmentedImageFileName*): Segmented image

Contributors

Yi Gao (gatech), Allen Tannenbaum (gatech), 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

> **Warning**
>
> This module is deprecated and will be removed in the future. It is recommended to use the *Segment Editor* module.

9.26.3 Simple Region Growing Segmentation

Overview

A simple region growing segmentation algorithm based on intensity statistics. To create a list of fiducials (Seeds) for this algorithm, click on the tool bar icon of an arrow pointing to a sphere fiducial to enter the 'place a new object mode' and then use the Markups module. This module uses the Slicer Command Line Interface (CLI) and the ITK filters CurvatureFlowImageFilter and ConfidenceConnectedImageFilter.

Panels and their use

Smoothing Parameters: Parameters to denoise the image prior to segmenting

- **Smoothing iterations** (*smoothingIterations*): Number of smoothing iterations
- **Timestep** (*timestep*): Timestep for curvature flow

Segmentation Parameters: Parameters to prescribe the region growing

- **Number of iterations** (*iterations*): Number of iterations of region growing
- **Multiplier** (*multiplier*): Number of standard deviations to include in intensity model
- **Neighborhood Radius** (*neighborhood*): The radius of the neighborhood over which to calculate intensity model
- **Output Label Value** (*labelvalue*): The integer value (0-255) to use for the segmentation results. This will determine the color of the segmentation that will be generated by the Region growing algorithm
- **Seeds** (*seed*): Seed point(s) for region growing

IO: Input/output parameters

- **Input Volume** (*inputVolume*): Input volume to be filtered
- **Output Volume** (*outputVolume*): Output filtered

Contributors

Jim Miller (GE)

Acknowledgements

This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Retired modules have been already removed from the application.

9.26.4 Retired Modules

The modules listed below have been retired and are no longer present in 3D Slicer. This page exists as a convenient marker for historical details about retired modules.

Retiring modules reduces burden on Slicer developers. Most common reason for retiring a module is that a new, improved module is introduced. A module may also be retired if it no longer works properly (for example due to changes in Slicer core or third-party libraries) and it is not worth investing time into fixing the issue, because the module is not widely used or alternatives exist.

- **Annotations**
  - Retired: 2023-01-02, commit f44849b
  - Reason: Superseded by the *Markups* module.
- **ACPC Transform**
  - Retired: 2023-06-07, commit TBD
  - Reason: This was not a general-purpose module, but one highly specific to neuroimaging. Therefore it was moved to SlicerNeuro extension.
- **Editor**
  - Retired: 2021-11-05, commit 39283db
  - Reason: No longer running properly and superseded by the *Segment Editor* module.
- **Label Statistics**
  - Retired: 2021-11-05, commit cf62bcf
  - Reason: Superseded by the *Segment Statistics* module.
- **Expert Automated Registration**
  - Retired: 2022-02-19, commit 87fd349
  - Reason: No longer running properly. BRAINS, Elastix, and ANTs registration toolkits offer superior registration results, see *Automatic Image Registration* section for details.
- **DataStore**
  - Retired: 2022-04-26, commit fbbac34
  - Reason: Infrastructure for uploading and organizing datasets was based on the obsolete Slicer server based on Midas. Associated datasets are available as release assets associated with the Slicer/SlicerDataStore GitHub project.

CHAPTER

TEN

EXTENSIONS

An extension is a delivery package that bundles one or more Slicer modules. Once installed, the associated modules appear in 3D Slicer just like built-in ones.

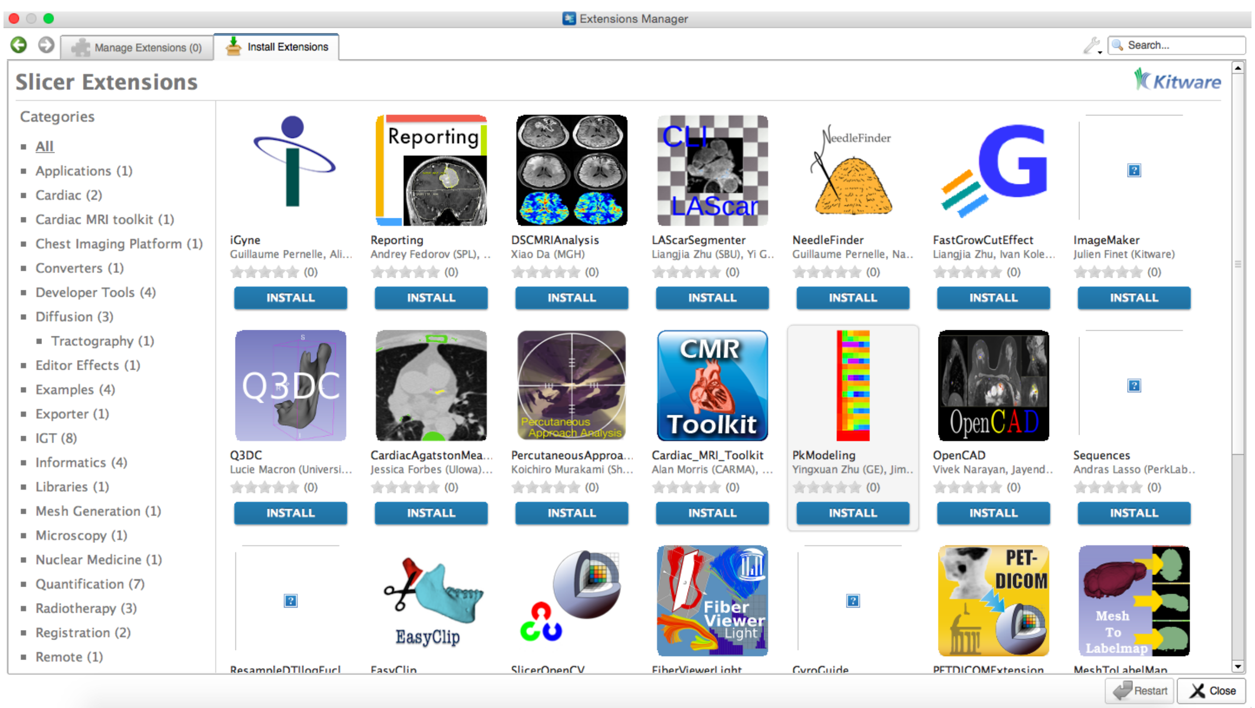
Users can access additional functionality by installing extensions through the Extensions Manager in the 3D Slicer. Alternatively, extension packages can be browsed and downloaded using a web browser from the Slicer Extensions Catalog.

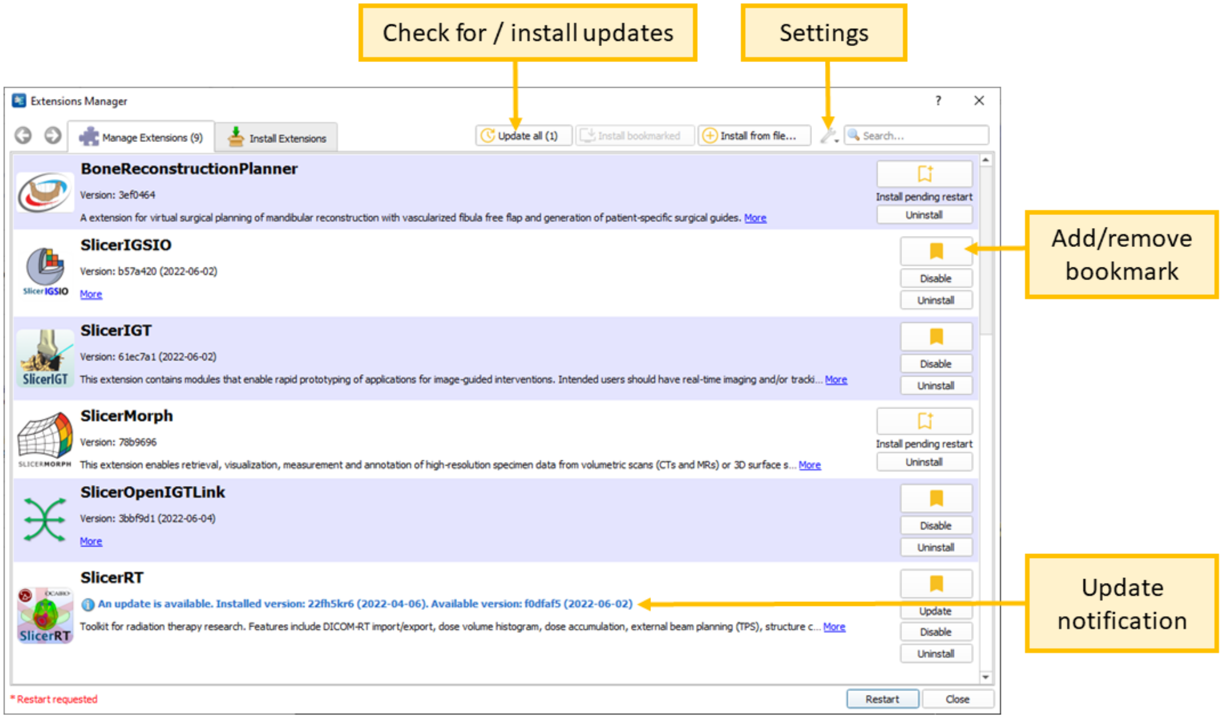
Developers can contribute new capabilities to the platform by creating extensions. For more information, see the *Extension Development Guide*.

10.1 Extensions Manager

10.1.1 Overview

The Extensions Manager is available in the 3D Slicer application. It can be used to find, install, update, and uninstall extensions.





10.1.2 How to

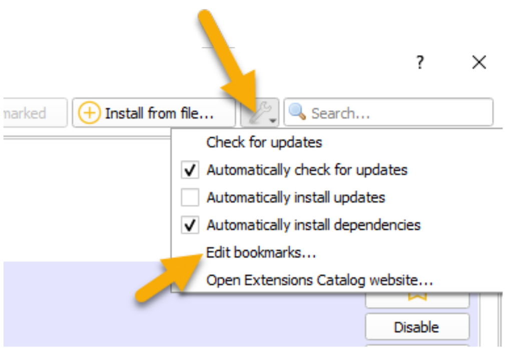
Install extensions

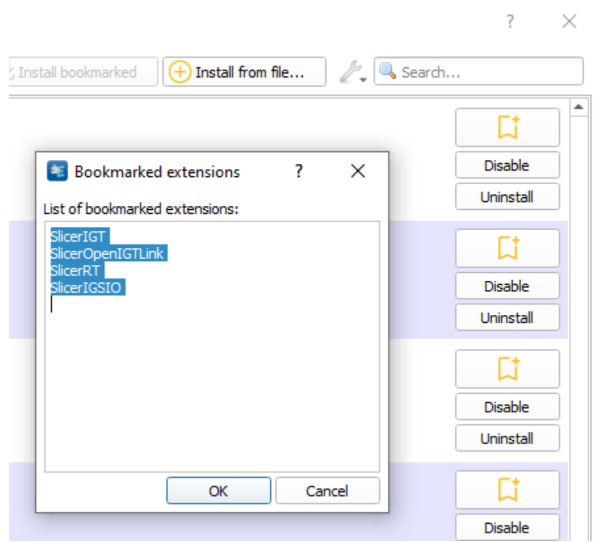
- Open Extensions manager using menu: View / Extensions manager. On macOS, Extensions manager requires the *application to be installed*.
- To install new extensions:
  - Go to “Install extensions” tab
  - Click “Install” button for each extension to be installed.
- To restore previously “bookmarked” extensions:
  - Go to “Manage Extensions” tab
  - Click “Install bookmarked” button to install all bookmarked extensions; or click “Install” button of specific extensions.
- If “Install dependencies” window appears, click “Yes” to install extensions that the selected extension requires. If dependencies are not installed then the chosen extension may not work properly. Automatic installation of dependencies can be enabled by using the “Automatically install dependencies” option in the “Settings” button’s menu.
- Wait until “Restart” button in the lower-right corner becomes enabled, then click “Restart”.
- Click “OK” if you are asked to restart Slicer, unless you have valuable changes in the scene. If there are changes in the scene that needs to be saved then choose “Cancel”, save the scene, and restart the application manually.

Note

Extensions can be "bookmarked" so that they can be easily reinstalled later, even in other Slicer versions. Bookmarks can be added or removed by clicking the orange bookmark icon in the extensions list in the "Manage extensions" tab.

All current bookmarks can be conveniently accessed and multiple bookmarks can be added using "Edit bookmarks…" in the "Settings" button's menu. This feature is also useful to quickly share the list of bookmarked extensions with others, or easily add multiple bookmarked extensions for easy installation.





Note

If the extension that you want to install is disabled and has "Not found for this version of the application" message is displayed below its name then follow instructions in *troubleshooting section*.

Update extensions

Update extensions for Slicer Stable Releases

- Open Extensions manager using menu: View / Extensions manager.
- Go to "Manage extensions" tab.
- Click "Check for updates" button for looking for updates for all installed extensions.
- For each extension that has an available update, "An update is available…" note is displayed.
- Click "Update all" button to install all available updates or click "Update" button for specific extensions.
- Click "Restart" button to restart the application.

Note

By enabling "Automatically check for updates" in the "Settings" button's menu, Slicer will check if extension updates are available at start-up and display a small orange notification marker on the Extensions Manager icon on the toolbar.

By enabling "Automatically install updates" in the "Settings" button's menu, Slicer will automatically install updates when an update is found.

Extensions are updated every night for the latest Slicer Stable Release. Extensions are not updated for earlier Slicer Stable Releases.

Update extensions for Slicer Preview Releases

Each Slicer Preview Release and all corresponding extensions are built at once, and only once (every night, from the latest version of Slicer). To get latest extensions for a Slicer Preview Release, download and install the latest Slicer Preview Release and reinstall the extensions.

Uninstall extensions

- Open Extensions manager using menu: View / Extensions manager
- Go to "Manage extensions" tab.
- Click "Uninstall" button for each extension to be uninstalled.
- Click "Restart" button to restart the application.

Disable extensions

Extensions can be temporarily disabled. Disabled extensions are not loaded into the application but still kept on the system. They can be easily re-enabled, without downloading from the extension manager.

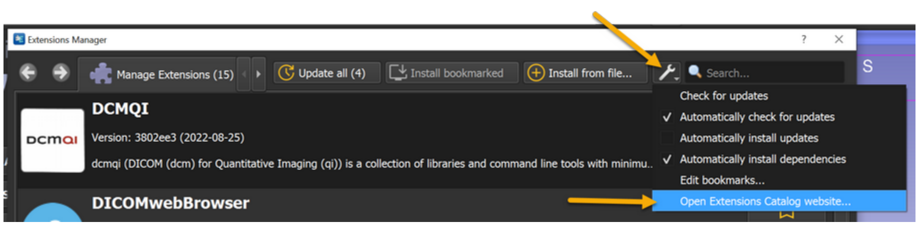
- Open Extensions manager using menu: View / Extensions manager
- Go to "Manage extensions" tab.
- Click "Disable" button for each extension to be uninstalled.
- Click "Restart" button to restart the application.

Install extensions without network connection

Extensions can be downloaded from the Extensions Catalog website and can be installed manually, without network connection, in Slicer using the Extensions manager.

Download extension packages

- Opening the Extension Catalog in the default web browser on your system by clicking on "Open Extensions Catalog website" in the "Settings" button's menu



- Click "Download" button of the selected extension(s) to download the extension package.

Install downloaded extension packages

- Open Extensions manager using menu: View / Extensions manager.
- Click the "Install from file…" button.
- Select the previously downloaded extension package(s). Multiple extension packages can be selected at once.
- Wait for the installations to complete.

- Click "Restart" button to restart the application.

10.1.3 Troubleshooting

Extensions manager takes very long time to start

When starting the extensions manager, the "Extensions manager is starting, please wait…" message is displayed immediately and normally list of extensions should show up within 10-20 seconds. If startup takes longer (several minutes) then most likely the Slicer Extensions Catalog server is temporarily overloaded. Retry in an hour. If the problem persists after 6 hours then report it on the Slicer forum.

Extensions manager does not show any extensions

This can be due to several reasons:

- On macOS: Extensions manager displays the message "Extensions can not be installed" if the application is not installed. See *application installation instructions*.
- Extensions Catalog server is temporarily overloaded (indicated by extensions manager taking several minutes to start and having dozens of `Error retrieving extension metadata` messages in the application log)
  - Recommended action: retry installing extensions an hour later.
- Extensions have not yet been built for the installed Slicer Preview Release (extensions are made available for latest Slicer Preview Release each day at around 12pm EST)
  - Recommended action: wait a few hours until the extension becomes available.
  - Alternative solution A: install Slicer Preview Release created the day before (this is a special link that uses `offset=-1` to request builds from a day before)
  - Alternative solution B: install latest Slicer Stable Release.
- Extensions manager does not have network access
  - Recommended action: Make sure you have internet access. Check your system proxy settings (Slicer uses these system proxy settings by default) and/or set proxy server information in environment variables `http_proxy` and `https_proxy` (more information). On Windows, it may be necessary to re-apply the proxy settings by disabling and re-enabling "Automatically detect settings".
  - Alternative solution: download extension package using a web browser (possibly on a different computer) and install the extension manually. See instructions *here*.
- On macOS: on some older macbooks, the extension manager window appears very bright, washed out (more information is in this issue)
  - Recommended action: setting the operating system to dark mode fixed the issue for several users.

Extension is not found for current Slicer version

For Slicer Stable Releases: If certain extensions are available, but a particular extension is not available for your current Slicer version then contact the maintainer of the extension. If the maintainer cannot be reached then ask for help on the Slicer forum.

For Slicer Preview Releases, due to the constantly updating nature of the preview release, extensions may be missing at times:

- Extensions for latest Slicer Preview Release are uploaded by about 10am EST each day. If you need complete set of extensions then either wait or install previous releases as described *above*.
- Factory system errors: Occasionally, issues with the factory system will prevent some or all extensions from building. See *dashboard* for status information.

- Extension build errors: The extension may not have been updated to work with your Slicer version or a problem may have been introduced recently. Contact the extension's maintainer. If the maintainer cannot be reached then ask for help on the Slicer forum.

Extensions manager is not visible in the menu

If extension manager is not visible then make sure that "Enable extension manager" option is enabled in Application Settings (menu: Edit / Application Settings / Extensions). If you changed the setting, Slicer has to be restarted for it to become effective.

10.1.4 Extensions settings

Settings of extensions manager can be edited in menu: Edit / Application settings / Extensions.

- Extensions manager: if unchecked then Extensions manager is not shown in the menu.
- Extensions server URL: address of the server used to download and install extensions.
- Extensions installation path: directory where extension packages should be extracted and installed.

Modules associated with an extension can also be disabled one by one in menu: Edit / Application settings / Modules.

CHAPTER

ELEVEN

APPLICATION SETTINGS

11.1 Editing application settings

The application settings dialog allows users to customize application behavior.

After starting Slicer, it can be accessed clicking in menu: `Edit` / `Application Settings`.

11.1.1 General

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

11.1.2 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
```

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.

Module folders of extensions are included in this list. To remove modules of an extension, it is recommended to use the *Extensions Manager* instead of just removing its module paths.

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 information. 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.

11.1.3 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 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.

11.1.4 Developer

Developer mode

Enable the following features:

- *Module selection toolbar*: Modules associated with the *Testing* category are visible by default.
- *WebServer module*: Javascript logging is enabled by default.
- *Module panel*: `Reload & Test` module panel section is displayed for scripted modules. It includes controls for reloading, testing and editing scripted modules as well as restarting the application.

modules/webserver.html

11.2 Information for Advanced Users

11.2.1 Settings file location

Settings are stored in *.ini files. If the settings file is found in application home directory (within organization name or domain subfolder) then that .ini file is used. This can be used for creating a portable application that contains all software and settings in a relocatable folder. Relative paths in settings files are resolved using the application home directory, and therefore are portable along with the application.

If .ini file is not found in the application home directory then it is searched in user profile:

- Windows: `%USERPROFILE%\AppData\Roaming\slicer.org\` (typically `C:\Users\<your_user_name>\AppData\Roaming\slicer.org\`)
- Linux: `~/.config/slicer.org/`
- Mac: `~/.config/slicer.org/`

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

There are two types of settings: `user specific settings` and `user and revision specific 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 console:

```
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 console:

```
slicer.app.slicerRevisionUserSettingsFilePath
```

11.2.2 Application startup file

Each time Slicer starts, it will look up for a startup script file named .slicerrc.py. Content of this file is executed automatically at each startup of Slicer.

The file is searched at multiple location and the first one that is found is used. Searched locations:

- Application home folder (`slicer.app.slicerHome`)
- Path defined in `SLICERRC` environment variable

- User profile folder (`~/.slicerrc.py`)

You can find the path to the startup script in Slicer by opening in the menu: Edit / Application Settings. ''Application startup script'' path is shown in the ''General'' section (or running `getSlicerRCFileName()` command in Slicer Python console).

11.2.3 Runtime environment variables

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

- `PYTHONNOUSERSITE`: if it is set to `1` then import of user site packages is disabled. For example, this will prevent Slicer to reuse packages downloaded/built by Anaconda.
- `QT_SCALE_FACTOR`: see Qt documentation. For example, font size can be reduced by running `set QT_SCALE_FACTOR=0.5` in the command console and then starting Slicer in that console.
- `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.
- `SLICERRC`: Custom application startup file path. Contains a full path to a Python script. By default it is `~/.slicerrc.py` (where ~ is the user profile a.k.a user home folder).
- `SLICER_EXTENSIONS_MANAGER_SERVER_URL`: URL of the extensions manager backend with the `/api` path. Default value is retrieved from the settings using the key `Extensions/ServerUrl`.
- `SLICER_EXTENSIONS_MANAGER_FRONTEND_SERVER_URL`: URL of the extension manager frontend displaying the web page. Default value is retrieved from the settings using the key `Extensions/FrontendServerUrl`.
- `SLICER_EXTENSIONS_MANAGER_SERVER_API`: Supported value is `Girder_v1`. Default value is hard-coded to `Girder_v1`.

11.2.4 Qt built-in command-line options

Slicer application accepts standard Qt command-line arguments that specify how Qt interacts with the windowing system.

Examples of options:

- `-qwindowgeometry geometry`, specifies window geometry for the main window using the X11-syntax. For example: `-qwindowgeometry 100x100+50+50`.
- `-display hostname:screen_number`, switches displays on X11 and overrides the `DISPLAY` environment variable.
- `-platform windows:dpiawareness=[0|1|2]`, sets the DPI awareness on Windows.
- `-widgetcount`, prints debug message at the end about number of widgets left undestroyed and maximum number of widgets existed at the same time.
- `-reverse`, sets the application's layout direction to `Qt::RightToLeft`.

To learn about the supported options:

- https://doc.qt.io/qt-5/qapplication.html#QApplication

- https://doc.qt.io/qt-5/qguiapplication.html#supported-command-line-options

Note

Since the Slicer launcher is itself a Qt application and the Qt built-in command-line options are expected to **only** be passed to the launched application `SlicerApp-real` and not the Slicer launcher, the list of arguments to filter is specified in the Main.cpp found in the `commontk/AppLauncher` project.

CHAPTER

TWELVE

DEVELOPER GUIDE

12.1 Slicer API

12.1.1 Tutorials

Check out these developer tutorials to get started with customizing and extending 3D Slicer using Python scripting or C++.

12.1.2 C++

Majority of Slicer core modules and all basic infrastructure are implemented in C++. Documentation of these classes are available at: https://apidocs.slicer.org/main

12.1.3 Python

Native Python documentation

Python-style documentation is available for the following packages:

mrml

This module corresponds to the Python wrapped `MRML` C++ classes.

slicer

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

> **Warning**
>
> These following attributes are only set in the Python environment embedded in the Slicer main application `SlicerApp-real` (launched by the Slicer executable):
>
> - *`app`*
> - *`mrmlScene`*
> - *`modules`*
> - *`moduleNames`*
>
> This means that they are not set in the Python environment of `python-real` (launched by the `PythonSlicer` executable).

`slicer.app`

This is set to the singleton instance of `qSlicerApplication`:

```
>>> slicer.app
qSlicerApplication (qSlicerApplication at: 0x7ffd066a07a0)
```

Warning

This attribute is only set in the Python environment embedded in the Slicer main application.

`slicer.mrmlScene`

This is set to the instance of the MRML Scene:

```
>>> slicer.mrmlScene
<MRMLCore.vtkMRMLScene(0x2797cb0) at 0x7fbfedb82520>

>>> slicer.mrmlScene == slicer.app.mrmlScene()
True
```

Warning

This attribute is only set in the Python environment embedded in the Slicer main application.

`slicer.modules`

This object provides access to all instantiated Slicer modules.

For each instantiated Slicer module, an attribute named after the lower-cased Slicer module name (`slicer.module.<modulename>`) is associated with the corresponding instance of `qSlicerAbstractCoreModule`.

For example:

```
>>> slicer.modules.volumes
qSlicerVolumesModule (qSlicerVolumesModule at: 0x4869000)

>>> slicer.modules.volumes.inherits("qSlicerAbstractModule")
True

>>> slicer.modules.volumes.inherits("qSlicerLoadableModule")
True
```

Warning

These attributes are only set in the Python environment embedded in the Slicer main application.

Additionally for all scripted modules (*qSlicerScriptedLoadableModule*), these additional attributes are also set:

- the attribute `<moduleName>Instance` is set to the corresponding instance of *`ScriptedLoadableModule`*.

  For example:

```
>>> slicer.modules.VectorToScalarVolumeInstance
<VectorToScalarVolume.VectorToScalarVolume object at 0x7fbfc1d1aa60>

>>> isinstance(slicer.modules.VectorToScalarVolumeInstance, slicer.
↪ScriptedLoadableModule.ScriptedLoadableModule)
True
```

- the attribute `<moduleName>Widget` is set to the corresponding instance of *ScriptedLoadableModuleWidget*.

  For example:

```
>>> slicer.util.selectModule(slicer.modules.vectortoscalarvolume)

>>> slicer.modules.VectorToScalarVolumeWidget
<VectorToScalarVolume.VectorToScalarVolumeWidget object at 0x7fbfbc055cd0>

>>> isinstance(slicer.modules.VectorToScalarVolumeWidget, slicer.
↪ScriptedLoadableModule.ScriptedLoadableModuleWidget)
True
```

Note

The `<moduleName>Widget` attribute is:

- Only set after the module has been displayed at least once.
- Updated when reloading the module using *slicer.util.reloadScriptedModule()*.

slicer.**moduleNames**

This object provides access to all instantiated Slicer module names.

The object attributes are the Slicer modules names, the associated value is the module name.

For example:

```
>>> slicer.moduleNames.Volumes
'Volumes'
```

```
>>> slicer.moduleNames.VectorToScalarVolume
'VectorToScalarVolume'
```

Warning

These attributes are only set in the Python environment embedded in the Slicer main application.

slicer.cli

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

slicer.cli.**cancel**(*node*)

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

Creates a new vtkMRMLCommandLineModuleNode for a specific module, with optional parameters slicer.cli.**run**(*module*, *node=None*, *parameters=None*, *wait_for_completion=False*, *delete_temporary_files=True*, *update_display=True*, *hide_window=True*)

Runs a CLI, optionally given a node with optional parameters, returning back the node (or the new one if created) node: existing parameter node (None by default) parameters: dictionary of parameters for cli (None by default) wait_for_completion: block if True (False by default) delete_temporary_files: remove temp files created during execution (True by default) update_display: show output nodes after completion hide_window: whether to hide the CLI process window (Windows only, True by default).

slicer.cli.**runSync**(*module*, *node=None*, *parameters=None*, *delete_temporary_files=True*, *update_display=True*, *hide_window=True*)

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

slicer.cli.**setNodeParameters**(*node*, *parameters*)

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

slicer.logic

slicer.packaging

Utilities for managing Python package dependencies in 3D Slicer.

This module provides functions for checking, installing, and managing Python packages in the Slicer environment. It is the canonical home for all pip-related functionality.

Key functions:

- `load_requirements()` / `load_pyproject_dependencies()` – parse dependency files
- `pip_check()` – check if requirements are satisfied (pure Python, no subprocess)
- `pip_ensure()` – high-level: check, prompt, install, restart detection
- `pip_install()` / `pip_uninstall()` – install/uninstall packages

For backward compatibility, `pip_install()` and `pip_uninstall()` are also available as `slicer.util.pip_install` and `slicer.util.pip_uninstall`.

slicer.packaging.**isPipInstallInProgress**() → bool

Check if a non-blocking pip install is currently in progress.

Use this to check before starting an operation that might conflict with an ongoing pip installation.

Returns

True if a non-blocking pip install is in progress, False otherwise.

Example:

```
if slicer.packaging.isPipInstallInProgress():
    slicer.util.warningDisplay(
        "Package installation is in progress. Please wait."
    )
else:
    slicer.packaging.pip_install("scipy", blocking=False)
```

slicer.packaging.**load_pyproject_dependencies**(*path: str* | *Path*) → list[Requirement]

Load dependencies from a pyproject.toml file.

Reads the `[project.dependencies]` list (PEP 621) and returns `Requirement` objects that can be used with *pip_check()* and *pip_ensure()*.

Only the `dependencies` list inside the `[project]` table is read. Other fields in the table (`name`, `version`, etc.) are not read or validated.

Parameters

path – Path to the pyproject.toml file. Can be a string or Path object.

Returns

List of `packaging.requirements.Requirement` objects. Returns an empty list if `[project]` exists but has no `dependencies` key.

Raises

KeyError – If the file has no `[project]` table.

Example pyproject.toml:

```
[project]
dependencies = [
    "numpy>=1.20",
    "scikit-image>=0.20",
]
```

Example:

```
from pathlib import Path
reqs = slicer.packaging.load_pyproject_dependencies(
    Path(__file__).parent / "pyproject.toml"
)
slicer.packaging.pip_ensure(reqs, requester="MyExtension")
```

slicer.packaging.**load_requirements**(*path: str* | *Path*) → list[Requirement]

Load requirements from a requirements.txt file.

Parses a requirements file and returns a list of Requirement objects that can be used with *pip_check()* and *pip_ensure()*.

Parameters

path – Path to the requirements.txt file. Can be a string or Path object.

Returns

List of `packaging.requirements.Requirement` objects.

Lines starting with # (comments), empty lines, and lines starting with - (pip options like `-r`, `-c`, `--index-url`) are skipped.

Example:

```
from pathlib import Path
reqs = slicer.packaging.load_requirements(Path(__file__).parent / "requirements.txt
↪")
slicer.packaging.pip_ensure(reqs, requester="MyExtension")
```

slicer.packaging.**pip_check**(*req: str* | *Requirement* | *list[str* | *Requirement]*, *_seen: set[tuple[str, frozenset[str]]]* | *None = None*) → bool

Check if requirement(s) are satisfied.

For requirements with extras like `package[extra1,extra2]>=1.0`, this:

1. Checks if the base package is installed at an acceptable version
2. Finds which dependencies are activated by the requested extras
3. Recursively verifies those dependencies are satisfied

Markers (e.g., `; sys_platform == 'win32'`) are evaluated - if a marker doesn't apply to the current environment, the requirement is considered satisfied (since it doesn't need to be installed).

Parameters

- **req** – Requirement(s) to check. Accepts a space-separated string (e.g., `"numpy>=1.20 scipy"`), a single `Requirement`, or a list of strings and/or Requirement objects.
- **_seen** – Internal parameter for tracking circular dependencies. Do not pass this parameter.

Returns

True if all requirements are satisfied, False otherwise.

Example:

```
if slicer.packaging.pip_check("numpy>=1.20"):
    print("numpy is satisfied")
```

`slicer.packaging.`**pip_ensure**(*requirements: str | Requirement | list[str | Requirement]*, *constraints: str | Path | None = None*, *skip_packages: list[str] | None = None*, *prompt_install: bool = True*, *prompt_restart: bool = True*, *requester: str | None = None*, *skip_in_testing: bool = True*, *show_progress: bool = True*) → list[str] | None

Ensure requirements are satisfied, installing if needed.

Call at the point where dependencies are actually needed (e.g., in an `onApplyButton` method). This function checks which requirements are missing and installs them, optionally showing a confirmation dialog first.

After installation, if any updated packages were already imported in the current session, a restart prompt is shown (the old versions remain loaded in memory and may not work correctly until Slicer is restarted).

Parameters

- **requirements** – Requirement(s) to ensure. Accepts a space-separated string (e.g., `"flywheel-sdk>=1.0 numpy"`), a single `Requirement`, or a list of strings and/or Requirement objects. Can also be obtained from *`load_requirements()`* or *`load_pyproject_dependencies()`*.
- **constraints** – Path to a constraints file (string or Path object), or None. When provided, passed to pip as `-c constraints.txt` during installation. Constraints files use the same format as requirements files but only constrain versions without triggering installation. Useful for ensuring compatible versions across multiple extensions.
- **skip_packages** – Discouraged workaround for transitive dependency conflicts; see the script repository for guidance. Forwarded to *`pip_install()`*.
- **prompt_install** – If True (default), show confirmation dialog before installing.
- **prompt_restart** – If True (default), check whether any updated packages were already imported and, if so, show a dialog recommending a restart. The user can choose to restart immediately or continue without restarting.
- **requester** – Name shown in dialog to identify who is requesting the packages (e.g., "TotalSegmentator", "MyFilter", "MyExtension").
- **skip_in_testing** – If True (default), skip installation when Slicer is running in testing mode (`slicer.app.testingEnabled()`). This prevents tests from modifying the Python environment. Set to False if your test explicitly needs to verify installation behavior.

- **show_progress** – If True (default), show progress dialog during installation with status updates and collapsible log details. If False, show only a busy cursor during installation.

Returns

When `skip_packages` is provided, a list of skipped requirement strings (forwarded from *pip_install()*). Otherwise `None`.

Raises

- **RuntimeError** – If user declines installation.
- **subprocess.CalledProcessError** – If installation fails.

Example:

```
class MyFilterWidget(ScriptedLoadableModuleWidget):

    def onApplyButton(self):
        slicer.packaging.pip_ensure("scikit-image>=0.20", requester="MyFilter")
        import skimage

        # Now safe to use skimage
        filtered = skimage.filters.gaussian(array, sigma=2.0)
```

For loading from a requirements file, use *load_requirements()*:

```
reqs = slicer.packaging.load_requirements(self.resourcePath("requirements.txt"))
slicer.packaging.pip_ensure(reqs, requester="MyFilter")
```

For more examples (constraints, skip_packages), see *Script repository* (Python package management section).

`slicer.packaging.`**`pip_install`**(*requirements: str | list[str]*, *constraints: str | Path | None = None*, *no_deps_requirements: str | list[str] | None = None*, *skip_packages: list[str] | None = None*, *blocking: bool = True*, *show_progress: bool = True*, *requester: str | None = None*, *parent: qt.QWidget | None = None*, *logCallback: Callable[[str], None] | None = None*, *completedCallback: Callable[[int], None] | None = None*) → list[str] | None

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 calling pip install directly.

Parameters

- **requirements** – requirement specifier in the same format as used by pip (https://docs.python.org/3/installing/index.html). It can be either a single string or a list of command-line arguments. In general, passing all arguments as a single string is the simplest. The only case when using a list may be easier is when there are arguments that may contain spaces, because each list item is automatically quoted (it is not necessary to put quotes around each string argument that may contain spaces).
- **constraints** – Path to a constraints file (string or Path object), or None. When provided, passed to pip as `-c constraints.txt`. Constraints files use the same format as requirements files but only constrain versions without triggering installation. Useful for ensuring compatible versions across multiple extensions.
- **no_deps_requirements** – Packages to install with `--no-deps` flag, installed before `requirements`. Use this when a package has overly strict dependency requirements that conflict with other packages. Can be a string or list, same format as `requirements`. When provided, installation happens in two steps: first `no_deps_requirements` are

installed with `--no-deps`, then `requirements` are installed normally. Mutually exclusive with `skip_packages`.

- **skip_packages** – Discouraged workaround for transitive dependency conflicts; see the script repository for guidance. Package names to exclude from the dependency tree: installs each requirement with `--no-deps`, walks its dependencies recursively, and skips any package in this list. Metadata is scrubbed so pip won't try to install them later. Name matching is case-insensitive and normalizes hyphens/underscores. Returns a list of the skipped requirement strings. Requires `blocking=True`. Mutually exclusive with `no_deps_requirements`.
- **blocking** – If True (default), block until installation completes and raise CalledProcessError on failure. If False, return immediately and use callbacks. Note: When running in PythonSlicer (without the full application), blocking mode is always used regardless of this setting.
- **show_progress** – If True (default), show visual feedback during installation. When blocking=True, shows a modal progress dialog with status updates and collapsible log details. When blocking=False, shows pip output lines in the status bar as installation progresses. Note: When running in PythonSlicer or in testing mode, progress display is skipped and simple blocking mode is used.
- **requester** – Name shown in dialog/status bar to identify who is requesting the packages (e.g., "MyExtension"). Only used when show_progress=True.
- **parent** – Parent widget for the progress dialog. Only used when show_progress=True and blocking=True.
- **logCallback** – When blocking=False, called with each line of pip output. If None, output is printed to Python console as usual. Signature: `logCallback(line: str) -> None`
- **completedCallback** – When blocking=False, called when pip completes. Receives return code (0 = success). Recommended when blocking=False to know when installation finished. Signature: `completedCallback(returnCode: int) -> None`

Returns

When `skip_packages` is provided, a list of skipped requirement strings (e.g., `["torch>=2.0", "SimpleITK>=2.0.2"]`). Otherwise `None`.

Raises

- **subprocess.CalledProcessError** – In blocking mode, if pip installation fails. When show_progress=True, an error dialog with the full log is shown before raising.
- **ValueError** – If `skip_packages` is combined with `no_deps_requirements` or with `blocking=False`.

> **Warning**
>
> When using `blocking=False`, the user can interact with the application while installation is in progress. Consider disabling relevant UI elements to prevent conflicts.

Example:

```
pip_install("pandas scipy scikit-learn")
```

For more examples (constraints, non-blocking mode, skip_packages, no_deps_requirements), see *Script repository* (Python package management section).

`slicer.packaging.`**`pip_uninstall`**(*requirements: str | list[str]*, *blocking: bool = True*, *logCallback: Callable[[str], None] | None = None*, *completedCallback: Callable[[int], None] | None = None*) → None

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.

Parameters

- **`requirements`** – requirement specifier in the same format as used by pip (https://docs.python.org/3/installing/index.html). It can be either a single string or a list of command-line arguments. It may be simpler to pass command-line arguments as a list if the arguments may contain spaces (because no escaping of the strings with quotes is necessary).
- **`blocking`** – If True (default), block until uninstall completes. If False, return immediately. Note: When running in PythonSlicer (without the full application), blocking mode is always used regardless of this setting.
- **`logCallback`** – When blocking=False, called with each line of pip output.
- **`completedCallback`** – When blocking=False, called when pip completes with return code.

Example: calling from Slicer GUI

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

Example: calling from PythonSlicer console

```
from slicer.packaging import pip_uninstall
pip_uninstall("tensorflow")
```

slicer.ScriptedLoadableModule

class `slicer.ScriptedLoadableModule.`**`ScriptedLoadableModule`**(*parent*)

Bases: `object`

`getDefaultModuleDocumentationLink`(*docPage=None*)

Return string that can be inserted into the application help text that contains link to the module's documentation in current Slicer version's documentation. The text is "For more information see the online documentation." If docPage is not specified then the link points to URL returned by `slicer.app.moduleDocumentationUrl()`.

`resourcePath`(*filename*)

Return the absolute path of the module `Resources` directory.

`runTest`(*msec=100*, ***kwargs*)

Parameters

`msec` – delay to associate with *`ScriptedLoadableModuleTest.delayDisplay()`*.

class `slicer.ScriptedLoadableModule.`**`ScriptedLoadableModuleLogic`**(*parent=None*)

Bases: `object`

`createParameterNode`()

Create a new parameter node The node is of vtkMRMLScriptedModuleNode class. Module name is added as an attribute to allow filtering in node selector widgets (attribute name: ModuleName, attribute value:

the module's name). This method can be overridden in derived classes to create a default parameter node with all parameter values set to their default.

`getAllParameterNodes()`

Return a list of all parameter nodes for this module Multiple parameter nodes are useful for storing multiple parameter sets in a single scene.

`getParameterNode()`

Return the first available parameter node for this module If no parameter nodes are available for this module then a new one is created.

`class slicer.ScriptedLoadableModule.ScriptedLoadableModuleTest(*args, **kwargs)`

Bases: `TestCase`

Base class for module tester class. Setting messageDelay to something small, like 50ms allows faster development time.

`delayDisplay(message, requestedDelay=None, msec=None)`

Display messages to the user/tester during testing.

By default, the delay is 50ms.

The function accepts the keyword arguments `requestedDelay` or `msec`. If both are specified, the value associated with `msec` is used.

This method can be temporarily overridden to allow tests running with longer or shorter message display time.

Displaying a dialog and waiting does two things: 1) it lets the event loop catch up to the state of the test so that rendering and widget updates have all taken place before the test continues and 2) it shows the user/developer/tester the state of the test so that we'll know when it breaks.

Note: Information that might be useful (but not important enough to show to the user) can be logged using logging.info() function (printed to console and application log) or logging.debug() function (printed to application log only). Error messages should be logged by logging.error() function and displayed to user by slicer.util.errorDisplay function.

`runTest()`

Run a default selection of tests here.

`setUp()`

Method called to prepare the test fixture. This is called immediately before calling the test method; other than AssertionError or SkipTest, any exception raised by this method will be considered an error rather than a test failure. The default implementation does nothing.

`takeScreenshot(name, description, type=-1)`

Take a screenshot of the selected viewport and store as and annotation snapshot node. Convenience method for automated testing.

If self.enableScreenshots is False then only a message is displayed but screenshot is not stored. Screenshots are scaled by self.screenshotScaleFactor.

Parameters

- **name** – snapshot node name
- **description** – description of the node
- **type** – which viewport to capture. If not specified then captures the entire window. Valid values: slicer.qMRMLScreenShotDialog.FullLayout, slicer.qMRMLScreenShotDialog.ThreeD, slicer.qMRMLScreenShotDialog.Red, slicer.qMRMLScreenShotDialog.Yellow, slicer.qMRMLScreenShotDialog.Green.

tearDown()

Method called immediately after the test method has been called and the result recorded. This is called even if the test method raised an exception, so the implementation in subclasses may need to be particularly careful about checking internal state. Any exception, other than AssertionError or SkipTest, raised by this method will be considered an additional error rather than a test failure (thus increasing the total number of reported errors). This method will only be called if the setUp() succeeds, regardless of the outcome of the test method. The default implementation does nothing.

class slicer.ScriptedLoadableModule.**ScriptedLoadableModuleWidget**(*parent=None*)

Bases: `object`

cleanup()

Override this function to implement module widget specific cleanup.

It is invoked when the signal *qSlicerModuleManager::moduleAboutToBeUnloaded(QString)* corresponding to the current module is emitted and just before a module is effectively unloaded.

onEditSource()

onReload()

Reload scripted module widget representation.

onReloadAndTest(***kwargs*)

Reload scripted module widget representation and call `ScriptedLoadableModuleTest.runTest()` passing `kwargs`.

onTest(***kwargs*)

Tests scripted module widget, can be used when reload and test doesn't work, calls `ScriptedLoadableModuleTest.runTest()` passing `kwargs`.

resourcePath(*filename*)

Return the absolute path of the module `Resources` directory.

setup()

setupDeveloperSection()

slicer.testing

slicer.testing.**exitFailure**(*message=''*)

slicer.testing.**exitSuccess**()

slicer.testing.**runUnitTest**(*path, testname*)

slicer.util

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**
>
> Not all widget classes are supported yet. Report any missing classes at https://discourse.slicer.org.

Example:

```
class SurfaceToolboxWidget(ScriptedLoadableModuleWidget, VTKObservationMixin):
  ...
  def 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.

Voxels values are deep-copied, therefore if the numpy array is modified after calling this method, voxel values in the volume node will not change.

Parameters

- **narray** – numpy array containing volume voxels.
- **ijkToRAS** – 4x4 numpy array or vtk.vtkMatrix4x4 that defines mapping from IJK to RAS coordinate system (specifying origin, spacing, directions)
- **name** – volume node name
- **nodeClassName** – type of created volume, default: `vtkMRMLScalarVolumeNode`. Use `vtkMRMLLabelMapVolumeNode` for labelmap volume, `vtkMRMLVectorVolumeNode` for vector volume.

Returns

created new volume node

Example:

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

Example:

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

`slicer.util.`**`addVolumeFromITKImage`**(*itkImage*, *name=None*, *nodeClassName=None*, *deepCopy=True*)

Create a new volume node from content of an ITK image and add it to the scene.

By default, voxels values are deep-copied, therefore if the ITK image is modified after calling this method, voxel values in the volume node will not change.

See `updateVolumeFromITKImage()` to understand memory ownership.

Parameters

- **`itkImage`** – ITK image containing volume voxels.
- **`name`** – volume node name
- **`nodeClassName`** – type of created volume, default: `vtkMRMLScalarVolumeNode`. Use `vtkMRMLLabelMapVolumeNode` for labelmap volume, `vtkMRMLVectorVolumeNode` for vector volume.
- **`deepCopy`** – Whether voxels values are deep-copied or not.

Returns

created new volume node

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

Return the array you are "most likely to want" from the indexth

MRML node that matches the pattern.

Raises

`RuntimeError` – if the node cannot be accessed as an array.

> **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 `arrayFromGridTransform()` has been completed.

`slicer.util.`**`arrayFromMarkupsControlPointData`**`(`*markupsNode*, *arrayName*`)`

Return control point data array of a markups 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.`**`arrayFromMarkupsControlPointDataModified`**`(`*markupsNode*, *arrayName*`)`

Indicate that modification of a numpy array returned by `arrayFromMarkupsControlPointData()` 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.`**`arrayFromMarkupsCurveData`**`(`*markupsNode*, *arrayName*, *world=False*`)`

Return curve measurement results from a markups node as a numpy array.

Parameters

- **`markupsNode`** – node to get the curve point data from.
- **`arrayName`** – array name to get (for example *Curvature*)
- **`world`** – if set to True then the point coordinates are returned in world coordinate system (effect of parent transform to the node is applied).

Raises

`ValueError` – in case of failure

> **Warning**
>
> - Not all array may be available in both node and world coordinate systems. For example, *Curvature* is only computed for the curve in world coordinate system.
> - The returned array is not intended to be modified, as arrays are expected to be written only by measurement objects.

`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.`**`arrayFromModelCellData`**`(`*modelNode*, *arrayName*`)`

Return cell 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.`**`arrayFromModelCellDataModified`**`(`*modelNode*, *arrayName*`)`

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

`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 *`arrayFromModelPointData()`* 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()`*.

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

`slicer.util.`**`arrayFromModelPointsModified`**`(`*modelNode*`)`

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

`slicer.util.`**`arrayFromModelPolyIds`**`(`*modelNode*`)`

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

These ids are the following format: [n(0), i(0,0), i(0,1), … i(0,n(00),…, 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.

As described here: https://vtk.org/wp-content/uploads/2015/04/file-formats.pdf

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*`)`

Get segment as numpy array.

> **Warning**
>
> Important: binary labelmap representation may be shared between multiple segments.

Deprecated since version 4.13.0: Use arrayFromSegmentBinaryLabelmap to access a copy of the binary labelmap that will not modify the original labelmap." Use arrayFromSegmentInternalBinaryLabelmap to access a modifiable internal labelmap representation that may be shared" between multiple segments.

`slicer.util.`**`arrayFromSegmentBinaryLabelmap`**`(`*segmentationNode*, *segmentId*, *referenceVolumeNode=None*`)`

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

Parameters

- **`segmentationNode`** – source segmentation node.
- **`segmentId`** – ID of the source segment. Can be determined from segment name by calling `segmentationNode.GetSegmentation().GetSegmentIdBySegmentName(segmentName)`.
- **`referenceVolumeNode`** – a volume node that determines geometry (origin, spacing, axis directions, extents) of the array. If not specified then the volume that was used for setting the segmentation's geometry is used as reference volume.

Raises

`RuntimeError` – in case of failure

Voxels values are copied, therefore changing the returned numpy array has no effect on the source segmentation. The modified array can be written back to the segmentation by calling `updateSegmentBinaryLabelmapFromArray()`.

To get voxels of a segment as a modifiable numpy array, you can use `arrayFromSegmentInternalBinaryLabelmap()`.

`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 source representation then voxel values in the volume node can be modified by changing values in the numpy array. After all modifications has been completed, call:

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

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

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

Return values of a table node's column as numpy array.

Values can be modified by modifying the numpy array. After all modifications has been completed, call `arrayFromTableColumnModified()`.

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

slicer.util.**arrayFromTableColumnModified**(*tableNode*, *columnName*)

Indicate that modification of a numpy array returned by `arrayFromTableColumn()` 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.

Returns

numpy array

Raises

RuntimeError – in case of failure

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.

Raises

RuntimeError – in case of failure

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

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

Return voxel array from volume node as numpy array.

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

Raises

RuntimeError – in case of failure

> **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)
self.ui.inputSelector.setMRMLScene(slicer.mrmlScene)
self.ui.outputSelector.setMRMLScene(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"

Raises

`RuntimeError` – in case of failure

> **Hint**
>
> For generating test data you can use this snippet of code:
>
> ```
> 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 a confirmation popup. Return if confirmed with OK.

When the application is running in testing mode (`slicer.app.testingEnabled() == True`), the popup is skipped and True ("Ok") is returned, with a message being logged to indicate this.

`slicer.util.confirmRetryCloseDisplay(text, windowTitle=None, parent=None, **kwargs)`

Display an error popup asking whether to retry, logging the text at error level. Return if confirmed with Retry.

When the application is running in testing mode (`slicer.app.testingEnabled() == True`), the popup is skipped and False ("Close") is returned, with a message being logged to indicate this.

`slicer.util.confirmYesNoDisplay(text, windowTitle=None, parent=None, **kwargs)`

Display a confirmation popup. Return if confirmed with Yes.

When the application is running in testing mode (`slicer.app.testingEnabled() == True`), the popup is skipped and True ("Yes") is returned, with a message being logged to indicate this.

`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 = createProgressDialog(autoClose=False)

# Update progress value
progressbar.value = 50

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

`slicer.util.dataframeFromMarkups(markupsNode)`

Convert markups node content to pandas dataframe.

Markups content is copied. Therefore, changes in markups node do not affect the dataframe, and dataframe changes do not affect the original markups 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*, *parent=None*, ***kwargs*`)`

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.`**`displayPythonShell`**`(`*display=True*`)`

Show the Python console while the code in the context manager is being run.

The console stays visible only if it was visible already.

Parameters

`display` – If show is False, the context manager has no effect.

```
with slicer.util.displayPythonShell():
  slicer.packaging.pip_install('nibabel')
```

`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.

If there is no main window, or if the application is running in testing mode (`slicer.app.testingEnabled() == True`), then the text is only logged (at error level).

`slicer.util.`**`executePythonModule`**`(`*module: str*, *args: list[str]*, *blocking: bool = True*, *logCallback: Callable[[str], None] | None = None*, *completedCallback: Callable[[int], None] | None = None*`)` → None

Execute a Python module as a script in Slicer's Python environment.

Internally python -m is called with the module name and additional arguments.

Parameters

- **`module`** – Python module name to execute (e.g., "pip")
- **`args`** – command-line arguments to pass to the module
- **`blocking`** – If True (default), block until completion. If False, return immediately.
- **`logCallback`** – When blocking=False, called with each line of output.

- **completedCallback** – When blocking=False, called when process completes.

Raises

- **RuntimeError** – if PythonSlicer executable not found
- **CalledProcessError** – in blocking mode, if process fails

`slicer.util.`**`exit`**`(`*status=0*`)`

Exits the application with the specified exit code.

The method does not stop 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.

..warning:

```
Forcing the application to exit may result in
improperly released files and other resources.
```

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

`slicer.util.`**`exportNode`**`(`*node*, *filename*, *properties={}*, *world=False*`)`

Export 'node' data into 'filename'.

If *world* is set to True then the node will be exported in the world coordinate system (equivalent to hardening the transform before exporting).

This method is different from saveNode in that it does not modify any existing storage node and therefore does not change the filename or filetype that is used when saving the scene.

`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*, *properties=None*`)`

Convenience method to access a widget by its `name` and/or Qt dynamic properties.

Parameters

- **widget** – parent widget to search within.
- **name** – name of the child widget, or `None` to skip name filtering.
- **properties** – optional dict of Qt dynamic property names and expected values. When provided, only children whose properties match all entries are considered.

Raises

RuntimeError – if a matching widget 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.

Parameters

- **widget** – parent widget where the widgets will be searched
- **name** – name attribute of the widget
- **text** – text attribute of the widget
- **title** – title attribute of the widget
- **className** – className() attribute of the widget

Returns

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)`

Get module object from module name.

Returns

module object

Raises

RuntimeError – in case of failure (no such module).

`slicer.util.getModuleGui(module)`

Get module widget.

Deprecated since version 4.13.0: Use the universal `getModuleWidget()` function instead.

`slicer.util.getModuleLogic(module)`

Get module logic object.

Module logic allows a module to use features offered by another module.

Parameters

module – module name or module object

Returns

module logic object

Raises

RuntimeError – if the module does not have widget.

`slicer.util.getModuleWidget(module)`

Return module widget (user interface) object for a module.

Parameters

module – module name or module object

Returns

module widget object

Raises

RuntimeError – if the module does not have widget.

`slicer.util.getNewModuleGui(module)`

Create new module widget.

Deprecated since version 4.13.0: Use the universal `getNewModuleWidget()` function instead.

`slicer.util.getNewModuleWidget(module)`

Create new module widget instance.

In general, not recommended, as module widget may be developed expecting that there is only a single instance of this widget. Instead, of instantiating a complete module GUI, it is recommended to create only selected widgets that are used in the module GUI.

Parameters

module – module name or module object

Returns

module widget object

Raises

RuntimeError – if the module does not have widget.

`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`.

This function is only intended for quick access to nodes for testing and troubleshooting, due to two important limitations. 1. Due to the filename pattern matching uses some special characters (**?[]*) it may be difficult to use it for nodes

that have these characters in their names.

2. In a scene often several nodes have the same name and this function only returns the first one.

If a node name uses special characters, or if there are multiple nodes with the same name, then you can use the node ID as input or use `getFirstNodeByClassByName()` to get the first node of a specific class.

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.getSubjectHierarchyItemChildren`(*parentItem=None*, *recursive=False*)

Convenience method to get children of a subject hierarchy item.

Parameters

- **parentItem** (*vtkIdType*) – Item for which to get children for. If omitted or None then use scene item (i.e. get all items)
- **recursive** (*bool*) – Whether the query is recursive. False by default

Returns

List of child item IDs

`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.

If there is no main window, or if the application is running in testing mode (`slicer.app.testingEnabled() == True`), then the text is only logged (at info level).

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

Return ITK image from volume node.

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

> **Warning**
>
> Important: Memory area of the returned ITK image is managed by VTK (through the `vtkImageData` object stored in the MRML volume node), therefore values in the Voxel values in the ITK image may be changed, but the ITK image must not be reallocated.

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

Indicate that modification of a ITK image returned by `itkImageFromVolume()` (or associated with a volume node using `updateVolumeFromITKImage()`) has been completed.

`slicer.util.launchConsoleProcess`(*args: list[str]*, *useStartupEnvironment: bool = True*, *updateEnvironment: dict[str, str] | None = None*, *cwd: str | None = None*, *blocking: bool = True*, *logCallback: Callable[[str], None] | None = None*, *completedCallback: Callable[[int], None] | None = None*) → Popen[str]

Launch a process. Hiding the console and captures the process output.

The console window is hidden when running on Windows.

Parameters

- **args** – executable name, followed by command-line arguments
- **useStartupEnvironment** – launch the process in the original environment as the original Slicer process

- **updateEnvironment** – map containing optional additional environment variables (existing variables are overwritten)
- **cwd** – current working directory
- **blocking** – If True (default), return the process object for use with *logProcessOutput()*. If False, process output asynchronously using a background thread and QTimer polling. The main thread remains free and the Qt event loop runs normally.
- **logCallback** – When blocking=False, called with each line of output. If None, output is printed to console (same as blocking mode). Signature: `logCallback(line: str) -> None`
- **completedCallback** – When blocking=False, called when process completes. Receives the return code as argument. If the process failed (non-zero return code) and no completedCallback is provided, a CalledProcessError is logged. Signature: `completedCallback(returnCode: int) -> None`

Returns

process object.

Blocking mode example (this is the typical usage pattern):

```
proc = slicer.util.launchConsoleProcess(args)
slicer.util.logProcessOutput(proc)
```

Non-blocking mode example:

```
def onLog(line):
    print(f"[pip] {line}")

def onComplete(returnCode):
    if returnCode == 0:
        print("Done!")
    else:
        print(f"Failed with code {returnCode}")

slicer.util.launchConsoleProcess(
    args,
    blocking=False,
    logCallback=onLog,
    completedCallback=onComplete
)
# Returns immediately, UI stays responsive
```

`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 fiber bundle node from file.

> **Warning**
>
> To ensure the FiberBundleFile reader is registered, the `SlicerDMRI` extension may need to be installed.

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.

Raises

RuntimeError – in case of failure

`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*`)`

Load markups closed curve from file.

Deprecated since version 4.13.0: Use the universal *loadMarkups()* function instead.

`slicer.util.loadMarkupsCurve(`*filename*`)`

Load markups curve from file.

Deprecated since version 4.13.0: Use the universal *loadMarkups()* function instead.

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

Load markups fiducials from file.

Deprecated since version 4.13.0: Use the universal *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=None*, *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. If not specified then the reader with the highest confidence is used.
- **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.

Raises
RuntimeError – in case of failure

`slicer.util.`**`loadNodesFromFile`**(*filename*, *filetype=None*, *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. If not specified then the reader with the highest confidence is used.
- **properties** – map containing additional parameters for the loading.

Returns

loaded node(s) in an iterator object.

Raises

`RuntimeError` – in case of failure

`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*, *properties={}*, *returnNode=False*)

Load node from file.

Parameters

- **filename** – full path of the file to load.
- **properties** – dict object with any of the following keys
  - name: this name will be used as node name for the loaded volume
  - autoOpacities: automatically make large segments semi-transparent to make segments inside more visible (only used when loading segmentation from image file)
  - colorNodeID: use a color node (that already in the scene) to display the image (only used when loading segmentation from image file)
- **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`** – dict object with any of the following keys - 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.

Parameters

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

Returns

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** – dict object with any of the following keys - 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 - colorNodeID: use a color node (that already in the scene) to display the image - 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*, *logCallback=None*`)`

Continuously write process output to the application log and the Python console.

Parameters

- **proc** – process object.
- **logCallback** – optional callback called with each output line (str).

`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*`)`

Loop over all top level widget associated with 'slicer.app' and return the one matching 'objectName'

Raises
RuntimeError – if no top-level widget is found by that name

`slicer.util.`**`mainWindow`**`()`

Get main window widget (qSlicerMainWindow object)

Returns
main window widget, or `None` if there is no main window

`slicer.util.`**`messageBox`**`(`*text*, *parent=None*, ***kwargs*`)`

Displays a messagebox.

ctkMessageBox is used instead of a default qMessageBox to provide "Don't show again" checkbox.

For example:

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

When the application is running in testing mode (`slicer.app.testingEnabled() == True`), an auto-closing popup with a delay of 3s is shown using *`delayDisplay()`* and `qt.QMessageBox.Ok` is returned, with the text being logged to indicate this.

`slicer.util.moduleNames()`

Get list containing name of all successfully loaded modules.

Returns
list of module names

`slicer.util.modulePath(moduleName)`

Return the path where the module was discovered and loaded from.

Parameters
`moduleName` – module name

Returns
file path of the module

`slicer.util.moduleSelector()`

Return module selector widget.

Returns
module widget object

Raises
`RuntimeError` – if there is no module selector (for example, the application runs without a main window).

`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, **kwargs)`

Install python packages.

This is a convenience wrapper that delegates to `slicer.packaging.pip_install()`. See that function for full documentation and parameter details.

Parameters
`requirements` – requirement specifier (string or list), same format as used by pip.

`slicer.util.pip_uninstall(requirements, **kwargs)`

Uninstall python packages.

This is a convenience wrapper that delegates to `slicer.packaging.pip_uninstall()`. See that function for full documentation and parameter details.

Parameters
`requirements` – requirement specifier (string or list), same format as used by pip.

`slicer.util.plot(`*narray*, *xColumnIndex=-1*, *columnNames=None*, *title=None*, *show=True*, *nodes=None*, *plotBoolAsUchar=False*`)`

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. If title is specified for the plot then title+columnName will be used as series name.
- **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.
- **plotBoolAsUchar** – If `True`, any boolean columns in the input array are automatically converted to unsigned integer arrays. This avoids issues with plotting bit arrays (`vtkBitArray`), which are not fully supported by VTK plotting. Defaults to `False`.

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)
```

(continues on next page)

(continued from previous page)

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

slicer.util.**pythonShell**()

Get Python console widget (ctkPythonConsole object)

Raises

RuntimeError – if not found

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 `importlib` 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**()

Restart the application.

No confirmation popup is displayed.

slicer.util.**saveNode**(*node*, *filename*, *properties={}*)

Save 'node' data into 'filename'.

It is the user responsibility to provide the appropriate file extension.

User has also the possibility to overwrite the fileType internally retrieved using method 'qSlicerCoreIOManager::fileWriterFileType(vtkObject*)'. This can be done by specifying a 'fileType'attribute to the optional 'properties' dictionary.

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

Save the current scene.

Based on the value of 'filename', the current scene is saved either as a MRML file, MRB file or directory.

If filename ends with '.mrml', the scene is saved as a single file without associated data.

If filename ends with '.mrb', the scene is saved as a MRML bundle (Zip archive with scene and data files).

In every other case, the scene is saved in the directory specified by 'filename'. Both MRML scene file and data will be written to disk. If needed, directories and sub-directories will be created.

`slicer.util.selectModule(module)`

Set currently active module.

Throws a RuntimeError exception in case of failure (no such module or the application runs without a main window).

Parameters
`module` – module name or object

Raises
`RuntimeError` – in case of failure

`slicer.util.selectedModule()`

Return currently active module.

Returns
module object

Raises
`RuntimeError` – in case of failure (no such module or the application runs without a main window).

`slicer.util.setApplicationLogoVisible(visible=True, scaleFactor=None, icon=None)`

Customize appearance of the application logo at the top of module panel.

Parameters

- `visible` – if True then the logo is displayed, otherwise the area is left empty.
- `scaleFactor` – specifies the displayed size of the icon. 1.0 means original size, larger value means larger displayed size.
- `icon` – a qt.QIcon object specifying what icon to display as application logo.

If there is no main window then the function has no effect.

`slicer.util.setDataProbeVisible(visible)`

Show/hide Data probe at the bottom of module panel.

If there is no main window then the function has no effect.

`slicer.util.setErrorLogVisible(visible)`

Show/hide Error log window.

If there is no main window then the function has no effect.

`slicer.util.setMenuBarsVisible(visible, ignore=None)`

Show/hide all menu bars, except those listed in ignore list.

If there is no main window then the function has no effect.

`slicer.util.setModuleHelpSectionVisible(visible)`

Show/hide Help section at the top of module panel.

If there is no main window then the function has no effect.

`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.

If there is no main window then the function has no effect.

`slicer.util.setPythonConsoleVisible(visible)`

Show/hide Python console.

If there is no main window then the function has no effect.

`slicer.util.setSliceViewerLayers(background='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)

`slicer.util.setStatusBarVisible(visible)`

Show/hide status bar

If there is no main window or status bar then the function has no effect.

`slicer.util.setToolbarsVisible(visible, ignore=None)`

Show/hide all existing toolbars, except those listed in ignore list.

If there is no main window then the function has no effect.

`slicer.util.setViewControllersVisible(visible)`

Show/hide view controller toolbar at the top of slice and 3D views

`slicer.util.settingsValue(key, default, converter=<function <lambda>>, settings=None)`

Return settings value associated with key if it exists or the provided default otherwise.

`settings` parameter is expected to be a valid `qt.Settings` object.

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

Display `message` in the status bar.

`slicer.util.sourceDir()`

Location of the Slicer source directory.

Type

`str` or `None`

This provides the location of the Slicer source directory, if Slicer is being run from a CMake build directory. If the Slicer home directory does not contain a `CMakeCache.txt` (e.g. for an installed Slicer), the property will have the value `None`.

`slicer.util.startQtDesigner(args=None)`

Start Qt Designer application to allow editing UI files.

`slicer.util.startupEnvironment()`

Returns the environment without the Slicer specific values.

Path environment variables like *PATH*, *LD_LIBRARY_PATH* or *PYTHONPATH* will not contain values found in the launcher settings.

Similarly *key=value* environment variables also found in the launcher settings are excluded.

The function excludes both the Slicer launcher settings and the revision specific launcher settings.

> **Warning**
>
> If a value was associated with a key prior starting Slicer, it will not be set in the environment returned by this function.

`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

> **Note**
>
> This directory is not automatically cleaned up.

`slicer.util.toBool`(*value*)

Convert any type of value to a boolean.

The function uses the following heuristic:

1. If the value can be converted to an integer, the integer is then converted to a boolean.
2. If the value is a string, return True if it is equal to 'true'. False otherwise. Note that the comparison is case insensitive.
3. If the value is neither an integer or a string, the bool() function is applied.

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

`slicer.util.toLatin1String`(*text*)

Convert string to latin1 encoding.

`slicer.util.toVTKString`(*text*)

Convert unicode string into VTK string.

Deprecated since version 4.11.0: 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.tryWithErrorDisplay`(*message=None*, *show=True*, *waitCursor=False*)

Show an error display with the error details if an exception is raised.

Parameters

- **message** – Text shown in the message box.
- **show** – If show is False, the context manager has no effect.
- **waitCursor** – If waitCursor is set to True then mouse cursor is changed to wait cursor while the context manager is being run.

```
import random

def risky():
  if random.choice((True, False)):
    raise Exception('Error while trying to do some internal operations.')

with slicer.util.tryWithErrorDisplay("Risky operation failed."):
  risky()
```

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.

Raises

RuntimeError – in case of failure

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 https://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.

See example in *addParameterEditWidgetConnections()* documentation.

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

slicer.util.**updateSegmentBinaryLabelmapFromArray**(*narray*, *segmentationNode*, *segmentId*, *referenceVolumeNode=None*)

Sets binary labelmap representation of a segment from a numpy array.

Parameters

- **narray** – voxel array, containing 0 outside the segment, 1 inside the segment.
- **segmentationNode** – segmentation node that will be updated.
- **segmentId** – ID of the segment that will be updated. Can be determined from segment name by calling `segmentationNode.GetSegmentation().GetSegmentIdBySegmentName(segmentName)`.
- **referenceVolumeNode** – a volume node that determines geometry (origin, spacing, axis directions, extents) of the array. If not specified then the volume that was used for setting the segmentation's geometry is used as reference volume.

Raises

RuntimeError – in case of failure

> **Warning**
>
> Voxels values are deep-copied, therefore if the numpy array is modified after calling this method, segmentation node will not change.

`slicer.util.updateTableFromArray`(*tableNode*, *narrays*, *columnNames=None*, *setBoolAsUchar=False*)

Set values in a table node from a NumPy array or an array-like object (list/tuple of NumPy arrays).

This function can handle:

- **Structured (record) array** with named fields. Each field becomes a separate column. If `columnNames` is not provided, the field names in the dtype are used.
- **1D NumPy array**, which is added as a single column.
- **2D NumPy array**, where each column in the array is added as a separate column in the table node (note that the array is transposed).
- **List/tuple** of 1D NumPy arrays, which are each added as separate columns.

Parameters

- **tableNode** (*vtkMRMLTableNode or None*) – The table node to be updated. If `None`, a new `vtkMRMLTableNode` is created and added to the scene.
- **narrays** (*np.ndarray, tuple, or list*) – NumPy array or array-like object (structured array, 1D/2D array, or list/tuple of 1D arrays).
- **columnNames** (*str, list of str, or None*) – Optional string or list of strings specifying names for the columns. If fewer names are provided than columns, only the specified columns are named; otherwise columns get default names. If `None` is passed, no column names are set.
- **setBoolAsUchar** (*bool*) – If `True`, boolean arrays are converted to `uint8` for plotting compatibility. This leverages the `_vtkArrayFromNumpyArray()` function's handling of `VTK_BIT` arrays. Default is `False`.

Returns

Updated `vtkMRMLTableNode`.

Raises

ValueError – If the input `narrays` is not a recognized format.

This function automatically detects the data type of each input array using `vtk.util.numpy_support.get_vtk_array_type` and maps it to the corresponding VTK array type. As a result, a broader range of numeric and complex data (e.g., int, float, and complex) is supported without requiring manual conversions. All existing columns in the target table node are removed before adding new columns.

> **Warning**
>
> Data in the table node is stored by value (deep copy). Modifying the NumPy array after calling this function does not update the table node's data.

```
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.

Raises

`RuntimeError` – in case of failure

`slicer.util.`**`updateVTKMatrixFromArray`**`(`*vmatrix*, *narray*`)`

Update VTK matrix values from a numpy array.

Parameters

- **`vmatrix`** – VTK matrix (vtkMatrix4x4 or vtkMatrix3x3) that will be update
- **`narray`** – input numpy array

Raises

`RuntimeError` – in case of failure

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

`slicer.util.`**`updateVolumeFromArray`**`(`*volumeNode*, *narray*`)`

Sets voxels of a volume node from a numpy array.

Raises

`RuntimeError` – in case of failure

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 voxel type of the source numpy array does not have to match the current content of the volume node.

`slicer.util.`**`updateVolumeFromITKImage`**`(`*volumeNode*, *itkImage*, *deepCopy=True*`)`

Set voxels of a volume node from an ITK image.

By default, voxels values are deep-copied, therefore if the ITK image is modified after calling this method, voxel values in the volume node will not change.

> **Warning**
>
> Important: Setting *deepCopy* to False means that the memory area is shared between the ITK image and the `vtkImageData` object in the MRML volume node, therefore modifying the ITK image values requires to call *`itkImageFromVolumeModified()`*.
>
> If the ITK image is reallocated, calling this function is required.

`slicer.util.`**`vtkMatrixFromArray`**`(`*narray*`)`

Create VTK matrix from a 3x3 or 4x4 numpy array.

Parameters
`narray` – input numpy array

Raises
`RuntimeError` – in case of failure

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.

If there is no main window, or if the application is running in testing mode (`slicer.app.testingEnabled() == True`), then the text is only logged (at warning level).

vtkTeem

This module corresponds to the Python wrapped `vtkTeem` C++ classes.

vtkAddon

This module corresponds to the Python wrapped `vtkAddon` C++ classes.

vtkITK

This module corresponds to the Python wrapped `vtkITK` C++ classes.

Doxygen-style documentation

Slicer core infrastructure is mostly implemented in C++ and it is made available in Python in the `slicer` namespace. Documentation of these classes is available at: https://apidocs.slicer.org/main/

C++ classes are made available in Python using two mechanisms: PythonQt and VTK Python wrapper. They have a few slight differences:

- Qt classes (class name starts with `q` or `Q`): for example, qSlicerMarkupsPlaceWidget. These classes are all derived from QObject and follow Qt conventions. They support properties, signals, and slots.
- VTK classes (class name starts with `vtk`): for example, vtkMRMLModelDisplayNode. These classes are all derived from vtkObject and follow VTK conventions.

This documentation is generated using the Doxygen tool, which uses C++ syntax. The following rules can help in interpreting this documentation for Python:

- Public member functions: They can be accessed as `objectName.memberFunctionName(arguments)` for example a method of the `slicer.mrmlScene` object can be called as: `slicer.mrmlScene.GetNumberOfNodesByClass('vtkMRMLTransformNode')`. In Qt classes, only methods that have `Q_INVOKABLE` keyword next to them are available from Python. `virtual` and `override` specifiers can be ignored (they just indicate that the method can be or is overridden in a child class).
  - `className` (for Qt classes): constructor, shows the arguments that can be passed when an object is created. Qt objects can be instantiated using `qt.QSomeObject()`. For example `myObj = qt.QComboBox()`.
  - `New` (for VTK classes): constructor, never needs an argument. VTK objects can be instantiated using `vtk.vtkSomeObject()`. For example `myObj = vtk.vtkPolyData()`.
  - `~className`: destructor, can be ignored, Python calls it automatically when needed (when there are no more references to an object). If a variable holds the last reference to an object then it can be deleted by calling `del` or setting the variable to a new value. For example: `del(myObj)` or `myObj = None`.
  - `SafeDownCast` (for VTK classes): not needed for Python, as type conversions are automatic.

- Static Public Member Functions: can be accessed as `slicer.className.memberFunctionName(arguments)` for example: `slicer.vtkMRMLModelDisplayNode.GetSliceDisplayModeAsString(0)`.
- Properties (for Qt classes): these are values that are accessible as object attributes in Python and can be read and written as `objectName.propertyName`. For example:

```
>>> w = slicer.qSlicerMarkupsPlaceWidget()
>>> w.deleteAllControlPointsOptionVisible
True
>>> w.deleteAllControlPointsOptionVisible=False
>>> w.deleteAllControlPointsOptionVisible
False
```

- Public slots (for Qt classes): publicly available methods. Note that `setSomeProperty` methods show up in the documentation but in Python these methods are not available and instead property values can be set using `someProperty = ...`.
- Signals (for Qt classes): signals that can be connected to Python methods

```
def someFunction():
  print("clicked!")

b = qt.QPushButton("MyButton")
b.connect("clicked()", someFunction)  # someFunction will be called when the button␣
↪is clicked
b.show()
```

- Public Types: most commonly used for specifying enumerated values (indicated by `enum` type). These values can be accessed as `slicer.className.typeName`, for example `slicer.qSlicerMarkupsPlaceWidget.HidePlaceMultipleMarkupsOption`
- Protected Slots, Member Functions, Attributes: for internal use only, not accessible in Python.

Mapping commonly used data types from C++ documentation to Python:

- `void` -> Python: if the return value of a method is this type then it means that no value is returned
- `someClass*` (object pointer) -> Python: since Python takes care of reference counting, it can be simply interpreted in Python as `someClass`. The called method can modify the object.
- `int`, `char`, `short` (with optional `signed` or `unsigned` prefix) -> Python: `int`
- `float`, `double` -> Python: `float`
- `double[3]` -> Python: initialize a variable before the method call as `point = np.zeros(3)` (or `point = [0.0, 0.0, 0.0]`) and use it as argument in the function
- `const char *`, `std::string`, `QString`, `const QString&` -> Python: `str`
- `bool` -> Python: `bool`

12.2 MRML Overview

12.2.1 Introduction

Medical Reality Modeling Language (MRML, pronounced “MURml”) 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 that implements MRML data in-memory representation, reading/writing files, visualization, processing framework, and GUI widgets for viewing and editing. The library is based on the VTK toolkit, uses ITK for reading/writing some file formats, and has a few additional optional dependencies, such as Qt for GUI widgets. The library is kept fully independent from 3D Slicer and so it can be used in any other medical applications, but 3D Slicer is the only major application that uses it and therefore MRML library source code is maintained in 3D Slicer's source code repository.
- MRML file: When MRML data is saved to file, an XML document is created (with a .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).

12.2.2 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. The most common event is a "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.

12.2.3 MRML nodes

Basic MRML node types

- **Data nodes** store basic properties of a data set. Because 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. The most important 3D Slicer core data nodes are the following:
  - **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 a volumetric mesh (tetrahedral, wedge elements, unstructured grid, etc.).
  - **Segmentation** (vtkMRMLSegmentationNode): complex data node that can store an image segmentation (also known as contouring, labeling). It can store multiple representations internally; for example it can store both a binary labelmap image and a closed surface mesh.
  - **Markups** (vtkMRMLMarkupsNode and subclasses): stores simple geometrical objects, such as point lists (formerly called "fiducial lists"), lines, angles, curves, planes for annotation and measurements.
  - **Transform** (vtkMRMLTransformNode): stores a 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 for displaying 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 node, each specifying different display properties and view nodes. Built-in 3D Slicer modules typically show and allow editing of only 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 displayed in only 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, a default display node can be added by calling its `CreateDefaultDisplayNodes()` method. In some cases, 3D Slicer detects if the display and storage node are missing and tries to create default nodes, but developers should not rely on this error-recovery mechanism.
- **Storage nodes** (vtkMRMLStorageNode and its subclasses) specify how to store a data node in a file. It can store one or more file names, compression options, coordinate system information, etc.
  - A 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 a 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 displayed and where. In addition to switching between built-in view layouts, custom view layouts can be specified using an XML description.
  - vtkMRMLInteractionNode specifies an interaction mode of viewers (view/transform, window/level, place markups), such as what happens when 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. A plot series node specifies a data series using one or two columns of a table node. A plot chart node specifies which series to plot and how. A plot view node specifies which plot chart to show in a view and how the 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. A sequence browser node specifies which one of the internal data nodes 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 here.

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

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.

MRML Events and Observers

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

12.2.4 Advanced topics

Parameter Nodes

Parameter nodes are a specific use of MRML nodes to store parameters for a given function/module. For more information, see *Parameter Nodes*.

Scene undo/redo

MRML Scene provides an 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, because 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.
- A MRML scene can save a snapshot of all nodes into special Undo and Redo stacks.
- The Undo and Redo stacks store copies of nodes that have changed from the previous snapshot. The nodes that have not changed are stored by a reference (pointer).
- When an Undo is called on the scene, the current state of the Undo stack is copied into the current scene and also into the Redo stack.
- All Undoable operations must store their data as MRML nodes

The developer controls at what point the snapshot is saved by calling the `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 the user interface while dragging a slider (for example vtkKWScale::ScaleValueStartChangingEvent).

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 do not need to be undoable separately).
- `vtkMRMLScene::SetUndoOn()` enables following SaveStateForUndo calls.
- `vtkMRMLScene::ClearUndoStack()` clears the undo history.
- `vtkMRMLScene::ClearRedoStack()` clears the redo history.

Creating Custom MRML Node Classes

If you are adding new functionality to 3D Slicer either via extensions, or even updates to the core, most of the time the existing MRML nodes will be sufficient. Many powerful C++ and Python extensions simply use and combine the existing node types to create new functionality. Instead of creating new MRML nodes from scratch, other extensions subclass from existing nodes and add just a few methods to get the needed functionality. That said, if existing MRML nodes do not offer enough (or almost enough) functionality to enable what needs to be done, it is possible to create custom MRML node classes with a little bit of effort.

There are a number of different MRML nodes and helper classes that can be implemented to enable new MRML data type functionality. Here is the not-so-short list. We will go over each of these in detail.

1. *Data node*
2. *Display node*
3. *Widget*
4. *Widget Representation*
5. *Displayable Manager*
6. *Storage node*

7. *Reader*
8. *Writer*
9. *Subject Hierarchy Plugin*
10. *Module*

While technically not all of these need to be implemented for a new MRML type to be used and useful, implementing all of them will yield the best results. The resulting MRML type will "be like the Model" and will streamline future maintenance work by providing relevant hints.

Note

MRML nodes are implemented in C++.

MRML nodes can be implemented in a 3D Slicer extension.

Note

All links to API class and function documentation redirecting to `https://apidocs.slicer.org` correspond to documentation generated from the latest commit of the `main` branch of 3D Slicer. This means that versions of this documentation associated with an older version of 3D Slicer may be out of sync with the linked API.

Convention

For the filenames and classes, replace `<MyCustomType>` with the name of your type.

The data node

The data node is where the essence of the new MRML type will live. It is where the actual data resides. Notably absent from the data node is any description of how the data should be displayed or stored on disk.

Files:

```
|-- <Extension>
        |-- <Module>
                |-- MRML
                       |-- vtkMRML<MyCustomType>Node.h
                       |-- vtkMRML<MyCustomType>Node.cxx
```

Key points:

- Naming convention for class: `vtkMRML<MyCustomType>Node`
  - E.g. `vtkMRMLModelNode`
- Inherits from `vtkMRMLDisplayableNode` if it is going to be displayed in the 3D or slice views.
- Constructing a new node:
  - Declare `vtkMRMLNode* CreateNodeInstance() override` and `static vtkMRMLYourNodeType* New()`. The implementations will be generated by using the macro `vtkMRMLNodeNewMacro(vtkMRMLYourNodeType);` in your cxx file.
  - Create a protected default constructor.

        * It must be protected because VTK allows its objects to be created through only the `New()` factory function.
        * Because of the use of the `New()` factory function, constructors with parameters are not typically used.
    - Create a destructor if needed.
    - Delete copy/move constructors and copy/move assignment operators.
- To save to a Medical Reality Bundle (MRB) file:
    - Override `const char* GetNodeTagName()` to return a unique XML tag.
    - Override `void ReadXMLAttributes(const char** atts)` and `void WriteXML(ostream& of, int indent)` to save to XML any attributes of the data node that will not be saved by the writer.
- To work with Transforms:
    - Override `bool CanApplyNonLinearTransforms() const` to return true or false depending on whether non-linear transforms can be applied to your data type.
    - Override `void OnTransformNodeReferenceChanged(vtkMRMLTransformNode* transformNode)`. It will be called when a new transform is applied to your data node.
    - Override `void ApplyTransform(vtkAbstractTransform* transform)`, which will be called when a transform is hardened.
    - See bullet on `ProcessMRMLEvents`.
- Override `void GetRASBounds(double bounds[6])` and `void GetBounds(double bounds[6])` to allow the "Center the 3D view on scene" button in the 3D viewer to work. Note that the difference between these functions is that `GetRASBounds` returns the bounds after all transforms have been applied, while `GetBounds` returns the pre-transform bounds.
- Use macro `vtkMRMLCopyContentMacro(vtkMRMLYourNodeType)` in the class definition and implement `void CopyContent(vtkMRMLNode* anode, bool deepCopy)` in the cxx file. This should copy the data content of your node via either a shallow or deep copy.
- Override `vtkMRMLStorageNode* CreateDefaultStorageNode()` to return an owning pointer default storage node type for your class (see *The storage node*).
- Override `void CreateDefaultDisplayNodes()` to create the default display nodes (for 3D and/or 2D viewing).
- Override `void ProcessMRMLEvents(vtkObject* caller, unsigned long event, void* callData)`:
    - This is used to process any events that happen regarding this object.
    - This method should handle the `vtkMRMLTransformableNode::TransformModifiedEvent` which is emitted any time the transform that is associated with the data object is changed.
- Convenience methods - while not necessarily needed, they are nice to have.
    - `Get<MyCustomType>DisplayNode()` function that returns the downcast version of `GetDisplayNode()` saves users of your class a bit of downcasting.
- Other methods:
    - Add other methods as your heart desires to view/modify the actual content of the data being stored.

Tip

Any methods with signatures that contain only primitives, raw pointers to VTK derived objects, or a few std library items like `std::vector` will be automatically wrapped for use in Python. Any functions signatures that contain other classes (custom classes, smart pointers from the std library, etc) will not be wrapped. For best results, try to use existing VTK data objects, or have your custom classes derive from `vtkObject` to get automatic wrapping.

The display node

The display node, contrary to what one may think, is not actually how a MRML object is displayed on screen. Instead it is the list of options for displaying a MRML object. Things like colors, visibility, opacity; all of these can be found in the display node.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- MRML
                 |-- vtkMRML<MyCustomType>DisplayNode.h
                 |-- vtkMRML<MyCustomType>DisplayNode.cxx
```

Key Points:

- Naming convention for class: `vtkMRML<MyCustomType>DisplayNode`
  - E.g. `vtkMRMLModelDisplayNode`
- Inherits from `vtkMRMLDisplayNode`.
- Constructing a new node is same as the *data node*.
- To save to an MRB is same as for the *data node*:
  - Some MRML types like Markups store display information when the actual data is being stored via the writer/storage node. If you do that, no action is needed in this class.
- Convenience methods:
  - `Get<YourDataType>Node()` function that returns a downcasted version of `vtkMRMLDisplayableNode* GetDisplayableNode()`.
- Other methods:
  - Add any methods regarding coloring/sizing/displaying your data node.

The widget

The widget is interaction half of actually putting a usable object in the 2D or 3D viewer. It is in charge of making the widget representation and interacting with it.

Note

If your MRML node is display only without any interaction from the viewers, the widget is not necessary, just the *widget representation* for displaying.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- VTKWidgets
                  |-- vtkSlicer<MyCustomType>Widget.h
                  |-- vtkSlicer<MyCustomType>Widget.cxx
```

Key points:

- Naming convention for class: `vtkSlicer<MyCustomType>Widget`
- Inherits from `vtkMRMLAbstractWidget`.
- Constructing a new node is same as the *data node*.
- For viewing:
  - Add function(s) to create *widget representation*(s). These will typically take a display node, a view node, and a vtkRenderer.
  - E.g. `void CreateDefaultRepresentation(vtkMRML<MyCustomType>DisplayNode* myCustomTypeDisplayNode, vtkMRMLAbstractViewNode* viewNode, vtkRenderer* renderer);`
- For interaction override some or all of the following methods from vtkMRMLAbstractWidget:
  - `bool CanProcessInteractionEvent(vtkMRMLInteractionEventData* eventData, double& distance2)`
  - `bool ProcessInteractionEvent(vtkMRMLInteractionEventData* eventData)`
  - `void Leave(vtkMRMLInteractionEventData* eventData)`
  - `bool GetInteractive()`
  - `int GetMouseCursor()`

The widget representation

The widget representation is the visualization half of displaying a node on screen. This is where any data structures describing your type are turned into `vtkActor`s that can be displayed in a VTK render window.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- VTKWidgets
                  |-- vtkSlicer<MyCustomType>WidgetRepresentation.h
                  |-- vtkSlicer<MyCustomType>WidgetRepresentation.cxx
```

Key Points:

- Naming convention for class: `vtkSlicer<MyCustomType>WidgetRepresentation`
- Inherits from `vtkMRMLAbstractWidgetRepresentation`.
- Constructing a new node is same as the *data node*.
- Override `void UpdateFromMRML(vtkMRMLNode* caller, unsigned long event, void* callData = nullptr)` to update the widget representation when the underlying data or display nodes change.
- To make the class behave like a vtkProp override:
  - `void GetActors(vtkPropCollection*)`

  - `void ReleaseGraphicsResources(vtkWindow*)`
  - `int RenderOverlay(vtkViewport* viewport)`
  - `int RenderOpaqueGeometry(vtkViewport* viewport)`
  - `int RenderTranslucentPolygonalGeometry(vtkViewport* viewport)`
  - `vtkTypeBool HasTranslucentPolygonalGeometry()`

Important

The points/lines/etc pulled from the data node should be post-transform, if there are any transforms applied.

Tip

Minimize the number of actors used for better rendering performance.

The displayable manager

The data node, display node, widget, and widget representation are all needed pieces for data actually showing up on the screen. The displayable manager is the glue that brings all the pieces together. It monitors the MRML scene, and when data and display nodes are added or removed, it creates or destroys the corresponding widgets and widget representations.

Files:

```
|-- <Extension>
        |-- <Module>
                |-- MRMLDM
                       |-- vtkMRML<MyCustomType>DisplayableManager.h
                       |-- vtkMRML<MyCustomType>DisplayableManager.cxx
```

Key Points:

- Naming convention for class: `vtkSlicer<MyCustomType>DisplayableManager`
- Inherits from `vtkMRMLAbstractDisplayableManager`.
- Constructing a new node is same as the *data node*.
- Override `void OnMRMLSceneNodeAdded(vtkMRMLNode* node)` to watch for if a new node of your type is added to the scene. Add an appropriate widget(s) and widget representation(s) for any display nodes.
- Override `void OnMRMLSceneNodeRemoved(vtkMRMLNode* node)` to watch for if a node of your type is removed from the scene. Remove corresponding widget(s) and widget representation(s).
- Override `void OnMRMLSceneEndImport()` to watch for an MRB file that has finished importing.
- Override `void OnMRMLSceneEndClose()` to clean up when a scene closes.
- Override `void ProcessMRMLNodesEvents(vtkObject* caller, unsigned long event, void* callData)` to watch for changes in the data node that would require the display to change.
- Override `void UpdateFromMRML()` and `void UpdateFromMRMLScene()` to bring the displayable manager in line with the MRML Scene.

The storage node

A storage node is responsible for reading and writing data nodes to files. A single data node type can have multiple storage node types associated with it for reading/writing different formats. A storage node will be created for both normal save/load operations for a single data node, as well as when you are saving a whole scene to an MRB.

It is common for a data node's storage node to also write relevant values out of the display node (colors, opacity, etc) at the same time it writes the data.

> **Note**
>
> The storage node is not sufficient in itself to allow the new data node to be saved/loaded from the normal 3D Slicer save/load facilities; the *reader* and *writer* will help with that.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- MRML
                 |-- vtkMRML<MyCustomType>StorageNode.h
                 |-- vtkMRML<MyCustomType>StorageNode.cxx
```

Key Points:

- Naming convention for class: `vtkMRML<MyCustomType>StorageNode`
  - If you have multiple storage nodes you may have other information in the name, such as the format that is written. E.g. `vtkMRMLMarkupsJSONStorageNode`.
- Inherits from `vtkMRMLStorageNode`.
- Constructing a new node is same as the *data node*.
- Override `bool CanReadInReferenceNode(vtkMRMLNode* refNode)` to allow a user to inquire at runtime if a particular node can be read in by this storage node.
- Override protected `void InitializeSupportedReadFileTypes()` to show what file types and extensions this storage node can read (can be more than one).
- Override protected `void InitializeSupportedWriteFileTypes()` to show what types and extensions this storage node can read (can be more than one).
  - It is recommended to be able to read and write the same file types within a single storage node.
- Override protected `int ReadDataInternal(vtkMRMLNode* refNode)`:
  - This is called by the public `ReadData` method.
  - This is where the actually reading data from a file happens.
- Override protected `int WriteDataInternal(vtkMRMLNode* refNode)`:
  - This is called by the public `WriteData` method.
  - This is where the actually writing data to a file happens.
- If your data node uses any coordinates (most nodes that get displayed should) it is recommended to be specific in your storage format whether the saved coordinates are RAS or LPS coordinates.
  - Having a way to allow the user to specify this is even better.
- Other methods

  - Adding a `vtkMRML<MyCustomType>Node* Create<MyCustomType>Node(const char* nodeName)` function will be convenient for implementing the writer and is also convenient for users of the storage node.

> **Tip**
>
> If your storage node reads/writes JSON, RapidJSON is already in the superbuild and is the recommended JSON parser.
>
> It is recommended to have your extension be `.<something>.json` where the `<something>` is related to your node type (e.g. `.mrk.json` for Markups).

The reader

The recommended way to read a file into a MRML node is through the storage node. The reader, on the other hand, exists to interface with the loading facilities of 3D Slicer (drag and drop, as well as the button to load data into the scene). As such, the reader uses the storage node in its implementation.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- qSlicer<MyCustomType>Reader.h
            |-- qSlicer<MyCustomType>Reader.cxx
```

Key Points:

- Naming convention for class: `qSlicer<MyCustomType>Reader`
- Inherits from `qSlicerFileReader`.
- In the class definition, the following macros should be used:
  - `Q_OBJECT`
  - `Q_DECLARE_PRIVATE`
  - `Q_DISABLE_COPY`
- Constructing a new node:
  - Create constructor `qSlicer<MyCustomType>Reader(QObject* parent = nullptr)`.
    * This constructor, even if it is not explicitly used, allows this file to be wrapped in Python.
- Override `QString description() const` to provide a short description on the types of files read.
- Override `IOFileType fileType() const` to give a string to associate with the types of files read.
  - This string can be used in conjunction with the python method `slicer.util.loadNodeFromFile`
- Override `QStringList extensions() const` to provide the extensions that can be read.
  - Should be the same as the storage node because the reader uses the storage node.
- Override `bool load(const IOProperties& properties)`. This is the function that actually loads the node from the file into the scene.

> **Important**
>
> The reader is not a VTK object, like the previous objects discussed. It is actually a QObject, so we follow Qt guidelines. One such guideline is the D-Pointer pattern, which is recommended for use.

The writer

The writer is the companion to the reader, so, similar to the reader, it does not implement the actual writing of files, but rather it uses the storage node. Its existence is necessary to use 3D Slicer's built in saving facilities, such as the save button.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- qSlicer<MyCustomType>Writer.h
            |-- qSlicer<MyCustomType>Writer.cxx
```

Key points:

- Naming convention for class: `qSlicer<MyCustomType>Writer`
- Inherits from `qSlicerNodeWriter`.
- See the *reader* for information on defining and constructing Qt style classes.
- Override `QStringList extensions(vtkObject* object) const` to provide file extensions that can be written to.
  - File extensions may be different, but don't have to be, for different data nodes that in the same hierarchy (e.g. Markups Curve and Plane could reasonably require different file extensions, but they don't).
- Override `bool write(const qSlicerIO::IOProperties& properties)` to do the actual writing (by way of a storage node, of course).

The subject hierarchy plugin

A convenient module in 3D Slicer is the Data module. It brings all the different data types together under one roof and offers operations such as cloning, deleting, and renaming nodes that work regardless of the node type. The Data module uses the Subject Hierarchy, which is what we need to plug into so our new node type can be seen in and modified by the Data module.

Files:

```
|-- <Extension>
      |-- <Module>
            |-- SubjectHierarchyPlugins
                  |-- qSlicerSubjectHierarchy<MyCustomType>Plugin.h
                  |-- qSlicerSubjectHierarchy<MyCustomType>Plugin.cxx
```

Key Points:

- Naming convention for class: `qSlicerSubjectHierarchy<MyCustomType>Plugin`
- Inherits from `qSlicerSubjectHierarchyAbstractPlugin`.
- See the *reader* for information on defining and constructing Qt style classes.

- Override `double canAddNodeToSubjectHierarchy(vtkMRMLNode* node, vtkIdType parentItemID=vtkMRMLSubjectHierarchyNode::INVALID_ITEM_ID) const`.
  – This method is used to determine if a data node can be placed in the hierarchy using this plugin.
- Override `double canOwnSubjectHierarchyItem(vtkIdType itemID) const` to say if this plugin can own a particular subject hierarchy item.
- Override `const QString roleForPlugin() const` to give the plugin's role (most often meaning the data type the plugin can handle, e.g. Markup).
- Override `QIcon icon(vtkIdType itemID)` and `QIcon visibilityIcon(int visible)` to set icons for your node type.
- Override `QString tooltip(vtkIdType itemID) const` to set a tool tip for your node type.
- Override the following to determine what happens when a user gets/sets the node color through the subject hierarchy:
  – `void setDisplayColor(vtkIdType itemID, QColor color, QMap<int, QVariant> terminologyMetaData)`
  – `QColor getDisplayColor(vtkIdType itemID, QMap<int, QVariant> &terminologyMetaData) const`

The module (aka putting it all together)

If you have used 3D Slicer for any length of time, you have probably noticed that for each type of node (or set of types as in something like markups) there is a dedicated module that is used solely for interacting with the single node type (or set of types). Examples would be the Models, Volumes, and Markups modules. These modules are useful from a user perspective and also necessary to get your new node registered everywhere it needs to be.

As these are normal 3D Slicer modules, they come in three main parts, the module, the logic, and the module widget. The recommended way to create a new module is through the *Extension Wizard*.

Files:

```
|-- <Extension>
       |-- <Module>
              |-- qSlicer<MyCustomType>Module.h
              |-- qSlicer<MyCustomType>Module.cxx
              |-- qSlicer<MyCustomType>ModuleWidget.h
              |-- qSlicer<MyCustomType>ModuleWidget.cxx
              |-- Logic
                    |-- vtkSlicer<MyCustomType>Logic.h
                    |-- vtkSlicer<MyCustomType>Logic.cxx
```

In `qSlicer<MyCustomType>Module.cxx`:

- Override the `void setup()` function:
  – Register your displayable manager with the `vtkMRMLThreeDViewDisplayableManagerFactory` and/or the `vtkMRMLSliceViewDisplayableManagerFactory`.
  – Register your subject hierarchy plugin with the `qSlicerSubjectHierarchyPluginHandler`.
  – Register your reader and writer with the `qSlicerIOManager`.
- Override the `QStringList associatedNodeTypes() const` function and return all the new MRML classes created (data, display, and storage nodes).

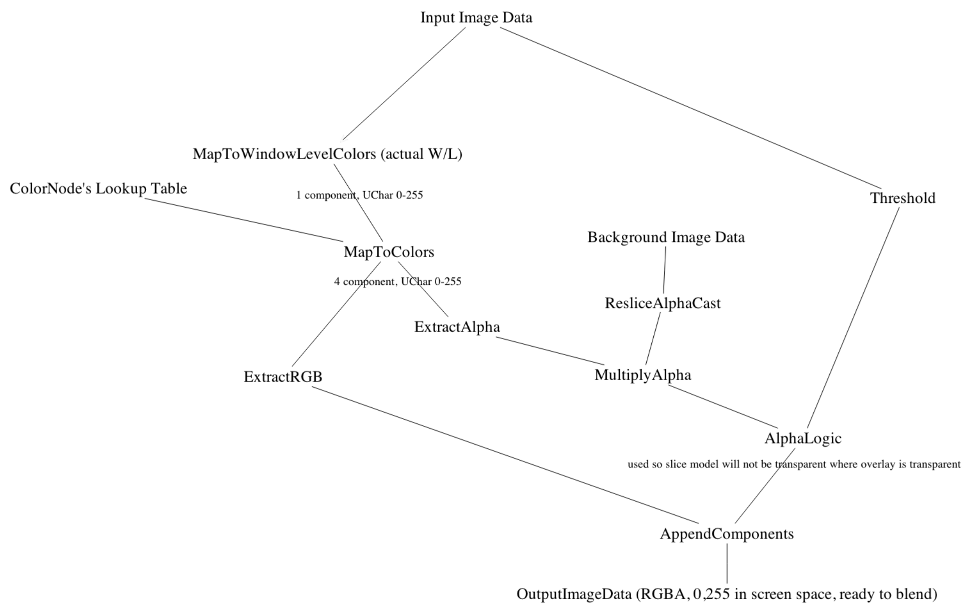
In `vtkSlicer<MyCustomType>Logic.cxx`:

- Override the protected `void RegisterNodes()` function and register all the new MRML classes created (data, display, and storage nodes) with the MRML scene.

In `qSlicer<MyCustomType>ModuleWidget.cxx`:

- Override `bool setEditedNode(vtkMRMLNode* node, QString role = QString(), QString context = QString())` and `double nodeEditable(vtkMRMLNode* node)` if you want this module to be connected to the Data module's "Edit properties…" option in the right click menu.

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.

Management of slice views is distributed between several objects:

- Slice node (vtkMRMLSliceNode): Stores the position, orientation, and size of the slice. This is a view node and as such it stores common view properties, such as the view name, layout color, background color.
- Slice display node (vtkMRMLSliceDisplayNode): Specifies how the slice should be displayed, such as visibility and style of display of intersecting slices. The class is based on classvtkMRMLModelDisplayNode, therefore it also specifies which 3D views the slice is displayed in.
- Slice composite node (vtkMRMLSliceCompositeNode): Specifies what volumes are displayed in the slice and how to blend them. It is ended up being a separate node probably because it is somewhat a mix between a data node (that tells what data to display) and a display node (that specifies how the data is displayed).
- Slice model node (vtkMRMLModelNode): It is a model node that displays the slice in 3D views. It stores a simple rectangle mesh on that the image cross-section is rendered as a texture.
- Slice model transform node (vtkMRMLTransformNode). The transform node is used for positioning the slice model node in 3D views. It is faster to use a transform node to position the plane than modifying the plane points each time the slice is moved.
- Slice logic (vtkMRMLSliceLogic): This is not a MRML node but a logic class, which implements operationson MRML nodes. There is one logic for each slice view. The object keeps reference to all MRML nodes, so

it is a good starting point for accessing any data related to slices and performing operations on slices. Slice logics are stored in the application logic object and can be retrieved like this: `sliceLogic = slicer.app.applicationLogic().GetSliceLogicByLayoutName('Red')`. There are a few other `GetSlicerLogic...` methods to get slice logic based on slice node, slice model display node, and to get all the slice logics.

- Slice layer logic(vtkMRMLSliceLayerLogic): Implements reslicing and interpolation for a volume. There is one slice layer logic for each volume layer (foreground, background, label) for each slice view.
- Slice link logic (vtkMRMLSliceLinkLogic): There is only a single instance of this object in the entire application. This object synchronizes slice view property changes between all slice views in the same view group.

Layout

A layout manager (qSlicerLayoutManager) shows or hides layouts:

- It instantiates, shows or hides relevant view widgets.
- It is associated with a vtkMRMLLayoutNode describing the current layout configuration and ensuring it can be saved and restored.
- It owns an instance of vtkMRMLLayoutLogic that controls the layout node and the view nodes in a MRML scene.
- Pre-defined layouts are described using XML and are registered in `vtkMRMLLayoutLogic::AddDefaultLayouts()`.
- Developer may register additional layout.

Registering a custom layout

See *example in the script repository*.

Layout XML Format

Layout description may be validated using the following DTD:

```
<!DOCTYPE layout SYSTEM "https://slicer.org/layout.dtd"
[
<!ELEMENT layout (item+)>
<!ELEMENT item (layout*, view)>
<!ELEMENT view (property*)>
<!ELEMENT property (#PCDATA)>

<!ATTLIST layout
type (horizontal|grid|tab|vertical) #IMPLIED "horizontal"
split (true|false) #IMPLIED "true" >

<!ATTLIST item
multiple (true|false) #IMPLIED "false"
splitSize CDATA #IMPLIED "0"
row CDATA #IMPLIED "0"
column CDATA #IMPLIED "0"
rowspan CDATA #IMPLIED "1"
colspan CDATA #IMPLIED "1"
>

<!ATTLIST view
class CDATA #REQUIRED
```

(continues on next page)

(continued from previous page)

```
singletontag CDATA #IMPLIED
horizontalStretch CDATA #IMPLIED "-1"
verticalStretch CDATA #IMPLIED "-1" >

<!ATTLIST property
name CDATA #REQUIRED
action (default|relayout) #REQUIRED >

]>
```

Notes:

- `layout` element:
  - `split` attribute applies only to layout of type `horizontal` and `vertical`
- `item` element:
  - `row`, `column`, `rowspan` and `colspan` attributes applies only to layout of type `grid`
  - `splitSize` must be specified only for `layout` element with `split` attribute set to `true`
- `view` element:
  - `class` must correspond to a MRML view node class name (e.g `vtkMRMLViewNode`, `vtkMRMLSliceNode` or `vtkMRMLPlotViewNode`)
  - `singletontag` must always be specified when `multiple` attribute of `item` element is specified.
- `property` element:
  - `name` attribute may be set to the following values:
    * `viewlabel`
    * `viewcolor`
    * `viewgroup`
    * `orientation` applies only if parent `view` element is associated with `class` (or subclass) of type `vtkMRMLSliceNode`

12.3 Module Overview

Slicer supports multiple types of modules:

- *Command Line Interface (CLI)*
- *Loadable*
- *Scripted*

The choice for a given type of module is usually based on the type of inputs/parameters for a given module.

12.3.1 Command Line Interface (CLI)

CLIs are standalone executables with a limited input/output arguments complexity (simple argument types, no user interactions…).

> **Hint**
>
> Recommended for implementing computational algorithms.

Specifications:

- CLI = Command-line interface
- Slicer can run any command-line application from the GUI (by providing an interface description `.xml` file).
- Can be implemented in any language (C++, Python, …).
- Inputs and outputs specified in XML file, GUI is generated automatically.
- Parameters passed through command line and files.
- Run in a separate processing thread, can report progress and be aborted.

Not supported (anti-patterns):

- Pass back intermediate results.
- Update the *views* while executing.
- Accept input while running to steer the module.
- Request input while running.

> **Getting started**
>
> - For CLI, *build Slicer* and create an initial skeleton using the *Extension Wizard* adding a module of type `cli`.
> - For Scripted CLI, create an initial skeleton using the *Extension Wizard* adding a module of type `scriptedcli`.

More information:

- *Developing and contributing extensions for 3D Slicer*
- Tutorials for software developers
- Slicer execution model: Describe the mechanism for incorporating command line programs as Slicer modules.
- Learn from existing Slicer CLI modules.

12.3.2 Loadable Modules

Loadable modules are C++ plugins that are built against Slicer. They define custom GUIs for their specific behavior as they have full control over the application.

> **Hint**
>
> Recommended for implementing complex, performance-critical, interactive components, application infrastructure (e.g., reusable of low-level GUI widgets).

Specifications:

- Written in C++ and distributed as shared libraries.
- Full *Slicer API* is accessible.

- Full control over the Slicer UI (based on Qt) and Slicer internals (*MRML*, logics, display managers…).

Getting started

Build Slicer and create an initial skeleton using the *Extension Wizard* adding a module of type `loadable`.

More information:

- *Developing and contributing extensions for 3D Slicer*
- Tutorials for software developers
- Learn from existing Slicer loadable modules.

12.3.3 Scripted Modules

Scripted modules are Python scripts that uses Slicer API. They define custom GUIs for their specific behavior as they have full control over the application.

Hint

Recommended for fast prototyping and custom workflow development.

Specifications:

- Written in Python.
- Full *Slicer API* is accessible.
- Full access to the API of *MRML*, VTK, Qt, ITK and SimpleITK since those are Python wrapped.

Getting started

Download Slicer and create an initial skeleton using the *Extension Wizard* adding a module of type `scripted`.

More information:

- *Developing and contributing extensions for 3D Slicer*
- Tutorials for software developers
- *Python FAQ*
- *Script repository*
- Learn from existing Slicer scripted modules.

12.3.4 Module Factory

Loading modules into slicer happens in multiple steps:

- Module factories must be registered into the factory manager.
- Directories where the modules to load are located must be passed to the factory manager.
- Optionally specify module names to ignore.
- Scan the directories and test which file is a module and register it (not instantiated yet).

- Instantiate all the register modules.
- Connect each module with the scene and the application.

More details can be found in the online doc

12.3.5 Association of MRML nodes to modules

Modules can be associated with MRML nodes, which for example allows determining what module can be used to edit a certain MRML node. A module can either specify the list of node types that it supports by overriding qSlicerAbstractCoreModule::associatedNodeTypes() method or a module can call qSlicerCoreApplication::addModuleAssociatedNodeTypes() to associate any node type with any module.

Multiple modules can be associated with the same MRML node type. The best module for editing a specific node instance is determined run-time. The application framework calls qSlicerAbstractModuleWidget::nodeEditable() for each associated module candidate and will activate the one that has the highest confidence in handling the node.

To select a MRML node as the "active" or "edited" node in a module the module widget's qSlicerAbstractModuleWidget::setEditedNode() method is called.

12.3.6 Remote Module

Purpose of Remote Modules

- Keep the Slicer core lean.
- Allow individuals or organizations to work on their own private modules and optionally make these modules available to the Slicer users without the need to use the extensions manager.

Policy for Adding Remote Modules

- Module is known to compile on Linux, MacOSX and Windows.
- Module is tested.
- Module is documented on the wiki.
- Module names must be unique.
- At no time in the future should a module in the main Slicer repository depend on Remote module.
- Remote modules MUST define a specific **unique** revision (i.e. git hash). It is important for debugging and scientific reproducibility that there be a unique set of code associated with each slicer revision.

Procedure for Adding a Remote Module

1. Discuss with Slicer core Developers
2. Add an entry into `SuperBuild.cmake` using Slicer_Remote_Add() macro. For example:

```
Slicer_Remote_Add(Foo
  GIT_REPOSITORY ${git_protocol}://github.com/awesome/foo
  GIT_TAG abcdef
  OPTION_NAME Slicer_BUILD_Foo
  LABELS REMOTE_MODULE
  )
list_conditional_append(Slicer_BUILD_Foo Slicer_REMOTE_DEPENDENCIES Foo)
```

3. Corresponding commit message should be similar to:

```
ENH: Add Foo remote module

The Foo module provide the user with ...
```

..note::

As a side effect of calling `Slicer_Remote_Add`, (1) the option `Slicer_BUILD_Foo` will automatically be added as an advanced option and (2) the CMake variables `Slicer_BUILD_Foo` and `Foo_SOURCE_DIR` will be passed to Slicer inner build.

Additionally, by specifying the `REMOTE_MODULE` label, within `Slicer/Modules/Remote/CMakeLists.txt`, the corresponding source directory will automatically be added using a call to `add_directory`.

`Slicer_Remote_Add` creates an in-source module target within `Slicer/Modules/Remote`. The SuperBuild target for a remote module only runs the source update step; there is no separate build step.

Procedure for Updating a Remote Module

1. Update the entry into `SuperBuild.cmake`
2. Commit with a message similar to:

```
ENH: Update Foo remote module

List of changes:

$ git shortlog abc123..efg456
John Doe (2):
  Add support for ZZZ spacing
  Refactor space handler to support multi-dimension
```

12.4 Parameter Nodes

12.4.1 Overview

Parameter nodes are often used in scripted modules to store data such that it can be easily saved to the MRML scene. They are simply MRML nodes that exist in a scene and store data. One of their most common uses is to save GUI state in module widgets.

The parameter node concept is implemented in C++ in the `vtkMRMLScriptedModuleNode` which has `GetParameter` and `SetParameter` methods for saving arbitrary string data to the scene. While the base `vtkMRMLScriptedModuleNode` is great for scene saving, treating all data as strings is not ideal, so a parameter node wrapper was implemented.

Parameter Node Wrapper

The parameter node wrapper allows wrapping around a `vtkMRMLScriptedModuleNode` parameter node with typed member access. The wrapper will serialize values into an underlying `vtkMRMLScriptedModuleNode`, and has a *caching mechanism* so multiple reads of a parameter without a write won't convert from string every time.

A simple example is as follows.

```python
import slicer
from slicer import vtkMRMLModelNode
from slicer.parameterNodeWrapper import *

```

(continues on next page)

(continued from previous page)

```
@parameterNodeWrapper
class CustomParameterNode:
  numIterations: int
  inputs: list[vtkMRMLModelNode]
  output: vtkMRMLModelNode
```

This will create a new class called `CustomParameterNode` that has 3 members properties, an `int` named `numIterations`, a `list` of `vtkMRMLModelNode`s named `inputs`, and a `vtkMRMLModelNode` named `output`.

The `@parameterNodeWrapper` decorator will generate a constructor for this class that takes one argument, a `vtkMRMLScriptedModuleNode` parameter node.

An example usage is as follows:

```
parameterNode = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLScriptedModuleNode')
param = CustomParameterNode(parameterNode)

# can set the property directly with an appropriate type
param.numIterations = 500
param.inputs = [slicer.mrmlScene.AddNewNodeByClass('vtkMRMLModelNode') for _ in range(5)]
param.output = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLModelNode')

# pythonic list usage
for inputModel in param.inputs:
  mesh = inputModel.GetMesh()
  # ...

for iteration in range(param.numIterations):
  # run iteration

param.output.SetAndObserveMesh(...)
```

Much more complex wrappers can be written:

For the full list of supported types, including how to make custom types with invariants, see *Supported types*

For restricting valid values for a parameter, see *Validators*.

GUI binding and creation

Qt widgets can automatically be created for and bound to `parameterNodeWrappers`.

See *GUI Connection* for how to bind a `parameterNodeWrapper` to existing widgets (including directly in a `.ui` file).

See *GUI Creation* for how to generate widgets for `parameterNodeWrappers`.

12.4.2 Supported types

Built-in types

The `@parameterNodeWrapper` decorator keys off of the type hints given in the class definition, similar to python's dataclasses.dataclass class.

The classes that are recognized by default are

- `int`
- `float`

- `str`
- `bool`
- `vtkMRMLNode` (including subclasses)
- `list` (hinted as `list[int]`, `list[str]`, etc)
- `tuple` (hinted as `tuple[int, bool]`, `tuple[str, vtkMRMLNode, float]`, etc)
- `dict` (hinted as `dict[keyType, valueType]`)
- `enum.Enum`
- `pathlib.Path`
- `pathlib.PosixPath`
- `pathlib.WindowsPath`
- `pathlib.PurePath`
- `pathlib.PurePosixPath`
- `pathlib.PureWindowsPath`
- `typing.Union` (hinted as `typing.Union[int, str]`, `typing.Union[bool, vtkMRMLModelNode, float]`, etc)
- `typing.Optional` (hinted as `typing.Optional[int]`, `typing.Optional[float]`, etc)
- `parameterPack` (see *Parameter Packs*)

When using container types like `list`, `tuple`, or `dict`, only these types are recognized as element types by default. Note that containers can be nested, such as `dict[str, list[int]]`.

For using custom types in parameter node wrappers, first see if *Parameter Packs*, will suit your needs. If not, check out the *Custom Classes* page.

MRML nodes

MRML nodes from non-core modules are supported, but to define a `parameterNodeWrapper` in the global space of a module (i.e. not inside a function) you can't use the classes from `slicer` namespace. This is because they are not added to the slicer namespace until after the class is created. Here is an example on how to use `vtkMRMLMarkupsFiducialNode`.

```
from slicer.parameterNodeWrapper import *
from slicer import vtkMRMLMarkupsFiducialNode

@parameterNodeWrapper
class CustomParameterNode:
  markup: vtkMRMLMarkupsFiducialNode
  markups: list[vtkMRMLMarkupsFiducialNode]
```

Warning

For uses of MRML nodes in Python code, it is preferred to use `slicer.<node>` rather than `from <specific-mrml-package> import <node>` as this is more robust if MRML nodes change packages (e.g. from an extension to the core).

Enum

Instances of `enum.Enum` are serialized by their *name*, not their value. This allows enums to hold unserialized metadata, accessible via `Enum.value`.

```
from slicer.parameterNodeWrapper import *
from enum import Enum

class Color(Enum):
    RED = '#FF0000'
    GREEN = '#00FF00'
    BLUE = '#0000FF'

@parameterNodeWrapper
class CustomParameterNode:
    color: Color
```

It also means that if you changed the name of an enum value (e.g. from `RED` to `Red`), it would break the loading of old MRB files or MRML scenes that saved a `Color` parameter.

Parameter Packs

It is often useful to group related information together in structures with useful names. Another decorator, `@parameterPack` was added to make this easier. This will make the class behave in a similar manner to Python's `@dataclasses.dataclass`. These `parameterPack`s can then be used in a `parameterNodeWrapper`. Typically the name will stored as `<pack-name>.<pack-member-name>` in the underlying parameter node when used with a `parameterNodeWrapper`.

```
from slicer.parameterNodeWrapper import *


@parameterPack
class Point:
    x: float
    y: float


@parameterPack
class BoundingBox:
    # can nest parameterPacks
    topLeft: Point
    bottomRight: Point


@parameterNodeWrapper
class ParameterNodeType:
    # Can add them to a @parameterNodeWrapper like any other type.
    # Will be stored in the underlying parameter node as
    #  - box.topLeft.x (default value is 0)
    #  - box.topLeft.y (default value is 1)
    #  - box.bottomRight.x (default value is 1)
    #  - box.bottomRight.y (default value is 0)
    box: BoundingBox = BoundingBox(Point(0, 1), Point(1, 0))
```

(continues on next page)

(continued from previous page)

```
parameterNode = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLScriptedModuleNode')
param = ParameterNodeType(parameterNode)

# can set wholesale
param.box.topLeft = Point(-4, 5)

# or can set piecewise
param.box.bottomRight.x = 4
param.box.bottomRight.y = -5
```

The created `parameterPack` will have the following attributes:

```
>>> from typing import Annotated
>>> from slicer.parameterNodeWrapper import *
>>>
>>> @parameterPack
>>> class ParameterPack:
>>>     # if the type is Annotated, it will treat the annotations the same as␣
↪@parameterNodeWrapper
>>>     x: Annotated[float, WithinRange(0, 10)]
>>>     option: Annotated[str, Choice(["a","b"])] = "b"
>>>
>>> # with no arguments the constructor will use the given (or implied) defaults.
>>> p1 = ParameterPack()  # == ParameterPack(x=0.0, option="b")
>>>
>>> # positional arguments are accepted in the order the members are declared in
>>> p2 = ParameterPack(3.0, "a")
>>>
>>> # keyword arguments are accepted with the keyword being the member names
>>> p3 = ParameterPack(option="a", x=3.0)
>>>
>>> # unspecified arguments use their default
>>> p4 = ParameterPack(4.5)  # == ParameterPack(x=4.5, option="b")
>>> p5 = ParameterPack(option="a")  # == ParameterPack(x=0.0, option="a")
>>>
>>> # validators are run on construction
>>> p6 = ParameterPack(-1, "a")
ValueError: Value must be within range [0, 10], is -1
>>>
>>> # validators are run on set attribute
>>> p4.option = "c"
ParameterPack(x=4.5, option=b)
>>>
>>> # the classes automatically have __eq__ added to them
>>> p1 == p2
False
>>> p2 == p3
True
>>>
>>> # the classes are also given a __repr__ and a __str__ that describes their attributes
>>> print(p4)
```

(continues on next page)

(continued from previous page)

```
ParameterPack(x=4.5, option="b")
```

If any of the autogenerated dunder methods (`__init__`, `__eq__`, `__str__`, `__repr__`) are overridden in the parameterPack, they will not be autogenerated.

```python
@parameterPack
class ParameterPack:
    i: int
    j: int
    k: str

    # custom constructor
    def __init__(self, k):
        self.i = 1
        self.j = 4
        self.k = k

    # default __eq__, __str__, and __repr__ are generated
```

Parameter Packs with invariants

Invariants can be added to `parameterPack`s via validators as described above. However this is only good for invariants on independent parameters (you can do something like `x: Annotated[int, Minimum(0)]`, but you can't do something like make sure parameter `x` is less than another parameter `y` with validators).

More complex invariants can be enforced by making the `parameterPack` parameters private (via leading underscore convention) and then offering public accessors that enforce the invariants. While this is a little bit more work, most of the time it is still less work than making a non-`parameterPack` custom class work with the `parameterNodeWrapper`.

E.g.

```python
from typing import Annotated
from slicer.parameterNodeWrapper import (
    parameterPack,
    Default,
)


class BadDateException(ValueError):
    pass


@parameterPack
class Date:
    # Private parameters that will be written to the scene.
    # Can still set defaults on the private parameters.
    _month: int = 1
    _day: int = 1
    _year: int = 1970

    # A checker for the multi-parameter invariant
    @staticmethod
    def checkDate(month, day, year):
```

(continues on next page)

(continued from previous page)

```
        # note: this is assuming leap years don't exist and negative years are allowable
        if month < 1 or month > 12 or day < 1 or day > 31:
            raise BadDateException(f"Bad date: {month}/{day}/{year}")
        if month == 2 and day > 28:
            raise BadDateException(f"Bad date: {month}/{day}/{year}")
        if month in (4, 6, 9, 11) and day > 30:
            raise BadDateException(f"Bad date: {month}/{day}/{year}")

    # override the __init__ function to enforce the invariant
    def __init__(self, month=None, day=None, year=None) -> None:
        if month is not None:
            self._month = month
        if day is not None:
            self._day = day
        if year is not None:
            self._year = year
        self.checkDate(self._month, self._day, self._year)

    def __str__(self) -> str:
        return f"Date(month={self.month}, day={self.day}, year={self.year})"

    # Add properties that access the private parameters and enforces the invariant.
    @property
    def month(self):
        return self._month

    @month.setter
    def month(self, value):
        self.checkDate(value, self.day, self.year)
        self._month = value

    @property
    def day(self):
        return self._day

    @day.setter
    def day(self, value):
        self.checkDate(self.month, value, self.year)
        self._day = value

    @property
    def year(self):
        return self._year

    @year.setter
    def year(self, value):
        self._year = value

    # Can even add helper functions for a nicer interface.
    def setDate(self, month: int, day: int, year: int) -> None:
        self.checkDate(month, day, year)
        self._month = month
```

(continues on next page)

(continued from previous page)

```
        self._day = day
        self._year = year
```

Warning

When trying to enforce invariants over complex classes, care needs to be taken to not allow the user to accidentally break the invariant. Consider the following code:

```python
from slicer.parameterNodeWrapper import (
    parameterPack,
)


@parameterPack
class InvariantTestPack:
    _listA: list[int]
    _listB: list[int]

    @staticmethod
    def checkInvariant(listA, listB):
        if not len(listA) <= len(listB):
            raise ValueError("Invariant failed")

    def __init__(self, listA=None, listB=None) -> None:
        if listA is not None:
            self._listA = listA
        if listB is not None:
            self._listB = listB
        self.checkInvariant(self._listA, self._listB)

    @property
    def listA(self):
        return self._listA

    @listA.setter
    def listA(self, value):
        self.checkInvariant(value, self.listB)
        self._listA = value

    @property
    def listB(self):
        return self._listB

    @listB.setter
    def listB(self, value):
        self.checkInvariant(self.listA, value)
        self._listB = value
```

It is very easy for the user to accidentally break the invariant

```python
pack = InvariantTestPack(listA=[], listB=[])

pack.listA = [1]  # raises ValueError

pack.listA.append(1)  # BAD: does not raise ValueError
```

The problem comes from the fact that giving direct access to `listA` and `listB` allows the user to change them without `InvariantTestPack`'s knowledge. This is just how Python works.

This can be worked around a couple of ways. One way is to essentially flatten the methods of `list` into the `parameterPack` (e.g. `def appendListA(value) -> None`, `def getListA(index) -> int`, `def setListB(index, value) -> None`). Or if the object you are returning has observation capabilities (similar to signals from Qt or `AddObserver` from VTK) you can try to hook into those.

12.4.3 Defaults

Default values

Default values can easily be used when creating `parameterNodeWrapper`s and `parameterPack`s by simply assigning a value to the parameter.

E.g.

```python
from typing import Annotated
from slicer.parameterNodeWrapper import parameterNodeWrapper


@parameterNodeWrapper
class ParameterNodeWrapper:
    iterations: int = 50
    text: str = "abc"
    tup: tuple[int, bool] = (7, True)
```

Note

When constructing a `parameterNodeWrapper` using a parameter node that already has values set (e.g. when loading a .mrb or a .mrml scene file), those values will be maintained. The default will only come into play if parameter node does not have a value for the parameter.

Unspecified defaults

When a parameter in a `parameterPack` or `parameterNodeWrapper` is not given an explicit default value, the following values will be used:

Warning

If `typing.Union[SomeType, None]` is used, the default will be `None`. This will only happen if there are exactly 2 options in the union and the last one is `None`. In Python (not just the parameter node wrappers), writing `typing.Union[SomeType, None]` is equivalent to writing `typing.Optional[SomeType]`.

Default generators

Occasionally, it may be useful to use a generator function to create the default value, rather than a fixed value. One possible use of this is to choose a default for a `Path` after Slicer has finished loading and the appropriate resource directories are known.

E.g.

```
import os
from typing import Annotated
import slicer
from slicer.parameterNodeWrapper import parameterNodeWrapper, Default

def defaultIcon():
    # this will not change, but it can't be queried until this module is loaded
    return pathlib.Path(os.path.join(
        os.path.dirname(slicer.util.modulePath(MyModule.__name__)),
        'Resources',
        'Icons',
        'defaultIcon.png')
    )


@parameterNodeWrapper
class PipelineCreatorMk2ParameterNode:
    icon: Annotated[pathlib.Path, Default(generator=defaultIcon)]
```

12.4.4 Validators

Overview

It can be useful to restrict the set of that values that a parameter node parameter can be set to. These can be done with `Validator` annotations.

```
from typing import Annotated
from slicer.parameterNodeWrapper import parameterNodeWrapper, Minimum, Default

@parameterNodeWrapper
class CustomParameterNode:
  numIterations: Annotated[int, Minimum(0)] = 500

  # To have a list where the values in the list need to be validated
  chosenFeatures: list[Annotated[str, Choice(["feat1", "feat2", "feat3"])]]
```

This will cause a `ValueError` to be raised if someone tried setting `numIterations` to a negative value.

Multiple validators can be placed in the `Annotated` block and they will be run in the order they were placed.

Built-in validators

The list of built-in validators is as follows:

Some built-in types have the following validators applied to them by default:

Custom Validators

Custom validators can easily be created and used with the `parameterNodeWrapper`.

```
import re
from typing import Annotated
from slicer.parameterNodeWrapper import parameterNodeWrapper, Validator

# Custom validators must derive from the Validator class.
```

(continues on next page)

(continued from previous page)

```
class MatchesRegex(Validator):
  def __init__(self, regex):
    self.regex = regex
  # Custom validators must implement a validate function that raises an Exception
  # if the given value is invalid.
  def validate(self, value):
    if re.match(self.regex, value) is None:
      raise ValueError("Did not match regex")

@parameterNodeWrapper
class CustomParameterNode:
  value: Annotated[str, MatchesRegex("[abc]+")] = "abcba"


param = CustomParameterNode(slicer.mrmlScene.AddNewNodeByClass('vtkMRMLScriptedModuleNode
↪'))

param.value = "abcabc" # ok
param.value = "d" # ValueError raised
```

12.4.5 GUI Connection

Data binding

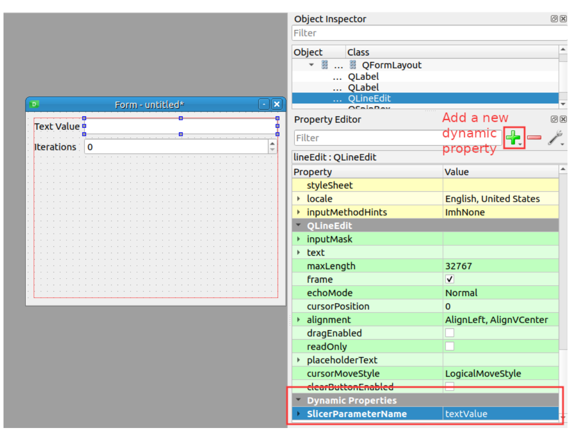
The parameter node wrappers have the ability to do two-way data binding between particular GUI elements and parameters of certain types. This means that when the GUI is updated the parameter node wrapper will be automatically updated, and when the parameter node wrapper is updated the GUI will automatically be updated.

There are two ways to declare which GUI widget elements connect to which parameters.

Connecting widgets from a .ui file

Qt Designer has the ability to set Dynamic Properties. These can be used to inform the parameter node wrapper infrastructure which parameters to connect to which widgets. This is the preferred way of connection from the `.ui` file of a scripted module to its parameter node wrapper.

Simply set a dynamic string property named "SlicerParameterName" to the parameter name it should bind with. If the property does not exist yet then add it by clicking the + button above the property list.



Then use the `connectGui` of the parameterNodeWrapper to connect.

```python
import slicer
from slicer.parameterNodeWrapper import parameterNodeWrapper

@parameterNodeWrapper
class MyModuleParameterNode:
  textValue: str
  iterations: int

class MyModuleWidget(ScriptedLoadableModuleWidget, VTKObservationMixin):
  def setup(self):
    ...

    # assuming the image above is in MyModule.ui
    uiWidget = slicer.util.loadUI(self.resourcePath('UI/MyModule.ui'))
    self.layout.addWidget(uiWidget)
    self.ui = slicer.util.childWidgetVariables(uiWidget)

    ...

  def enter(self):
    self.initializeParameterNode()
    # the connectGui call sets up the bindings and returns a tag that can be
    # used to disconnect the GUI from the parameter node wrapper.
    self._parameterNodeConnectionTag = self._parameterNode.connectGui(self.ui)

  def exit(self):
    # Do not react to parameter node changes (GUI will be updated when the user enters␣
↪into the module)
    self._parameterNode.disconnectGui(self._parameterNodeConnectionTag)
    self._parameterNodeConnectionTag = None

  def onApply(self):
    # Because the "SlicerParameterName" properties were set in the .ui file, textValue
    # and iterations are updated whenever their corresponding widgets are updated.
    self.logic.run(self._parameterNode.textValue, self._parameterNode.iterations)
```

Note

The dynamic properties can also be set in code via `widget.setProperty("SlicerParameterName", "parameterName")`

Tip

Widgets can be connected piecewise to parameter packs by using a dot syntax.

```python
@parameterPack
class Point:
  x: float
  y: float
```

```
@parameterPack
class BoundingBox:
  topLeft: Point
  bottomRight: Point

@parameterNodeWrapper
class CustomParameterNode:
  box: BoundingBox

# In the .ui file, there could be 4 QDoubleSpinBoxes that had the following
↪"SlicerParameterName"s
#   box.topLeft.x
#   box.topLeft.y
#   box.bottomRight.x
#   box.bottomRight.y
#
# Each of the QDoubleSpinBoxes would be bound to the appropriate sub-piece of the␣
↪parameterPacks in
# the parameterNodeWrapper
```

Manual connection

If a `.ui` is not used, the widget to parameter mapping can be manually specified.

```
from typing import Annotated
from slicer.parameterNodeWrapper import (
  parameterNodeWrapper,
  parameterPack,
  Validator,
)

@parameterPack
class Point:
  x: float
  y: float

@parameterPack
class BoundingBox:
  topLeft: Point
  bottomRight: Point

@parameterNodeWrapper
class CustomParameterNode:
  iterations: int
  box: BoundingBox

param = CustomParameterNode(slicer.mrmlScene.AddNewNodeByClass('vtkMRMLScriptedModuleNode
↪'))

topLeftXSpinbox = qt.QDoubleSpinBox()
topLeftYSpinbox = qt.QDoubleSpinBox()
bottomRightXSpinbox = qt.QDoubleSpinBox()
```

(continues on next page)

(continued from previous page)

```
bottomRightYSpinbox = qt.QDoubleSpinBox()
iterationsSlider = qt.QSlider()

mapping = {
  # Key is parameter name, value is widget object
  "iterations", iterationsSlider,

  # For parameterPacks, can access nested parameter items through dot syntax
  "box.topLeft.x": topLeftXSpinbox,
  "box.topLeft.y": topLeftYSpinbox,
  "box.bottomRight.x": bottomRightXSpinbox,
  "box.bottomRight.y": bottomRightYSpinbox,
}

connectionTag = param.connectParametersToGui(mapping)

# When the GUI items are updated, it will automatically update the value
# in the parameter node wrapper.
# Also, when the parameter node wrapper is updated, it will automatically
# update the GUI.
param.box.topLeft.x = 4.2
# Now topLeftXSpinbox.value == 4.2 because of the connections

# can use the disconnectGui method to break the connection
param.disconnectGui(connectionTag)
```

Available connectors

It is not possible to convert from all widget types to all data types. For instance, converting from a `QCheckBox` to a `vtkMRMLModelNode` is not really possible.

The following connections are supported:

Note

Some widget bindings will look at the Validators on the type and make updates to the widget accordingly.

E.g.

When connecting an `int` to a `QSpinBox`, if the `Minimum` annotation is used, it will call `spinbox.setMinimum` when `connectGui/connectParametersToGui` is called.

For widgets that are not listed here see Slicer Issue 7308 for discussion and progress.

Extra annotatations

For some GUIs, extra annotations can be used to set other widget properties. This is especially useful when used in conjunction with `createGui` (see *GUI Creation*).

Custom connectors and reusable widgets

To create custom widgets for parameter packs, see *Custom Widgets for Parameter Packs*

12.4.6 GUI Creation

Auto-generating widgets

It is possible to auto-generate GUIs for parameter node wrappers, parameter packs, and their underlying types. The GUI creation will take into account the validators used when choosing which type of widget to generate for a type.

Usage is simple:

```
from typing import Annotated

from slicer.parameterNodeWrapper import (
  createGui,
  parameterNodeWrapper,
  parameterPack,
)

@parameterPack
class Point:
    x: float
    y: float

@parameterNodeWrapper
class ParameterNodeWrapper:
    point: Point
    reduction: Annotated[float, WithinRange(0, 1)]
    text: str

parameterNodeWidget = createGui(ParameterNodeWrapper)
```

The `createGui` methods takes a (possibly annotated) type as input and returns an appropriate widget.

Warning

createGui is intended to work with the *GUI connection*, so it won't call things like setMinimum/setMaximum on the created widgets, as it expects <parameterNodeWrapper>.connectGui to do that.

Created widget types

* These are qSlicerWidgets so they have a setMRMLScene method. The generated GUI will automatically connect the top level setMRMLScene to any children that have a setMRMLScene method (e.g. qMRMLComboBox).

Tip

You can use the Label annotation to give a nicer user facing label to a parameter in a parameterPack or parameterNodeWrapper.

```
from typing import Annotated
from slicer import vtkMRMLScalarVolumeNode
from slicer.parameterNodeWrapper import (
  createGui,
  parameterNodeWrapper,
  Label,
)

@parameterNodeWrapper
class ParameterNodeWrapper:
    inputVolume: Annotated[vtkMRMLScalarVolumeNode, Label("Input Volume")]

widget = createGui(ParameterNodeWrapper)
# the label is now "Input Volume" instead of "inputVolume".
```

12.4.7 Advanced

Custom Widgets for Parameter Packs

Sometimes a parameter pack is used multiple times in a parameter node wrapper. Take the following example:

```
from slicer.parameterNodeWrapper import *

@parameterPack
class Point:
    x: float
    y: float

@parameterPack
class BoundingBox:
    # can nest parameterPacks
    topLeft: Point
    bottomRight: Point

@parameterNodeWrapper
class ParameterNodeType:
    box1: BoundingBox
    box2: BoundingBox
```

in this example it could be useful to have a `PointWidget` that directly represented the `Point` class. The widget could be reused multiple times (even across multiple files) to ensure a consistent experience for inputting points.

Here is an example on how that could be done. The `PointWidget` is being created in code here, but this could also apply to a custom widget created in a `.ui` file.

```
class PointWidget(qt.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.setLayout(qt.QHBoxLayout(self))

        self.xWidget = qt.QDoubleSpinBox()
        # set the parameterPack parameter this widget corresponds to
        self.xWidget.setProperty("SlicerPackParameterName", "x")
        self.yWidget = qt.QDoubleSpinBox()
        # set the parameterPack parameter this widget corresponds to
        self.yWidget.setProperty("SlicerPackParameterName", "y")

        self.layout().addWidget(self.xWidget)
        self.layout().addWidget(self.yWidget)


topLeft1Widget = PointWidget()
topLeft2Widget = PointWidget()

param = ParameterNodeType(slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScriptedModuleNode
↪"))

param.connectParametersToGui({
    "box1.topLeft": topLeft1Widget,
    "box2.topLeft": topLeft2Widget,
})
```

Setting the `"SlicerPackParameterName"` property on a child widget of the custom widget is enough for the GUI connection infrastructure to make the binding.

The infrastructure uses parent-child relationships to work through nested values.

Here is an example of a `BoundingBoxWidget` that is completely separate from `PointWidget` and uses parent-child relationships to match the nesting structure of the parameter packs.

```
class BoundingBoxWidget(qt.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)

        self.setLayout(qt.QVBoxLayout(self))

        # Making a widget for the topLeft point. This particular QWidget doesn't
        # do anything special for the UI, but it parents the x and y widgets for the
        # top left points
        topLeftWidget = qt.QWidget()
        self.layout.addWidget(topLeftWidget)
        topLeftWidget.setProperty("SlicerPackParameterName", "topLeft")
        topLeftWidget.setLayout(qt.QHBoxLayout())
```

(continues on next page)

(continued from previous page)

```
        topLeftXWidget = qt.QDoubleSpinBox()
        topLeftXWidget.setProperty("SlicerPackParameterName", "x")
        topLeftYWidget = qt.QDoubleSpinBox()
        topLeftYWidget.setProperty("SlicerPackParameterName", "y")

        topLeftWidget.layout().addWidget(topLeftXWidget)
        topLeftWidget.layout().addWidget(topLeftYWidget)

        # Making a widget for the bottomRight point. This particular QWidget doesn't
        # do anything special for the UI, but it parents the x and y widgets for the
        # top left points
        bottomRightWidget = qt.QWidget()
        self.layout.addWidget(bottomRightWidget)
        bottomRightWidget.setProperty("SlicerPackParameterName", "bottomRight")
        bottomRightWidget.setLayout(qt.QHBoxLayout())

        bottomRightXWidget = qt.QDoubleSpinBox()
        bottomRightXWidget.setProperty("SlicerPackParameterName", "x")
        bottomRightYWidget = qt.QDoubleSpinBox()
        bottomRightYWidget.setProperty("SlicerPackParameterName", "y")

        bottomRightWidget.layout().addWidget(bottomRightXWidget)
        bottomRightWidget.layout().addWidget(bottomRightYWidget)
```

Custom Classes

Note

Before creating your own custom class to be used by the parameter node wrapper infrastructure, see if *parameter packs* fit your needs.

If you just want to be able to use a custom, reusable widget for a parameter pack, see *Custom Widgets for Parameter Packs*.

Using custom classes in a parameterNodeWrapper

This section is meant as a guide to make new types available for use in the `parameterNodeWrapper` and `parameterPacks`. The serialization section must always be done, but the GUI sections are only needed if you want to support GUI connection or creation with your type. New types should be able to be added in extensions (but this is untested).

Serialization into the underlying vtkMRMLScriptedModuleNode

One of the main things done by the `parameterNodeWrapper` is the serialization and deserialization of Python types into the `vtkMRMLScriptedModuleNode` (colloquially called a parameter node). This is thing that the `parameterNodeWrapper` wraps.

There are two ways to store information in the `vtkMRMLScriptedModuleNode`, as a string or as a reference to a MRML node. All non-MRML node types use the string approach. This means that when an `int` is used in the `parameterNodeWrapper`, it is stored and read as a string under the hood.

A `Serializer` interface exists to read and write types into the `vtkMRMLScriptedModuleNode`. It is located in the

Slicer repository in `Base/Python/slicer/parameterNodeWrapper/serializers.py`. For each new class to be used, you must create a custom serializer derived from `Serializer` and implement the needed methods. Refer to the `serializers.py` file for the methods that need to be implemented.

Additionally, there is a `@parameterNodeSerializer` decorator that the custom serializer must be decorated with. This will register the serializer into the infrastructure so it can be found when the associated type is used in a `parameterNodeWrapper` or `parameterPack`.

Here is an example of a custom class serializer.

```
import dataclasses
import slicer
from slicer.parameterNodeWrapper import parameterNodeWrapper, Serializer,␣
↪ValidatedSerializer

# Note: for this case it would be much simpler to just use a parameterPack.
@dataclasses.dataclass
class CustomClass:
    x: int
    y: int
    z: int

# The Serializer class is used to read and write the values to the underlying
# vtkMRMLScriptedModuleNode. There are built-in serializers for each of the support␣
↪built-in types.
# Adding a new serializer involves deriving from Serializer and implementing the␣
↪following methods.
# The @parameterNodeSerializer decorator registers the serializer so it can be found by a
# parameterNodeWrapper.
@parameterNodeSerializer
class CustomClassSerializer(Serializer):
    @staticmethod
    def canSerialize(type_) -> bool:
        """
        Whether the serializer can serialize the given type if it is properly␣
↪instantiated.
        """
        return type_ == CustomClass

    @staticmethod
    def create(type_):
        """
        Creates a new serializer object based on the given type. If this class does not␣
↪support the given type,
        None is returned.

        It is common for the returned type to actually be a ValidatedSerializer wrapping␣
↪this serializer that implements
        any default validators (NotNone and IsInstance are common).
        """
        if CustomClassSerializer.canSerialize(type_):
            # in our example, lets say that we don't allow None. We will use NotNone() to␣
↪enforce this
            return ValidatedSerializer(CustomClassSerializer(), [NotNone(),␣
```

(continues on next page)

(continued from previous page)

```
↪IsInstance(CustomClass)])
        return None

    def default(self):
        """
        The default value to use if another default is not specified.
        """
        return CustomClass(0, 0, 0)

    def isIn(self, parameterNode: slicer.vtkMRMLScriptedModuleNode, name: str) -> bool:
        """
        Whether the parameterNode contains a parameter of the given name.
        Note that most implementations can just use parameterNode.HasParameter(name).
        """
        return parameterNode.HasParameter(name)

    def write(self, parameterNode: slicer.vtkMRMLScriptedModuleNode, name: str, value) ->
↪ None:
        """
        Writes the value to the parameterNode under the given name.
        Note: It is acceptable to mangle the name as long the same name can be used for␣
↪reading.
        For example the built-in ListSerializer does this.
        """
        parameterNode.SetParameter(name, f"{value.x},{value.y},{value.z}")

    def read(self, parameterNode: slicer.vtkMRMLScriptedModuleNode, name: str):
        """
        Reads and returns the value with the given name from the parameterNode.
        """
        val = parameterNode.GetParameter(name)
        vals = val.split(',')
        return CustomClass(int(vals[0]), int(vals[1]), int(vals[2]))

    def remove(self, parameterNode: slicer.vtkMRMLScriptedModuleNode, name: str) -> None:
        """
        Removes the value of the given name from the parameterNode.
        """
        parameterNode.UnsetParameter(name)

@parameterNodeWrapper
class CustomClassParameterNode(object):
    # can now use CustomClass like any other type for building parameterNodeWrappers
    custom: CustomClass = CustomClass(1,2,3)
    listOfCustom: list[CustomClass]
```

Note that this example doesn't allow caching of the custom class. See *Caching* for more information on this.

Tip

If doing more complicated container-style classes similar to `list` or `dict`, or if you are calling `parameterNode.setParameter` multiple times in a single `write` call, make sure to use `slicer.util.NodeModify` on the vtkMRMLScriptedModuleNode parameterNode before calling the writes so there is only one modification notification. This also prevent modification notifications while the value is half written.

Tip

If saving multiple sub-values in one write call in your custom serializer, use a character that is not allowed in a Python variable name as a separator. This will help prevent name clashes.

For example, the ListSerializer stores a separate entry in the parameter node for each element, as well as the size. Consider the following example:

```
@parameterNodeWrapper
class ParameterNodeWrapper:
    strings: list[str]

param = ParameterNodeWrapper(slicer.mrmlScene.AddNewNodeByClass(
↪"vtkMRMLScriptedModuleNode"))
param.strings = ["a", "b", "c"]
```

Four separate elements are stored in the underlying vtkMRMLScriptedModuleNode.

- strings.len = 4
- strings.0 = "a"
- strings.1 = "b"
- strings.2 = "c"

By using a . in the element keys (as opposed to something like _ which is valid in variable names), we ensure that we cannot accidentally conflict with another variable named something like strings_len.

GUI connection

Automatic GUI connection is another thing that may be useful for your custom class. Similar to the serializer above, there is a GuiConnector interface that must be implemented and a @parameterNodeGuiConnector decorator for registering it.

The point of the GuiConnector is to give all widgets the same interface so they can be used generically by the infrastructure. It is assumed that all widgets used with the parameterNodeWrapper allow the input of some value. The GuiConnector is responsible for converting the widget's representation of that this value to something the serializer can understand (namely an instance of the Python type used in the type hint).

For each parameter and its corresponding widget in a parameterNodeWrapper.connectGui call, the infrastructure will ask each registered GuiConnector type if it can connect the parameter type to the widget. If it says yes, it will create that connector. The connectors can inspect the type for any annotations, such as Validators, and use that information to adjust the properties of the given widget.

See the GuiConnector class in Base/Python/slicer/parameterNodeWrapper/guiConnectors.py for a complete list of methods to implement.

Note

If your custom class can be represented by multiple widgets, it is often easiest to make a separate GuiConnector for each type-widget pair.

GUI creation

Should you also desire automatic GUI creation of your custom class, that can be achieved as well.

There is a `GuiCreator` interface that can be implemented and a `@parameterNodeGuiCreator` decorator for registering the creator.

Because it is possible to represent types using different widgets, and annotations like `Minimum` and `Maximum` can influence which widget is the "best" representation, `GuiCreator`s are asked how well they can represent a possibly annotated type, and the one with the highest representation value is used.

The range for representation values is 0 - 100 (although there is no enforcement of this). A value of 0 means the type cannot be represented by a widget created by this creator, whereas 100 means this creator can make the perfect widget for representing the type. The highest representation value given by the built in creator is 90, which is when the `Choice` validator is used. This means you could write `GuiCreator`s that for already supported types that will override the widget used if certain custom annotations are found.

See the `GuiCreator` class in `Slicer-build/bin/Python/slicer/parameterNodeWrapper/guiCreation.py` for a complete list of methods to implement.

Caching

Caching overview

With the exception of MRML nodes, all values are stored in the underlying `vtkMRMLScriptedModuleNode` as strings. The constant conversion from string to non-string (e.g. `int`, `float`, `list[int]`, etc) each time a parameter is read can be very inefficient. To help with this, a caching system was put in place. If a value hasn't been written through the *parameter node wrapper* since the last read, then it will use the cached value. Note this means if the underlying `vtkMRMLScriptedModuleNode` parameter node changes outside of the wrapper, the cache will not be updated and the cached value will be wrong.

This is the chosen behavior for the following reasons:

- Mixed usage of the parameter node wrapper and the `vtkMRMLScriptedModuleNode` parameter node for the same parameter node is not expected.
- The `vtkMRMLScriptedModuleNode` parameter node does not offer per parameter VTK event callbacks. Therefore, if a callback was setup off the `ModifiedEvent`, *all* parameters would be re-read for *every* write to *any* parameter in the node.
  - The `vtkMRMLScriptedModuleNode` parameter node may be updated in the future to give a `ParameterModifiedEvent` that gives the parameter that was modified. If this happens, the caching behavior may be revisited.

The caching system will cache the last written for each parameter that allows it. Determination of whether a value can be cached is done a per type basis, and defaults to no caching for custom types. Most of the built in type support has caching enabled, with the biggest exception being `typing.Any`. The container classes (`list`, `dict`, `parameterPack`, etc) usually only allow caching if all of their element types allow caching.

Caching support for list and dict

As mentioned above, `list` and `dict` can be cached if their element types (key and value type for `dict`) can all be cached. Because Python does return by reference for complex types, and types like `list` and `dict` are mutable, some extra steps were taken to ensure if the containers were updated, it would propagate down to the underlying `vtkMRMLScriptedModuleNode`.

This was done by adding `ObservedList` and `ObservedDict` classes that were able to write to the `vtkMRMLScriptedModuleNode` when they are modified. This means that anytime a `list` or `dict` is used in `parameterNodeWrapper`, it is actually an `ObservedList` or an `ObservedDict` that is returned. These classes function very much like their non-observed counterparts. Reading values and iterating over the values will not cause any string to type conversions, but writing will. Modification of the containers is allowed (e.g. `ObservedListInstance += ["list", "of", "items"]`). Also you can store the container as a variable and update the variable.

```
strings = parameterNodeWrapperInstance.listOfStrings
strings.append("text")
# the parameter node is automatically updated to add "text" its list
```

Caching custom classes

Custom classes can also be cached, as long as you can ensure that any changes to the class can automatically propagate to the underlying `vtkMRMLScriptedModuleNode`. Use `ListSerializer` and `ObservedList` as an example. These are in `/data/Projects/Slicer/Base/Python/slicer/parameterNodeWrapper/serializers.py`.

In your custom `Serializer` that you will write, you will need to override the `supportsCaching` method to return `True`. See the *Custom Classes* page for more information on creating a custom class and making new `Serializers`.

12.5 Modules API

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.

12.5.1 Colors

Color table CSV file format (.csv)

The color file format can store a color table node in a standard comma-separated value file with the `.csv` extension. Columns are separated by comma (`,`) character. Values that contain comma must be put between quotation marks (`"`). Due to a limitation in VTK library's delimited text reader/writer classes, values must not contain quotation mark characters.

Columns:

- `LabelValue`: label value that is associated with the color. Typically it is used for assigning color and meaning to a voxel value in a label image. It must be a positive integer. 0 value always corresponds to the background value and generally should not be included in the file (only in the rare case when the background needs to be colored). Label values do not have to be sorted. Label values may be skipped (left undefined), but since label values may be stored in a contiguous table from 0 to the largest label value, it is generally recommended to keep the range of values small (e.g., do not use label values such as 1000, 1001, 1002, 15000, 15001 but instead use the values 10, 11, 12, 20, 21).
- `Name`: internal (project-specific) name. It is only intended for convenience during development and testing. Content of a color table entry is intended to be described by standard terminology columns. Optional.
- `Color_R`, `Color_G`, `Color_B`, `Color_A`: red, green, blue, and opacity (alpha) component of the color. Values are integer, between 0 and 255. `Color_A` column is optional, if missing then the default value 255 (fully opaque) is used.
- Standard terminology columns for specifying content of the segment. The columns are optional and values in them can be empty. However, if there are non-empty values then `Category` and `Type` must be specified. For clinical imaging, it is recommended to use coding schemes defined for use in DICOM.
  - `Category_CodingScheme`, `Category_CodeValue`, `Category_CodeMeaning`: category, such as "anatomical structure" or "physical object"
  - `Type_CodingScheme`, `Type_CodeValue`, `Type_CodeMeaning`: type, such as "Kidney" or "Mass"

- TypeModifier_CodingScheme, TypeModifier_CodeValue, TypeModifier_CodeMeaning: type modifier, such as “Left” or “Right”
- Region_CodingScheme, Region_CodeValue, Region_CodeMeaning: region where the item is located in, such as “Kidney”
- RegionModifier_CodingScheme, RegionModifier_CodeValue, RegionModifier_CodeMeaning: region modifier, such as “Left” or “Right”

Example:

```
LabelValue,Name,Color_R,Color_G,Color_B,Color_A,Category_CodingScheme,Category_CodeValue,
→Category_CodeMeaning,Type_CodingScheme,Type_CodeValue,Type_CodeMeaning,TypeModifier_
→CodingScheme,TypeModifier_CodeValue,TypeModifier_CodeMeaning,Region_CodingScheme,
→Region_CodeValue,Region_CodeMeaning,RegionModifier_CodingScheme,RegionModifier_
→CodeValue,RegionModifier_CodeMeaning
1,left_kidney,185,102,83,255,SCT,123037004,Anatomical Structure,SCT,64033007,Kidney,SCT,
→7771000,Left,,,,,,
5,right_kidney,185,102,83,255,SCT,123037004,Anatomical Structure,SCT,64033007,Kidney,SCT,
→24028007,Right,,,,,,
6,right_kidney_mass,144,238,144,255,SCT,49755003,Morphologically Altered Structure,SCT,
→4147007,Mass,,,,SCT,64033007,Kidney,SCT,24028007,Right
10,catheter_renal_artery,127,127,127,255,SCT,260787004,Physical object,SCT,19923001,
→Catheter,,,,SCT,2841007,Renal artery,,,
```

Color table text file format (.txt, .ctbl)

The color file format can store a color node in a plain text file with the .txt or .ctbl extension. It is a text file with values separated by space, with a custom header to specify lookup table type. Header lines are prefixed with #.

Discrete scale color lookup table

Header starts with `# Color table file`. Each data line contains `color index` (integer), `color name` (string, if the name contains spaces then the spaces must be replaced by underscore), `red` (0-255), `green` (0-255), `blue` (0-255), and `opacity` (0-255).

Example:

```
# Color table file C:/Users/andra/OneDrive/Projects/SlicerTesting2022/20220109-
→ColorLegend/Segmentation-label_ColorTable.ctbl
# 4 values
0 Background 0 0 0 0
1 artery 216 101 79 255
2 bone 241 214 145 255
3 connective_tissue 111 184 210 255
```

Continuous scale color lookup table

Header starts with `# Color procedural file`. Each data line contains `position` (mapped value, a floating-point number), `red` (0.0-1.0), `green` (0.0-1.0), `blue` (0.0-1.0).

Example:

```
# Color procedural file /path/to/file.txt
# 5 points
# position R G B
```

(continues on next page)

(continued from previous page)

```
0 0 0 0
63 0 0.501961 0.490196
128 0.501961 0 1
192 1 0.501961 0
255 1 1 1
```

Debugging

Access scalar bar actor

Access to the scalar bar VTK actor may can be useful for debugging and for experimenting with new features. This code snippet shows how to access the actor in the Red slice view using Python:

```
displayableNode = getNode('Model')
colorLegendDisplayNode = slicer.modules.colors.logic().
↪GetColorLegendDisplayNode(displayableNode)
sliceViewLabel = "Red"
sliceViewNode = slicer.mrmlScene.GetNodeByID(f"vtkMRMLSliceNode{sliceViewLabel}")
appLogic = slicer.app.applicationLogic()
displayableManager = appLogic.GetViewDisplayableManagerByClassName(sliceViewNode,
↪"vtkMRMLColorLegendDisplayableManager")
colorLegendActor = displayableManager.GetColorLegendActor(colorLegendDisplayNode)

# Experimental adjustment of a parameter that is not exposed via the␣
↪colorLegendDisplayNode
colorLegendActor.SetBarRatio(0.2)
sliceView.forceRender()
```

12.5.2 DICOM (Digital Imaging and Communications in Medicine)

Refer to the *Slicer User Guide DICOM Section* for a comprehensive overview of the DICOM standard and related features in Slicer.

The overall DICOM Implementation in 3D Slicer consists of two main bodies of code embedded within the application.

- CTK code is responsible for the implementation of the DICOM database and DIMSE networking layer. The CTK code is implemented in C++ and follows the Qt style.
- The DICOM Module exposes CTK functionality to Slicer users, and provides hooks through which other modules can register DICOM Plugins to handle the conversion of specific DICOM data objects into the corresponding MRML representation. Once the data is in slicer, it can be operated on via the standard Slicer mechanisms.

Additional tools are available for Slicer Python scripted module development:

- DICOM Toolkit (DCMTK) command line application tools are installed with 3D Slicer and can be called synchronously within a Slicer scripted module.
- The DICOMweb Client (`dicomweb-client`) Python module is available within 3D Slicer for DICOMweb networking.
- Various Slicer-specific implementations for DICOM management and network are provided in the DICOMLib scripted module.

Customize table columns in DICOM browser

Columns in the browser can be renamed, reordered, shown/hidden, etc. An example snippet to customize the browser can be found in the *script repository*.

The changes made like this are written into the database (ColumnDisplayProperties table), so the column customizations can be done only once per database.

Customize DICOM browser table content

The way the raw DICOM tags are represented in the fields of the DICOM tables is determined by the displayed field generator rules. These rules are subclasses of the ctkDICOMDisplayedFieldGeneratorAbstractRule class, and need to be registered to the displayed field generator in order to take part of the generation.

The fields are updated when 1) files are added to the database or 2) the database schema is updated (happens when opening an older database with a newer Slicer). In the current version only those files (i.e. instances) are processed for which the displayed fields have never been generated.

When updating the displayed fields, every rule defines the fields it is responsible for using the cached DICOM tags in the database. Tags can be requested to be cached in the rules from the getRequiredDICOMTags function. New field values are generated by the rules instance by instance. First, the getDisplayedFieldsForInstance function is called for each rule in which the custom values are generated from the raw tags, then the results are merged by calling mergeDisplayedFieldsForInstance for all the rules. Each field can be requested to be merged with "expect same value", which uses the only non-empty value and throws a warning if conflicting values are encountered, or with "concatenate", which simply concatenates the displayed field values together.

The existing two rules can be used as examples: the default and the RT rules.

Plugin architecture

There are many different kind of DICOM information objects and import/export of all of them would not be feasible to implement in the Slicer core (see more information in the *DICOM module user manual*). Therefore, extensions can implement custom importer/exporter classes based on DICOMLib.DICOMPlugin class and add them to slicer.modules.dicomPlugins dictionary object. The DICOM module uses these plugins to determine list of loadable and exportable items.

DICOM Networking Development

3D Slicer leverages CTK, DCMTK, and `dicomweb-client` to provide DICOM networking capabilities. Medical DICOM images stored on a remote Picture and Archiving Communication Systems (PACS) can be queried, fetched to Slicer, annotated, and then stored back to the PACS.

We suggest that Slicer developers dealing with DICOM networking explore local test environment options to fit their needs.

- DCMTK provides tools for testing DIMSE networking. You can use DCMTK's `storescp.exe` tool distributed with 3D Slicer to test asynchronous store requests with DIMSE.

Run the following command from a shell prompt:

```
> "path/to/Slicer x.y.z\bin\storescp.exe" <port-number>
```

Then in Slicer, send DICOM data to the server at `localhost:<port-number>`.

- Several open source PACS platforms can be installed locally to support DICOMweb testing in an isolated environment, including:
  - Orthanc
  - DCM4CHE

References

See the CTK web site for more information on the internals of the DICOM implementation. This tool uses the DCMTK DICOM library.

Examples

Examples for common DICOM operations are provided in the *script repository*.

12.5.3 Markups

Markups json file format (.mrk.json)

All markups node types (point list, line, angle, curve, etc.) can be saved to and loaded from json files. Detailed specification of all elements of the file is available in the JSON schema.

A simple example that specifies a markups point list with 3 points that can be saved to a `myexample.mrk.json` file and loaded into Slicer:

```
{"@schema": "https://raw.githubusercontent.com/Slicer/Slicer/main/Modules/Loadable/
↪Markups/Resources/Schema/markups-schema-v1.0.0.json#",
"markups": [{"type": "Fiducial", "coordinateSystem": "LPS", "controlPoints": [
    { "label": "F-1", "position": [-53.388409961685827, -73.33572796934868, 0.0] },
    { "label": "F-2", "position": [49.8682950191571, -88.58955938697324, 0.0] },
    { "label": "F-3", "position": [-25.22749042145594, 59.255268199233729, 0.0] }
]}]}
```

Markups fiducial point list file format (.fcsv)

> **Warning**
>
> The fcsv file format is a legacy format that only stores control point coordinates and a limited set of display properties in a custom CSV-like file format. Support for this file format will be removed in future software versions.
>
> It is recommended to use *Markups JSON format* instead. Existing fcsv files can be converted using Slicer application (by loading from fcsv and saving as mrk.json) or - outside the 3D Slicer Python environment - using the slicerio Python package (see example).
>
> To save control point coordinates in standard CSV format, markups control point coordinates can be exported to a table node and save in standard CSV format.

vtkMRMLMarkupsFiducialStorageNode uses a comma separated value file with a custom header to store the control points on disk. A simple example:

```
# Markups fiducial file version = 4.13
# CoordinateSystem = LPS
# columns = id,x,y,z,ow,ox,oy,oz,vis,sel,lock,label,desc,associatedNodeID
0,-19.906699999999987,13.9347,29.442970822281154,0,0,0,1,1,1,0,F-1,,
1,-7.3939,-76.94990495817181,17.552540297898375,0,0,0,1,1,1,0,F-2,,
2,81.73332450520303,-42.9415,9.625586614976527,0,0,0,1,1,1,0,F-3,,
```

File header:

- Line 1: a comment line specifying the Slicer version that created the file
- Line 2: a comment line specifying the coordinate system (`CoordinateSystem = LPS` or `CoordinateSystem = RAS`). In earlier versions of Slicer, numeric codes were used: RAS = 0, LPS = 1.

- Line 3: a comment line explaining the fields in the csv (`columns = id,x,y,z,ow,ox,oy,oz,vis,sel,lock,label,desc,associatedNodeID`)

Each line after the header specifies a control point. Meaning of columns:

- id: a string giving a unique id for this control point, usually based on the class name
- x,y,z: the floating point coordinate of the control point
- ow,ox,oy,oz: the orientation quaternion of this control point, angle and axis, default 0,0,0,1 (or 0,0,0,0,0,0,1.0)
- vis: the visibility flag for this control point, 0 or 1, default 1
- sel: the selected flag for this control point, 0 or 1, default 1
- lock: the locked flag for this control point, 0 or 1, default 0
- label: the name for this control point, displayed beside the glyph, with quotes around it if there is a comma in the field
- desc: a string description for this control point, optional
- associatedNodeID = an id of a node in the scene with which the control point is associated, for example the volume or model on which the control point was placed, optional

Markups control points table file format (.csv, .tsv)

Markups control points can be imported from and exported to a table node that can be written to/read from standard comma (or tab) separated file format. A simple example:

```
label,l,p,s,defined,selected,visible,locked,description
F-1,-19.9067,13.9347,29.443,1,1,1,0,
F-2,-7.3939,-76.9499,17.5525,1,1,1,0,
F-3,81.7333,-42.9415,9.62559,1,1,1,0,
```

Definition of columns:

- label: label that is displayed next to each control point
- l, p, s: coordinate values in LPS coordinate system (in this case, l and a columns should not be used)
- r, a, s: coordinate values in RAS coordinate system (in this case, r and a columns should not be used)
- defined: 0 = the position of the point is not defined (coordinate values can be ignored; useful for creating templates); 1: the position is defined
- selected: 0 = unselected; 1 = selected (the control point appears with different colors an may be used as additional input for analysis)
- visible: 0 = hidden; 1 = visible
- locked: 0 = the point can be interactively moved; 1 = the point position is locked
- description: text providing additional information for the point

References

- History and design considerations

Examples

Examples for common operations on transform are provided in the *script repository*.

12.5.4 Plots

Design

MRML nodes

- vtkMRMLTableNode: Table node stores values that specify data point positions or bar heights in the plots.
- vtkMRMLPlotSeriesNode: Defines a data series by referring to a table node and column name(s) for X and Y axes and labels.
  - It also defines display properties, such as plot type, color, line style.
  - Line and bar plots only require Y axis (points along X axis are equally spaced), scatter plots require two input data columns, for X and Y axes.
- vtkMRMLPlotChartNode: Specifies which data series need to be shown in the chart.
  - Also contains global display properties for the chart, such as titles and font style.
- vtkMRMLPlotViewNode: Specifies which chart is to be displayed in the plot view and how the user can interact with it.
  - There has to be exactly one plot view node for each plot view widget. This class can not be created or copied unless is connected with a plot view.

Widgets

- qMRMLPlotView: Displays a plot. It can be embedded into a module user interface.
- qMRMLPlotWidget: Displays a plot and in a popup window a plot view controller widget.
- qMRMLPlotViewControllerWidget: plot view controller widget.
- qMRMLPlotSeriesPropertiesWidget: Display/edit properties of a plot series node.
- qMRMLPlotChartPropertiesWidget: Display/edit properties of a plot series node.
- qMRMLPlotViewControllerWidget: Display/edit properties of a plot view node.

Signals

qMRMLPlotView objects provide `dataSelected(vtkStringArray* mrmlPlotSeriesIDs, vtkCollection* selectionCol)` signal that allow modules to respond to user interactions with the Plot canvas. The signal is emitted when a data point or more has been selected. Returns the series node IDs and a list of selected point IDs (as a collection of `vtkIdTypeArray` objects).

Python API example:

```
# Switch to a layout that contains a plot view to create a plot widget
layoutManager = slicer.app.layoutManager()
layoutWithPlot = slicer.modules.plots.logic().GetLayoutWithPlot(layoutManager.layout)
layoutManager.setLayout(layoutWithPlot)

# Select chart in plot view
plotWidget = layoutManager.plotWidget(0)
plotViewNode = plotWidget.mrmlPlotViewNode()
```

(continues on next page)

(continued from previous page)

```
# Add a PlotCharNode
# plotViewNode.SetPlotChartNodeID("PlotChartNode".GetID())

# Print selected point IDs
def onDataSelected(mrmlPlotDataIDs, selectionCol):
    print("Selection changed:")
    for selectionIndex in range(mrmlPlotDataIDs.GetNumberOfValues()):
        pointIdList = []
        pointIds = selectionCol.GetItemAsObject(selectionIndex)
        for pointIndex in range(pointIds.GetNumberOfValues()):
            pointIdList.append(pointIds.GetValue(pointIndex))
        print("  {0}: {1}".format(mrmlPlotDataIDs.GetValue(selectionIndex), pointIdList))


# Connect the signal with a slot "onDataSelected"
plotView = plotWidget.plotView()
plotView.connect("dataSelected(vtkStringArray*, vtkCollection*)", self.onDataSelected)
```

Examples

Examples for common DICOM operations are provided in the *script repository*.

12.5.5 Self Tests

FAQ

Frequently asked questions about how to write and execute tests for Slicer.

How are the tests executed ?

There are two main mechanisms:

- On-demand execution using:
  - `Reload & Test` module panel section displayed for scripted modules when the *developer mode* is enabled in the application settings.
  - *Self Tests* module user interface.
- Automatic execution using:
  - CTest in the context of *Nightly tests*.

How are the tests discovered ?

To be discovered, scripted modules are expected to have the following:

- implement a test case class named `<ModuleName>Test` (itself deriving from *`slicer.ScriptedLoadableModule.ScriptedLoadableModuleTest`*).
- provide a function called `runTest()`.

These will ensure that the `runTest()` function is always discovered and executed.

Note

Scripted modules generated using the *Extension Wizard* already implement the expected test case.

12.5.6 Segmentations

Segmentation labelmap file format (.seg.nrrd)

Segmentation is stored in a standard NRRD image file with custom fields to store additional metadata.

The **slicerio Python package** can read .seg.nrrd files and import segments into numpy arrays and dictionary objects. The package can be used in any Python environment (installed using `pip install slicerio`) - not just in 3D Slicer.

Image data

If segments do not overlap then the file has 3 spatial dimensions. Even if a single-slice image is segmented, the spatial dimension must be still 3 because that is required for specification origin, spacing, and axis directions in 3D space. If segments overlap then the file has one `list` dimension and 3 spatial dimensions. Each 3D volume in the list is referred to as a `layer`.

If the image contains a single voxel then it has a special meaning: it means that no image data is available and the image geometry (origin, spacing, axis directions, extents) will be ignored. This special case is necessary because it is not possible to create a nrrd image file without any voxel data. Therefore, a segmentation without segments or with only empty segments is saved to file (e.g., to be used as a template for new segmentations) then it will be saved with an image data containing a single voxel.

Metadata

Additional metadata is stored in custom data fields (starting with `Segmentation_` or `SegmentN_` prefixes), which provide hints on how the segments should be displayed or what they contain.

Common custom fields

- `Segmentation_ContainedRepresentationNames`: names of segmentation representations (separated by `|` character) that should be displayed. Common representation names include `Binary labelmap`, `Closed surface`, `Planar contours`, `Fractional labelmap`.
- `Segmentation_ConversionParameters`: parameters for algorithms that compute requested representations. Each parameter definition is separated by the `&` character. A parameter definition consists of parameter name, value, and description text, separated by `|` character.
- `Segmentation_SourceRepresentation`: defines what representation is stored in the file. It is most commonly `Binary labelmap`, but there are other representations, too, which are stored in an image volume - such as `Fractional labelmap`.
- `Segmentation_ReferenceImageExtentOffset`: This field allows storing an image with arbitrary extent in a NRRD file. NRRD file only stores size of the volume. This field stores voxel coordinates (separated by spaces), therefore the extent can be computed as `[ReferenceImageExtentOffsetI, ReferenceImageExtentOffsetI+sizeI-1, ReferenceImageExtentOffsetJ, ReferenceImageExtentOffsetJ+sizeJ-1, ReferenceImageExtentOffsetK, ReferenceImageExtentOffsetk+sizeK-1]`.

Segment custom fields

These fields specify properties for each segment. `N` refers to the segment index, which is an integer between 0 and `NumberOfSegments - 1`.

- `SegmentN_ID`: identifier of the segment, it is unique within the segmentation. Can be used for unanmiguously refer to a segment even if the segment's display name is changed.
- `SegmentN_Name`: display name of the segment (the name that is shown to users).
- `SegmentN_NameAutoGenerated`: if value is `0` then it means that `SegmentN_Name` was chosen manually by the user, if value is 1 then it means that the segment's name is generated automatically (for example, determined from terminology).
- `SegmentN_Color`: recommended segment display color, defined with red, green, blue values (between 0.0 - 1.0) separated by space character.
- `SegmentN_ColorAutoGenerated`: if value is `0` then it means that `SegmentN_Color` was chosen manually by the user, if value is 1 then it means that the segment's color is generated automatically (for example, color defined based on terminology).
- `SegmentN_Extent`: 6 space-separated values [`minI`, `maxI`, `minJ`, `maxJ`, `minZ`, `maxZ`] defining extent of non-empty region within the segment.
- `SegmentN_Tags`: List of key-value pairs for storing additional information about the segment. Key and value is separated by the `:` character, pairs are separated from each other by the `|` character.
- `SegmentN_Layer`: Index of the 3D volume that contains the segment. Its value is between 0 and `DimensionAlongTheListImageAxis - 1`. For segmentations in which the segments do not overlap, the segmentation can be represented as a single 3D volume with 1 layer. Upon saving, segments are typically collapsed to as few layers as possible.
- `SegmentN_LabelValue`: The scalar value used to represent the segment within its own layer. Segments in separate layers can have the same label value.

TerminologyEntry tag

A frequently used key is `TerminologyEntry`, which defines what the segment contains using DICOM compliant terminology. Value stores 7 parts: `terminology context name`, `category`, `type`, `type modifier`, `region context name`, `region`, and `region modifier`. Parts are separated from each other by ~ character. Five of these parts - `category`, `type`, `type modifier`, `region`, and `region modifier` are defined by a triplet of (`coding scheme designator`, `code value`, and `code meaning`). Components of the triplet are separated by ^ character.

Example:

A mass in the right side of the adrenal gland would be encoded with the following the tag:

```
TerminologyEntry:
Segmentation category and type - 3D Slicer General Anatomy list
~SCT^49755003^Morphologically Altered Structure
~SCT^4147007^Mass
~^^
~Anatomic codes - DICOM master list
~SCT^23451007^Adrenal gland
~SCT^24028007^Right"
```

Interpretation:

- `terminology context name: Segmentation category and type - 3D Slicer General Anatomy list`

- `category`: `codingScheme=SCT` (Snomed Clinical Terms), `codeValue=49755003`, `codeMeaning=Morphologically Altered Structure`
- `type`: `codingScheme=SCT`, `codeValue=4147007`, `codeMeaning=Mass`
- `type modifier`: not specified
- `region context name`: `Anatomic codes - DICOM master list`
- `region`: `codingScheme=SCT`, `codeValue=23451007`, `codeMeaning=Adrenal gland`
- `region modifier`: `codingScheme=SCT`, `codeValue=24028007`, `codeMeaning=Right`

Segmentation surface file format (.seg.vtm)

Segmentation is stored in a standard VTK multiblock data set file format. Multiblock data set consists of an index (.vtm) file that stores the file path of each block. Each block file is a VTK polydata (.vtp) file, typically stored in a subdirectory.

Metadata

Metadata fields are stored in the polydata files (.vtp) as arrays in `FieldData`:

- `Segmentation_SourceRepresentation`: type: `String`, definition: see in "Common custom fields" section above
- `Segmentation_ConversionParameters`: type: `String`, definition: see in "Common custom fields" section above
- `Segmentation_ContainedRepresentationNames`: type: `String`, definition: see in "Common custom fields" section above
- `Segment_ID`: type = `String`, definition: see in "Segment custom fields" section above
- `Segment_Name`: type = `String`, definition: see in "Segment custom fields" section above
- `Segment_NameAutoGenerated`: type = `String`, definition: see in "Segment custom fields" section above
- `Segment_Color`: type = `Float64`, number of components = 3, definition: see in "Segment custom fields" section above
- `Segment_ColorAutoGenerated`: type = `String`, definition: see in "Segment custom fields" section above
- `Segment_Tags`: type = `String`, definition: see in "Segment custom fields" section above

12.5.7 Segment editor

See examples of using Segment editor effects from Python scripts in the *script repository*.

Effect parameters

Setting/getting effect parameters:

- Numeric or string parameter values: Common parameters must be set using `setCommonParameter(parameterName, parameterValue)` method, while effect-specific parameters can be set using `setParameter(parameterName, parameterValue)` method of the effect. Both common and effect-specific parameters can be retrieved using `parameter(parameterName)` method (for `string` or `enum` type), `integerParameter(parameterName)` (for `int` type), `doubleParameter(parameterName)` (for `float` type).
- Node reference (`noderef` type) parameter: It holds a reference to a node object, therefore it must be get/set by calling `nodeReference(parameterName)` and `setNodeReference(parameterName, node)` or `setCommonNodeReference(parameterName, node)` methods of the parameter set node.

Fill between slices

Grow from seeds

Hollow

Islands

Logical operators

Margin

Mask volume

Paint effect and Erase effect

Scissors

Smoothing

Threshold

Examples

Examples for common operations on segmentations are provided in the *script repository*.

12.5.8 Sequences

Volume sequence file format (.seq.nrrd)

Whenever it is possible (when origin, spacing, axis directions, and extents are the same in all sequence items), a sequence of volumes is stored in a standard NRRD image file with custom fields to store additional metadata.

Image data

The image is always stored in 3 spatial dimensions. Even if an image consists of a single slice, the spatial dimension must be still 3 because that is required for specification origin, spacing, and axis directions in 3D space. The file has an additional `list` dimension.

Metadata

Standard fields

- `dimension`: `4`
- `sizes`: image size (I, J, K) followed by the number of time points
- `space directions`: I, J, K axis directions in physical space followed by `none`
- `kinds`: `domain domain domain list`
- `labels`: `"" "" ""` followed by name of the sequence index (e.g., `"time"`)
- other standard fields (`type`, `space`, `endian`, `encoding`, `space origin`, etc.) can have any value

Custom fields

The sequence-specific metadata is stored in NRRD key–value pairs that follow this pattern:

- axis <A> index type:=<IndexType>
- axis <A> index values:=<v0> <v1> ... <vN-1>
- axis <A> item <I> <AttrName>:=<AttrValue>
- DataNodeClassName:=<MRMLNodeClass>

Where:

- <A> is the **list axis index**. In Slicer-generated .seq.nrrd files the list axis is **3**, because sizes are ordered as I, J, K, list.
- <IndexType> is either numeric or text and determines how index values are interpreted (e.g., numeric vs. lexicographic sorting).
- <v0> ... <vN-1> are the **per-item index values** along the list axis (must have as many values as there are list items). These are stored here rather than using standard NRRD spacing because sequence indices may be **non-uniform** or **non-numeric**.
- <I> is the **0-based item index** along the list axis.
- <AttrName> / <AttrValue> store **per-item node attributes** for the data nodes in the sequence.
- <MRMLNodeClass> is the MRML class name of the proxy node to instantiate when reading the sequence (e.g., vtkMRMLScalarVolumeNode; for labelmaps use vtkMRMLLabelMapVolumeNode).

Field definitions

- axis <A> index type:=numeric|text Declares the index value type for the list axis (affects sorting and parsing).
- axis <A> index values:=<v0> <v1> ... <vN-1> Provides the explicit index value for each sequence item, separated by spaces. Values can be arbitrary strings, stored using URL encoding.
- axis <A> item <I> <AttrName>:=<AttrValue> Stores arbitrary per-item attributes from the sequence's proxy nodes (e.g., acquisition time, frame UID).
- DataNodeClassName:=<MRMLNodeClass> Specifies the MRML node class to create for items in this sequence. Common values include:
  - vtkMRMLScalarVolumeNode (default)
  - vtkMRMLLabelMapVolumeNode (labelmaps)

Note

- Slicer writes the list axis as **axis 3** and sets kinds: domain domain domain list.
- The count of index values **must match** the size of the list axis (sizes[3] in Slicer files).
- Use numeric when index values are intended to be ordered by numeric value (e.g., frame number, time in seconds). Use text for non-numeric or mixed tokens (e.g., "pre" "post" "followup1").

Example

```
NRRD0004
# Complete NRRD file format specification at:
# http://teem.sourceforge.net/nrrd/format.html
type: short
dimension: 4
space: left-posterior-superior
sizes: 128 104 72 10
space directions: (1.1499999999999999,0,0) (0,1.1499999999999999,0) (0,0,2) none
kinds: domain domain domain list
labels: "" "" "" "frame"
endian: little
encoding: gzip
space origin: (-62.200000000000001,-68,1692.0989989999998)
DataNodeClassName:=vtkMRMLScalarVolumeNode
axis 3 index type:=numeric
axis 3 index values:=0 1 2 3 4 5 6 7 8 9
# Example per-item attribute (item 5)
axis 3 item 5 AcquisitionTime:=2024-06-21T10:32:45.120Z
```

12.5.9 Transforms

Related MRML nodes

- vtkMRMLTransformableNode: any node that can be transformed
- vtkMRMLTransformNode: it can store any linear or deformable transform or composite of multiple transforms
  - vtkMRMLLinearTransformNode: Deprecated. The transform does exactly the same as `vtkMRMLTransformNode` but has a different class name, which are still used for showing only certain transform types in node selectors. In the future this class will be removed. A `vtkMRMLLinearTransformNode` may contain non-linear components after a non-linear transform is hardened on it. Therefore, to check linearity of a transform the vtkMRMLTransformNode::IsLinear() and vtkMRMLTransformNode::IsTransformToWorldLinear() and vtkMRMLTransformNode::IsTransformToNodeLinear() methods must be used instead of using `vtkMRMLLinearTransformNode::SafeDownCast(transform) != nullptr`.
  - vtkMRMLBSplineTransformNode: Deprecated. The transform does exactly the same as `vtkMRMLTransformNode` but has a different class name, which are still used for showing only certain transform types in node selectors. In the future this class will be removed.
  - vtkMRMLGridTransformNode: Deprecated. The transform does exactly the same as `vtkMRMLTransformNode` but has a different class name, which are still used for showing only certain transform types in node selectors. In the future this class will be removed.

Transform files

- Slicer stores transforms in VTK classes in memory, but uses ITK transform IO classes to read/write transforms to files. ITK's convention is to use LPS coordinate system as opposed to RAS coordinate system in Slicer (see the *Coordinate systems* page for details). Conversion between VTK and ITK transform classes are implemented in vtkITKTransformConverter.
- ITK stores the transform in resampling (a.k.a., image processing) convention, i.e., that transforms points from fixed to moving coordinate system. This transform is usable as is for resampling a moving image in the coordinate system of a fixed image. For transforming points and surface models to the fixed coordinate system, one needs the

transform in the modeling (a.k.a. computer graphics) convention, i.e., transform from moving to fixed coordinate system (which is the inverse of the "image processing" convention).

- Transform nodes in Slicer can store transforms in both modeling (when "to parent" transform is set) and resampling way (when "from parent" transform is set). When writing transform to ITK files, linear transforms are inverted as needed and written as an `AffineTransform`. Non-linear transforms cannot be inverted without losing information (in general), therefore if a non-linear transform is defined in resampling convention in Slicer then it is written to ITK file using special "Inverse" transform types (e.g., `InverseDisplacementFieldTransform` instead of `DisplacementFieldTransform`). Definition of the inverse classes are available in vtkITKTransformInverse. The inverse classes are only usable for file IO, because currently ITK does not provide a generic inverse transform computation method. Options to manage inverse transforms in applications:
  - Create VTK transforms from ITK transforms: VTK transforms can compute their inverse, transform can be changed dynamically, the inverse will be always updated automatically in real-time (this approach is used by Slicer)
  - Invert transform in ITK statically: by converting to displacement field and inverting the displacement field; whenever the forward transform changes, the complete inverse transform has to be computed again (which is typically very time consuming)
  - Avoid inverse non-linear transforms: make sure that non-linear transforms are only set as FromParent
- Transforms module in Slicer shows linear transform matrix values "to parent" (modeling convention) in RAS coordinate system. Therefore to retrieve the same values from an ITK transforms as shown in Slicer GUI, one has switch between RAS/LPS and modeling/resampling. See example *here*.

Displacement field files

Slicer can store displacement field transforms (or "grid transforms" in VTK naming convention) in 3D NRRD or NIFTI format; or sequence of transforms in NRRD format. Image file header fields are used to distinguish the displacement field transforms form regular images and specify how to correctly interpret these images as transforms.

NRRD

Required `kinds`:

- `vector domain domain domain` for a single 3D displacement field
- `vector domain domain domain list` for a **sequence** of displacement fields

Required custom `intent_code`:

- `intent_code := 1006` Mirrors the NIfTI convention `NIFTI_INTENT_DISPVECT`, allowing ITK readers to detect displacement fields reliably.

Recommended `measurement frame`:

- `measurement frame: (1,0,0) (0,1,0) (0,0,1)` Use identity to state unambiguously that vector components are spatial and expressed in world space.

Optional usage hint (`displacement field type`):

- `displacement field type := resampling | modeling`
  - `resampling` (default): Interpret as a **resampling** transform (ITK convention, fixed to moving). In Slicer's transform tree this corresponds to **"from parent"**.
  - `modeling`: Interpret as a **modeling** transform (useful for points/markups/models, moving to fixed). In Slicer's transform tree this corresponds to **"to parent"**.

Per-item attributes for sequences (custom fields): Use these only when `kinds` includes a `list` axis.

- axis <A> index type := numeric | text Declares the type of the list-axis index values (affects parsing/sorting).
- axis <A> index values := <v0> <v1> ... <vN-1> Provides one index value per sequence item (values may be non-uniform or non-numeric). Provides the explicit index value for each sequence item, separated by spaces. Values can be arbitrary strings, stored using URL encoding.
- axis <A> item <I> <AttrName> := <AttrValue> Stores an arbitrary per-item attribute (becomes a node attribute on the corresponding MRML data node).

Where:

- <A> = the **list axis index** (the axis whose `kinds` entry is `list`)
- <I> = **0-based** item index along the list axis
- Quote values that contain spaces per NRRD rules
- The count of `index values` **must equal** the size of the list axis

> **Note**
>
> In 3D + time sequences the vector axis is typically 0, followed by three spatial `domain` axes, and the `list` axis last. Do not assume a fixed number—always use the axis where `kinds` is `list` for <A>.

Example 3D displacement field file header

```
NRRD0005
# Complete NRRD file format specification at:
# http://teem.sourceforge.net/nrrd/format.html
type: double
dimension: 4
space: left-posterior-superior
sizes: 3 256 256 130
space directions: none (0,1,0) (0,0,-1) (-1.2999954223632812,0,0)
kinds: vector domain domain domain
endian: little
encoding: raw
space origin: (86.644897460937486,-133.92860412597656,116.78569793701172)
measurement frame: (1,0,0) (0,1,0) (0,0,1)
displacement field type:=resampling
intent_code:=1006
```

Example 3D+t displacement field sequence file header

```
NRRD0005
# Complete NRRD file format specification at:
# http://teem.sourceforge.net/nrrd/format.html
type: double
dimension: 5
space: left-posterior-superior
sizes: 3 88 160 72 14
space directions: none (0,1.5,0) (0.13855585501996717,0,-1.4935870497027235) (-1.4935870497027182,0,-0.13855585501997181) none
```

(continues on next page)

(continued from previous page)

```
kinds: vector domain domain domain list
labels: "" "" "" "" "time"
units: "" "" "" "" "ms"
endian: little
encoding: gzip
space origin: (43.348771859606991,-103.34095001221,82.564098750818999)
measurement frame: (1,0,0) (0,1,0) (0,0,1)
axis 3 index type:=numeric
axis 3 index values:=18.551218032837 55.653656005859 92.756088256836 129.8585357666 166.
↪96096801758 204.06340026855 241.16584777832 278.26824951172 315.37069702148 352.
↪47314453125 389.57556152344 426.6780090332 463.78045654297 500.88287353516
displacement field type:=modeling
intent_code:=1006
```

The 3 in `axis 3 index type` and `axis 3 index values` fields refer to the 4th range axis (list).

NIfTI

- `intent_code` **must** be `1006` (`NIFTI_INTENT_DISPVECT`)
- Displacement vector components are stored in the **RAS** coordinate system

Events

When a transform node is observed by a transformable node, vtkMRMLTransformableNode::TransformModifiedEvent is fired on the transformable node at observation time. Anytime a transform is modified, vtkCommand::ModifiedEvent is fired on the transform node and vtkMRMLTransformableNode::TransformModifiedEvent is fired on the transformable node.

Examples

Examples for common operations on transform are provided in the *script repository*.

12.5.10 Volume rendering

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
- vtkMRMLMarkupsROINode controls the clipping planes
- vtkMRMLVolumeRenderingDisplayableManager responsible for adding VTK actors to the renderers

Format of .vp.json (volume property JSON)

Since Slicer version 5.9, volume rendering properties are stored in JSON format by default, with the file extension `.vp.json`. This format is self-describing and easier to interpret than the legacy text file format. Details are specified by this JSON schema: https://raw.githubusercontent.com/slicer/slicer/main/Modules/Loadable/VolumeRendering/Resources/Schema/volume-property-schema-v1.0.0.json

Example `.vp.json` file:

```
{
    "@schema": "https://raw.githubusercontent.com/slicer/slicer/main/Modules/Loadable/
↪VolumeRendering/Resources/Schema/volume-property-schema-v1.0.0.json#",
    "volumeProperties": [
        {
            "effectiveRange": [0.0, 220.0],
            "components": [
                {
                    "shade": true,
                    "lighting": {
                        "diffuse": 1.0,
                        "ambient": 0.2,
                        "specular": 0.0,
                        "specularPower": 1.0
                    },
                    "rgbTransferFunction": {
                        "type": "colorTransferFunction",
                        "points": [
                            { "x": 0.0, "color": [0.0, 0.0, 0.0] },
                            { "x": 20.0, "color": [0.168627, 0.0, 0.0] },
                            { "x": 40.0, "color": [0.403922, 0.145098, 0.0784314] },
                            { "x": 40.0, "color": [0.403922, 0.145098, 0.0784314] },
                            { "x": 120.0, "color": [0.780392, 0.607843, 0.380392] },
                            { "x": 220.0, "color": [0.847059, 0.835294, 0.788235] },
                            { "x": 1024.0, "color": [1.0, 1.0, 1.0] }
                        ]
                    },
                    "scalarOpacityUnitDistance": 1.0,
                    "scalarOpacity": {
                        "type": "piecewiseLinearFunction",
                        "points": [
                            { "x": 0.0, "y": 0.0 },
                            { "x": 120.0, "y": 0.3 },
                            { "x": 220.0, "y": 0.375 },
                            { "x": 1024.0, "y": 0.5 }
                        ]
                    },
                    "gradientOpacity": {
                        "type": "piecewiseLinearFunction",
                        "points": [
                            { "x": 0.0, "y": 1.0 },
                            { "x": 255.0, "y": 1.0 }
                        ]
                    }
                }
            ]
        }
    ]
}
```

Format of Volume Property (.vp) file

Legacy text file format for storing volume properties. It is only supported for backward compatibility and it is recommended to use the .vp.json file format instead.

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 -86.9767 0 45.3791 0.169643 139.919 0.589286 347.907 0.607143 1224.16 0.
↪607143 3071 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 -86.9767 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 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)
```

Examples

Examples for common operations on transform are provided in the *script repository*.

Volume rendering presets

Volume rendering presets that are bundled in Slicer core are specified in a preset file and corresponding icon is stored as an application resource.

To add a new volume rendering preset:

- Add a new entry to presets.xml
- Add a corresponding icon of 128x100 pixels into presets icons folder.
- Add the icon to qSlicerVolumeRenderingModule.qrc

Presets can be also added at runtime - see *example in the script repository*. In this case, icons are loaded from the preset scene file.

12.5.11 Volumes

Examples

Examples for common operations on volumes are provided in the *script repository*.

12.6 Extensions

Developers can create extensions to provide additional features to users. See an overview of extensions in the *Extensions manager page*.

12.6.1 Create an extension

If you have developed a script or module that you would like to share with others then it is recommended to submit it to the Slicer Extensions Index. Indexed extensions get listed in the Extensions Manager in Slicer and user can install them by a few mouse clicks.

- Scan through the *user* and developer extension FAQs
- Inform a community about your plans on the Slicer forum 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 you have not done already, use the *Extension Wizard* module in Slicer to create an extension that will contain your module(s).
- If developing *C++ loadable or CLI modules* (not needed if developing in Python):
  - *build Slicer application*.
  - *build your extension*

12.6.2 Build an extension

Note

To compile extensions containing C++-implemented modules, it's essential to *build Slicer from source* on your local machine. These extensions cannot be compiled against a binary download. However, if your modules are solely developed in Python, there's no need to build the extension.

Once your work is ready for sharing, refer to *Distribute an extension* for guidelines.

Similarly to the building of Slicer core, multi-configuration builds are not supported: one build tree can be only used for one build mode (Release or Debug or RelWithDebInfo or MinSizeRel). If a release and debug mode build are needed then the same source code folder can be used (e.g., `C:\D\SlicerHeart`) but a separate binary folder must be created for each build mode (e.g., `C:\D\SlicerHeart-R` and `C:\D\SlicerHeart-D` for release and debug modes).

Assuming that the source code of your extension is located in folder `MyExtension`, an extension can be built by the following steps.

Tip

For testing purposes, it is possible to force the Slicer revision associated with the extension build by setting the `Slicer_REVISION` environment variable before configuring the project:

```
$ cd MyExtension-debug
$ export Slicer_REVISION=31806
$ cmake -DCMAKE_BUILD_TYPE:STRING=Debug -DSlicer_DIR:PATH=/path/to/Slicer-SuperBuild-
↪Debug/Slicer-build ../MyExtension
[...]
-- SlicerConfig: Forcing Slicer_REVISION to '31806'
[...]
-- Configuring done
-- Generating done
-- Build files have been written to: /path/to/MyExtension-debug
```

Linux and macOS

Start a terminal.

```
$ mkdir MyExtension-debug
$ cd MyExtension-debug
$ cmake -DCMAKE_BUILD_TYPE:STRING=Debug -DSlicer_DIR:PATH=/path/to/Slicer-SuperBuild-
↪Debug/Slicer-build ../MyExtension
$ make
```

CMAKE_OSX_ variables

On macOS, the extension must be configured specifying `CMAKE_OSX_*` variables matching the one used to configure Slicer: `-DCMAKE_OSX_ARCHITECTURES:STRING=x86_64 -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=/same/as/Slicer -DCMAKE_OSX_SYSROOT:PATH=SameAsSlicer`

Instead of manually setting these variables, within your extension, including the ConfigurePrerequisites component before the project statement should ensure it uses the same CMAKE_OSX_* variables as Slicer:

```
find_package(Slicer COMPONENTS ConfigurePrerequisites REQUIRED)

project(Foo)

# [...]

find_package(Slicer REQUIRED)
include(${Slicer_USE_FILE})

# [...]
```

For more details, see here.

Windows

Run `CMake (cmake-gui)` from the Windows Start menu.

- Select source and build directory
- Click `Configure`
- Select generator (just accept the default if you only have one compiler toolset installed)
- Choose to create build directory if asked
- The configuration is expected to display an error message due to `Slicer_DIR` variable not specified yet.
- Specify `Slicer_DIR` by replacing `Slicer_DIR-NOTFOUND` by the Slicer inner-build folder (for example `c:/D/SD/Slicer-build`).
- Click `Configure`. No errors should be displayed.
- Click `Generate` button.
- Click `Open project` button to open `MyExtension.sln` in Visual Studio.
- Select build configuration (Debug, Release, . . .) that matches the build configuration (Release, Debug, . . .) of the chosen Slicer build.
- In the menu choose Build / Build Solution.

12.6.3 Test an extension

Run Slicer with your custom modules

To test an extension, you need to specify location(s) where Slicer should look for additional modules.

- If the extension is not built: append all source code folders that contain module `.py` files to "additional module paths".
  - Option A (recommended): Drag-and-drop the `.py` files (or the parent folder, or that folder's parent) to the application window, select `Add Python scripted modules to the application` in the popup window, click OK. Select which modules to add to load immediately and click `Yes`. The selected modules will be immediately loaded and made available in the module list. If you want to load the modules only in the current session, then uncheck `Add selected modules to 'Additional module paths'` option before clicking `Yes`.
  - Option B: In menu: Edit / Application settings / Modules panel, drag-and-drop files to the `Additional module paths` list.
- If the extension is built:
  - Option A: start the application using the `SlicerWithMyExtension` executable in your build directory. This starts Slicer, specifying additional module paths via command-line arguments.
  - Option B: specify additional module paths manually in application settings. Assuming your extension has been built into folder `MyExtension-debug`, add these module folders (if they exist) to the additional module paths in Slicer's application settings:
    * `C:\path\to\MyExtension-debug\lib\Slicer-4.13\qt-scripted-modules`
    * `C:\path\to\MyExtension-debug\lib\Slicer-4.13\qt-loadable-modules\Debug` or `C:\path\to\MyExtension-debug\lib\Slicer-4.13\qt-loadable-modules\Release` or (on systems where multi-configuration builds are not used, such as linux) simply `C:\path\to\MyExtension-debug\lib\Slicer-4.13\qt-loadable-modules`:
    * `C:\path\to\MyExtension-debug\lib\Slicer-4.13\cli-modules\Debug`, `C:\path\to\MyExtension-debug\lib\Slicer-4.13\cli-modules\Release`, or `C:\path\to\MyExtension-debug\lib\Slicer-4.13\cli-modules`

Run automatic tests

Automatic tests of your extension can be launched by following the instructions below.

Linux and macOS

Start a terminal.

```
$ ctest -j<NUMBEROFCORES>
```

Windows

Run all tests

Open a command prompt.

```
cd C:\path\to\MyExtension-debug
"c:\Program Files\CMake\bin\ctest.exe" -C Release -V
```

Replace `Release` with the build mode of your extension build (`Debug`, `Release`, …).

To debug a test

- Launch Visual Studio from the Command Line Prompt: `C:\D\SD\Slicer-build\Slicer.exe --VisualStudio --launcher-no-splash --launcher-additional-settings C:\path\to\MyExtension-debug\AdditionalLauncherSettings.ini C:\path\to\MyExtension-debug\MyExtension.sln`
- Find the project of the test you want to debug (e.g., `qSlicerMODULE_NAMEModuleGenericCxxTests`).
- Go to the project debugging properties (right-click -> Properties, then Configuration Properties / Debugging).
- In `Command Arguments`, type the name of the test you want to run (e.g., `qSlicerMODULE_NAMEModuleGenericTest`).
- If the test takes arguments, enter the arguments after the test name in `Command Arguments`.
- Set the project as the StartUp Project (right-click -> Set As StartUp Project).
- Start debugging (F5).

Continuous integration

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

- Dashboard for Slicer Stable Releases
- Dashboard for Slicer Preview Releases

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

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

`https://slicer.cdash.org/index.php?project=SlicerStable&filtercount=1&showfilters=1&field1=buildname&co`

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

`https://slicer.cdash.org/index.php?project=SlicerStable&filtercount=1&showfilters=1&field1=buildname&co`

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.

12.6.4 Create an extension package

Assuming your extension has been built into folder `MyExtension-release` (redistributable packages must be built in release mode), this could be achieved doing:

Linux and macOS

Start a terminal.

```
$ make package
```

Windows

- Open `MyExtension.sln` in Visual Studio.
- Right-click on `PACKAGES` project, then select `Build`.

12.6.5 Write documentation for an extension

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

Thumbnails to YouTube videos can be generated by using an URL of the form `https://img.youtube.com/vi/<insert-youtube-video-id-here>/0.jpg` and adding a playback button using this free service (the second red arrow is recommended).

12.6.6 Distribute an extension

- 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 an extension that contains *C++ loadable or CLI modules* (not needed if developing in Python):
  - Build the `PACKAGE` target to create a package file.
  - Test your extension by installing the created package file using the Extensions Manager.
- Complete the extension submission checklist) then submit it to the Slicer Extensions Index:
- Submit the extension to the Extensions Index:
  - Fork ExtensionIndex repository on GitHub by clicking ''Fork" button on the Slicer Extensions Index page
  - Create an *extension catalog entry file (json)*
  - Add your `.json` file to your forked repository: it can be done using a git client or simply by clicking ''Upload files" button
    * To make the extension appear in the latest Slicer Preview Release: upload the file into the `main` branch.
    * To make the extension appear in the latest Slicer Stable Release: upload the file into the branch corresponding to the stable release version, for example: `4.10`.
  - Create a pull request: by clicking ''Create pull request" button
  - Follow the instructions in the pull request template

Tip

Once your extension is successfully integrated into the Extensions Index, you gain access to daily build and test reports for verification. For additional information, refer to *Continuous integration*.

While it is possible to develop Python-based extensions without a *local extension build*, creating one can significantly expedite the development cycle. This allows you to promptly confirm that the extension can be properly packaged and distributed, providing a quicker feedback loop compared to waiting for the daily build and test reports.

12.6.7 Application settings

After installing an extension, the directories are added to revision-specific settings:

- Folders containing modules bundled within an extension are added to Modules / AdditionalPaths. This ensures that libraries associated with modules are found.
- Folders containing third-party dynamic libraries, Python libraries, etc. are added to LibraryPaths, Paths, PYTHONPATH, QT_PLUGIN_PATH. This ensures that libraries associated with modules can be successfully loaded.

12.6.8 Extension catalog entry file

An extension catalog entry file is a JSON file describing how to build and package a Slicer extension.

It is named after the extension name (e.g `ExtensionName.json`) and may include the following entries:

Nam	Description	Required
category	Extension category.	Y
scm_	Read-only url used to checkout the extension source code.	Y
scm_	Revision allowing to checkout the expected source code.	N
scm_	Type of revision control system. Default to `git`.	N
build	List of extensions required to build this extension. For example: `["extensionA", "extensionB"]`.	N
dicom_	Rule engine expression to determine when the extension should be suggested for DICOM data based on DICOM attributes. The expression is evaluated using the rule-engine library. DICOM attributes used include `Modality` (e.g., 'PT', 'MR', 'CT', 'SEG') and `SOPClassUID`. For example: `"Modality == 'PT' or Modality == 'RWVM'"` for PET imaging, or `"Modality == 'SEG'"` for segmentation objects. Complex expressions can combine multiple conditions: `"(Modality == 'SR' and SOPClassUID == '1.2.840.10008.5.1.4.1.1.88.22') or Modality == 'M3D'"`.	N
build	Name of the inner build directory in case of superbuild based extension. Default to `.`.	N
enabled	Specify if the extension should be enabled after its installation. Valid values are `true` (default) = enabled; `false` = disabled.	N

The full schema for the extension catalog entry file is below:

12.6.9 Extension description file

Warning

This file is used internally by *Extensions build system* and it has been superseded by *Extension catalog entry file*.

An extension description file is a text file with `s4ext` extension allowing to specify metadata associated with an extensions.

The description file is automatically generated by the build system in the build tree of the extension from the metadata specified in the top-level `CMakeLists.txt` file in the source code of the extension. Since the description is a very simple text file, the description can be created using a text editor. It may be simpler to use a text editor to create the file if the extension only contains scripted modules, which can be developed without building the extension.

Note that the Extension manager ignores many fields of the extension description file and instead uses information specified in `CMakeLists.txt`. Therefore, when making any changes to the extension description file, it has to be done in the `CMakeLists.txt` file as well.

For superbuild-type extensions (that build their own dependencies, such as external libraries implemented in C++), it is critical that the build_subdirectory is initialized to the inner build location in s4ext. The value of this variable in `CMakeLists.txt` is not used in all places by the dashboard scripts.

Syntax

- A metadata is specified using a keyword followed by at least one spaces and the associated value.
- Multiline values are not supported.
- Empty lines are ignored
- Lines starting with a # are considered comments and ignored.

The following code block illustrates how comments, metadata and associated value can be specified:

```
# This is a comment
metadataname This is the value associated with 'metadataname'

# This is an other comment
anothermetadata This is the value associated with 'anothermetadata'
```

Supported metadata fields

Name	Description	Required
scm	Source code management system. Must be `git`.	Y
scmurl	Read-only url used to checkout the extension source code.	Y
scmrevision	Revision allowing to checkout the expected source code.	Y
depends	List of extensions required to build this extension. Specify "NA" if there are no dependency. Extension names should be separated by a single space. For example: extensionA extensionB.	N
build_subd	Name of the inner build directory in case of superbuild based extension. Default to `.`.	N
homepage	URL of the web page describing the extension.	Y
contributors	Extension contributor specified as `Firstname1 Lastname1 ([SubOrg1, ]Org1), Firstname2 Lastname2 ([SubOrg2, ]Org2)`.	N
category	Extension category.	Y
iconurl	URL to an icon (png file format and size of 128x128 pixels is recommended).	N
description	One line describing what is the purpose of the extension.	N
screenshoturls	Space separated list of urls to images.	N
enabled	Specify if the extension should be enabled after its installation. Valid values are 1 (default) = enabled; `0` = disabled.	N
status	Give people an idea what to expect from this code. This is currently not used.	N

Note: Parameters in URLS (such as `&foo=bar`) are not supported. URL shortener services can be used if necessary.

12.6.10 Extensions server

The official Slicer extensions server, the "Extensions Catalog" is available at https://extensions.slicer.org/. To get a list of extensions, specify the Slicer revision and platform in the URL, for example: https://extensions.slicer.org/catalog/All/30117/win.

The Extensions Catalog is implemented a web application (source code), which connects to a Girder server, a general-purpose storage server with the Slicer Package Manager plugin (source code), which provides a convenient REST API for accessing Slicer extension packages and metadata.

The "Manage extensions" tab in the Extensions Manager in Slicer uses this REST API to get information on updates and get packages to install or update.

The extension server is designed so that organizations can set up and maintain their own extensions servers, for example to distribute extensions for custom applications. Extensions server address can be set in the Application Settings, in the Extensions section.

Until August 2021, a Midas-based server at `https://slicer.kitware.com/midas3` was used. This server is not online anymore, as it was not feasible to perform all software updates that would have kept it secure.

12.6.11 Extensions Index

The ExtensionsIndex is a repository containing a list of *extension catalog entry files* `*.json` used by the Slicer *Extensions build system* to build, test, package and upload extensions on the *extensions server*.

The ExtensionsIndex is hosted on GitHub: https://github.com/Slicer/ExtensionsIndex

Each branch of the repository contains extension description files that corresponds to the same branch in the Slicer repository. For example, `main` branch contains descriptions for Slicer `main` branch, and `4.11` branch contains extension descriptions for Slicer's `4.11` branch.

Extension developers have to make sure that the extension description in each branch of the Extensions index is compatible with the corresponding Slicer version. Extension developers often create the same branches (`main`, `4.11`, `4.13`, ...) in their repository and they specify this branch name in the extensions descriptor file.

12.6.12 Extensions build system

The extensions build system allows to drive the build, test, packaging and upload of Slicer extensions.

Using the extensions build system source code, it is possible to build extensions using either manual build or dashboard-driven automatic build. The extension description files must be simply placed in a folder, the same way as they are in the Extensions Index repository.

Build list of extensions manually

Locally building a list of extensions is a convenient way to test building of extensions and get extension packages for a custom-build Slicer application.

Given a directory containing one or more extension description files, with the help of the extensions build system it is possible to configure and build the associated extensions specifying the following CMake options:

CMake variable	Description
Slicer_DIR	Path to Slicer build tree. Required.
Slicer_EXTENSIOI	Path to folder containing extension description files. Required.
CMAKE_BUILD_T	Build type of the associated Slicer build directory
CTEST_MODEL	By default set to `Experimental`. Allow to choose on which CDash track results are submitted as well as setting the submission type associated with the uploaded extension.
Slicer_UPLOAD_E	By default set to `OFF`. If enabled, extension builds will be submitted to Slicer dashboard and associated packages will be uploaded to extensions server.
SLICER_PACKAGI	Slicer extensions server URL specifying where the extension should be uploaded. For example `https://slicer-packages.kitware.com`. source code. Note that this variable is expected to be set in place of `MIDAS_PACKAGE_URL`.
SLICER_PACKAGI	Token allowing to authenticate to the extensions server. Note that this variable is expected to be set in place of `MIDAS_PACKAGE_API_KEY` and `MIDAS_PACKAGE_EMAIL`.

The following folders will be used in the examples below:

Build, test, and package

Linux and macOS:

```
cd ~/ExtensionsIndex-Release

cmake -DSlicer_DIR:PATH=~/Slicer-SuperBuild-Release/Slicer-build \
 -DSlicer_EXTENSION_DESCRIPTION_DIR:PATH=~/ExtensionsIndex \
 -DCMAKE_BUILD_TYPE:STRING=Release \
 ~/Slicer/Extensions/CMake

make
```

Windows:

```
cd /d C:\D\ExtensionsIndexR

"c:\Program Files\CMake\bin\cmake.exe" -DSlicer_DIR:PATH=C:/D/SR/Slicer-build ^
 -DSlicer_EXTENSION_DESCRIPTION_DIR:PATH=C:/D/ExtensionsIndex ^
 -DCMAKE_BUILD_TYPE:STRING=Release ^
 C:/D/S/Extensions/CMake

"c:\Program Files\CMake\bin\cmake.exe" --build . --config Release
```

Build, test, package, and upload to extensions server

Submit the configure/build/test results to the Slicer Dashboard `Extensions-Experimental` track and upload the extension to a custom Extensions Server.

Linux and macOS:

```
cd ~/ExtensionsIndex-Release
cmake -E env \
  SLICER_PACKAGE_MANAGER_CLIENT_EXECUTABLE=/path/to/slicer_package_manager_client \
  SLICER_PACKAGE_MANAGER_URL=https://slicer-packages.kitware.com \
  SLICER_PACKAGE_MANAGER_API_KEY=a0b012c0123d012abc01234a012345a0 \
```

(continues on next page)

(continued from previous page)

```
  \
cmake -DSlicer_DIR:PATH=~/Slicer-SuperBuild-Release/Slicer-build \
 -DSlicer_EXTENSION_DESCRIPTION_DIR:PATH=~/ExtensionsIndex \
 -DCMAKE_BUILD_TYPE:STRING=Release \
 -DCTEST_MODEL:STRING=Experimental \
 -DSlicer_UPLOAD_EXTENSIONS:BOOL=ON \
 ~/Slicer/Extensions/CMake

make
```

Windows:

```
cd /d C:\D\ExtensionsIndexR
"c:\Program Files\CMake\bin\cmake.exe" -E env ^
  SLICER_PACKAGE_MANAGER_CLIENT_EXECUTABLE=/path/to/slicer_package_manager_client ^
  SLICER_PACKAGE_MANAGER_URL=https://slicer-packages.kitware.com ^
  SLICER_PACKAGE_MANAGER_API_KEY=a0b012c0123d012abc01234a012345a0 ^
  ^
"c:\Program Files\CMake\bin\cmake.exe" -DSlicer_DIR:PATH=~/Slicer-SuperBuild-Release/
↪Slicer-build ^
 -DSlicer_EXTENSION_DESCRIPTION_DIR:PATH=C:/D/ExtensionsIndex ^
 -DCMAKE_BUILD_TYPE:STRING=Release ^
 -DCTEST_MODEL:STRING=Experimental ^
 -DSlicer_UPLOAD_EXTENSIONS:BOOL=ON ^
 C:/D/S/Extensions/CMake

make
```

Build complete Extensions Index with dashboard submission

Continuous and nightly extension dashboards are setup on the Slicer factory machine maintained by Kitware. Developers can set up similar infrastructure privately for their custom applications.

By customizing the extension template dashboard script, it is possible to easily setup dashboard client submitting to CDash. See example dashboard scripts that are used on official Slicer build machines here. Note that these scripts are more complex than the template to allow code reuse between different configurations, but they are tested regularly and so guaranteed to work.

12.6.13 Frequently asked questions

Can an extension contain different types of modules?

Yes. Extensions are used to package together all types of Slicer modules.

Should the name of the source repository match the name of the extension?

Assuming your extension is named `AwesomeTool`, generally, we suggest to name the extension repository `SlicerAwesomeTool`. Doing so will minimize confusion by clearly stating that the code base is associated with Slicer.

We suggest to use the `Slicer` prefix in the extension name, too, when the extension is a Slicer interface to some third-party library (such as SlicerOpenIGTLink, SlicerElastix, SlicerANTs SlicerOpenCV).

Where can I find the extension templates?

The module and extension templates are available in the Slicer source tree: https://github.com/Slicer/Slicer/tree/main/Utilities/Templates/

Using the *Extension Wizard* module, developers can easily create a new extension without having to copy, rename and update manually every files.

How are Superbuild extension packaged?

Extensions using the Superbuild mechanism build projects in two steps:

- First, the project dependencies are built in an outer-build directory.
- Then, the project itself is built in an inner-build directory.

Extensions can use the Superbuild (i.e., CMake's ExternalProject_Add) mechanism. However, developers have to be careful that the packaging macros clean the project before reconfiguring it. This means that if one uses the Slicer extension packaging macros inside the inner-build directory, when packaging and uploading the extension package, the project will be reconfigured, and variables passed from the outer-build directory will be lost. If the project only depends on libraries that Slicer builds, this is not an issue. If the project has specific dependencies that Slicer does not compile on its own, the developer should be careful to instantiate the Slicer extension packaging macros only in the outer-build directory. This only means that in the latter case, tests should be instantiated in the outer-build directory to allow the Slicer extension building process to test the extension before uploading the extension and the tests results.

How to build a custom Slicer package with additional extensions bundled?

To build custom Slicer versions, it is recommended to use the Slicer Custom Application Template.

Can an extension depend on other extensions?

Yes. The dependency should be specified as a list by setting the variable `EXTENSION_DEPENDS` in the extension `CMakeLists.txt` and in the `dependency` field in the extension description. If the user installs `Extension2` that depends on `Extension1` then the extension manager will install `Extension1` automatically.

If you have `ModuleA`, `ModuleB` and `ModuleC` and `ModuleA` can be used as standalone one. You could create the following extensions:

- `Extension1` containing `ModuleA`.
- `Extension2` containing `ModuleB` and `ModuleC`, depending on `Extension1`.

Add the following variable to `Extension2/CMakeLists.txt`:

```
set(EXTENSION_DEPENDS Extension1)
```

How dependent extensions are configured and built?

If an ExtensionB depends on an ExtensionA, ExtensionA should be listed as dependency in the metadata of ExtensionB.

This can be done setting EXTENSION_DEPENDS in the CMakeLists.txt or by specifying depends field in the [[Documentation/{{documentation/version}}/Developers/Extensions/DescriptionFile|description file]].

Doing so will ensure that:

*(1) the extension build system configure the extensions in the right order *(2) ExtensionB is configured with option ExtensionA_DIR.

What are the extension specific targets: INSTALL, PACKAGE, packageupload, ...?

Slicer extension build system provides the developer with a set of convenient targets allowing to build and upload extensions.

- `RUN_TESTS`: Locally execute all tests.
- `PACKAGE` or `package`: Locally package the extension.
- `packageupload`: Locally package and upload the extension and upload to the extensions server. Requires access privileges to the server.
- `Experimental`: Configure, build, test the extension and publish result on CDash.
- `Continuous / Nightly`: retrieves the latest / latest nightly revision and runs the same steps as for the `Experimental` target.

Is –launch flag available for a MacOSX installed Slicer.app?

On MacOSx, running Slicer with the `--help` argument does NOT list the usual launcher related options.

```
$ ./Slicer.app/Contents/MacOS/Slicer --help
Usage
 Slicer [options]

 Options
   --, --ignore-rest                     Ignores the rest of the labeled arguments␣
↪following this flag. (default: false)
   -h, --help                            Display available command line arguments.
   [...]
   --version                             Displays version information and exits.
```

To provide some background information, when generating the package that will be distributed, an application bundle `Slicer.app` is created. As explained here, a bundle is a directory with a standardized hierarchical structure that holds executable code and the resources used by that code. It means that since all libraries contained within a bundle are referenced relatively to the location of either the CLI or the Slicer executable, the use of launcher does NOT make sens.

To help fixing-up the libraries, executables and plugins so that they reference each other in a relative way, CMake provides us with the BundleUtilities module.

This module is used in two situations:

- Fixup of Slicer application itself. See SlicerCPack.cmake#L36-68 and SlicerCPackBundleFixup.cmake.in.
- Fixup of an extension package. See SlicerExtensionCPack.cmake#L126-143 and SlicerExtensionCPackBundleFixup.cmake.in.

How to check if an extension is built by Slicer Extensions build system?

Sometimes it is desirable to build the same source code in two different modes: as a standalone package or as a Slicer extension. To differentiate the two cases, the developer could check for the value of `<ExtensionName>_BUILD_SLICER_EXTENSION` CMake variable. This variable will be set to ON when the extension is built by the Slicer Extensions build system and it is not set otherwise. See details here

How often extensions are uploaded on the extensions server?

Slicer extensions are built and uploaded to the extensions server every day.

- Packages for Slicer stable release are rebuilt and uploaded during the day (Eastern time). Results are available at https://slicer.cdash.org/index.php?project=SlicerStable

- Packages for the latest Slicer Preview Release is built every night (Eastern time). Results are available at https://slicer.cdash.org/index.php?project=SlicerPreview

Note that packages are not updated for previous Slicer Preview Releases. To get latest extensions for a Slicer Preview Release, install the latest Slicer Preview Release.

Will an extension be uploaded if associated tests are failing?

Independently of the extension test results, if the extension could be successfully packaged, it will be uploaded on the extensions server.

How do I associate a remote with my local extension git source directory?

- Start a terminal (or Git Bash on Windows)
- Get the associated SSH remote url. Need help?
- Associate the remote URL with your local git source tree

```
git remote add origin https://github.com/<username>/MyExtension
```

Which remote name is expected for extension git checkout?

When packaging an extension and generating the associated extension description file, the system will look for a remote named `origin`.

In case you get the error reported below, you will have to either rename or add a remote. Need help?

```
CMake Warning at /path/to/Slicer/CMake/FindGit.cmake:144 (message):
No remote origin set for git repository: /path/to/MyExtension
Call Stack (most recent call first):
/path/to/Slicer/CMake/SlicerMacroExtractRepositoryInfo.cmake:99 (GIT_WC_INFO)
/path/to/Slicer/CMake/SlicerExtensionCPack.cmake:55 (SlicerMacroExtractRepositoryInfo)
CMakeLists.txt:25 (include)
```

Why ExtensionWizard failed to describe extension: "script does not set 'EXTENSION_HOMEPAGE'"?

The issue is that the your extension has a "non standard" layout and the wizard has no way of extracting the expected information.

Similar issue has been discussed and reported for the SPHARM-PDM or UKF extension.

First half of the solution would be to move the metadata from `Common.cmake` to `CMakeLists.txt` as it is done in here. Then, you could make sure there is a project() statement by following what is suggested here.

If you prefer not to re-organize your extension, you could still contribute extension description file by creating it manually.

Is project() statement allowed in extension CMakeLists.txt?

Following Slicer r22038, the project statement is required.

Is call to "if(NOT Slicer_SOURCE_DIR)" required to protect "find_package(Slicer)" in extension CMakeLists.txt?

Following Slicer r22063, protecting call to `find_package(Slicer)` with `if(NOT Slicer_SOURCE_DIR)` is no longer needed and should be removed to keep code simpler and easier to maintain.

Before:

```
cmake_minimum_required(VERSION 2.8.9)

if(NOT Slicer_SOURCE_DIR)
  find_package(Slicer COMPONENTS ConfigurePrerequisites)
endif()

if(NOT Slicer_SOURCE_DIR)
  set(EXTENSION_NAME EmptyExtensionTemplate)
  set(EXTENSION_HOMEPAGE "https://www.slicer.org/wiki/Documentation/Nightly/Extensions/
↪EmptyExtensionTemplate")
  set(EXTENSION_CONTRIBUTORS "Jean-Christophe Fillion-Robin (Kitware)")
  set(EXTENSION_DESCRIPTION "This is an example of extension bundling N module(s)")
  set(EXTENSION_ICONURL "http://viewvc.slicer.org/viewvc.cgi/Slicer4/trunk/Extensions/
↪Testing/EmptyExtensionTemplate/EmptyExtensionTemplate.png?revision=21746&view=co")
  set(EXTENSION_SCREENSHOTURLS "https://www.slicer.org/w/img_auth.php/4/42/Slicer-r19441-
↪EmptyExtensionTemplate-screenshot.png")
endif()

if(NOT Slicer_SOURCE_DIR)
  find_package(Slicer REQUIRED)
  include(${Slicer_USE_FILE})
endif()

add_subdirectory(ModuleA)

if(NOT Slicer_SOURCE_DIR)
  include(${Slicer_EXTENSION_CPACK})
endif()
```

After:

```
cmake_minimum_required(VERSION 2.8.9)

find_package(Slicer COMPONENTS ConfigurePrerequisites)

project(EmptyExtensionTemplate)

set(EXTENSION_HOMEPAGE "https://www.slicer.org/wiki/Documentation/Nightly/Extensions/
↪EmptyExtensionTemplate")
set(EXTENSION_CONTRIBUTORS "Jean-Christophe Fillion-Robin (Kitware)")
set(EXTENSION_DESCRIPTION "This is an example of empty extension")
set(EXTENSION_ICONURL "http://viewvc.slicer.org/viewvc.cgi/Slicer4/trunk/Extensions/
↪Testing/EmptyExtensionTemplate/EmptyExtensionTemplate.png?revision=21746&view=co")
set(EXTENSION_SCREENSHOTURLS "https://www.slicer.org/w/img_auth.php/4/42/Slicer-r19441-
↪EmptyExtensionTemplate-screenshot.png")

find_package(Slicer REQUIRED)
include(${Slicer_USE_FILE})

add_subdirectory(ModuleA)

include(${Slicer_EXTENSION_CPACK})
```

Why is the –contribute option is not available with the ExtensionWizard?

Wizard contribute option is available only (1) if Slicer is built with OpenSSL support or (2) directly from the nightly.

To build Slicer with SSL support, you need to build (or download) Qt with SSL support and configure Slicer with `-DSlicer_USE_PYTHONQT_WITH_OPENSSL:BOOL=ON`

How to package third party libraries?

Extensions integrating third party libraries should follow the SuperBuild extension template.

Each third party libraries will be configured and built using a dedicated `External_MyLib.cmake` file, the install location of binaries and libraries should be set to `Slicer_INSTALL_BIN_DIR` and `Slicer_INSTALL_LIB_DIR`.

Also, starting with Slicer v4.8.0 (introduced in this commit), extension can package python modules and packages using `PYTHON_SITE_PACKAGES_SUBDIR` CMake variable to specify the install destination.

These relative paths are the one that the extensions manager will consider when generating the launcher and application settings for a given extension.

Can I use C++14/17/20 language features?

We try to balance between compatibility and using new features. As a result, currently C++14 features are allowed, but usage of C++17/20 language features are discouraged in extensions (relying on C++17/20 features may lead to build errors on some build configurations).

If your extension can be compiled as a standalone project where you would like to use newer feature, you could rely on CMake detecting compile features. See cmake-compile-features for more details.

See the labs topic on upgrading compiler infrastructure for additional information/status.

How do I publish a paper about my extension?

Consider publishing a paper describing your extension. This page contains a list of journals that publish papers about software. *Citing 3D Slicer* in all papers that use 3D Slicer is greatly appreciated.

How to force Slicer to download extensions corresponding to a different Slicer revision?

Since extensions available from the Extensions Manager are associated with a particular Slicer revision, for development versions, typically no extension builds will appear in the Extensions Manager. For testing purposes, the current revision can be overridden in the Python console in Slicer:

```
>>> extensionManagerModel = slicer.app.extensionsManagerModel()
>>> extensionManagerModel.slicerRevision = "25742"
```

On other approach is to re-configure and build Slicer setting the `Slicer_FORCED_WC_REVISION` option.

How to address ITK test driver caught an ITK exception "Could not create IO object for reading file"?

If the following exception is reported when trying to run tests associated with a CLI modules:

```
ITK test driver caught an ITK exception:

itk::ImageFileReaderException (0x1bd8430)
Location: "unknown"
File: /path/to/Slicer-SuperBuild/ITK/Modules/IO/ImageBase/include/itkImageFileReader.hxx
Line: 139
Description:  Could not create IO object for reading file /path/to/image.nrrd
  Tried to create one of the following:
```

(continues on next page)

(continued from previous page)

```
    MRMLIDImageIO
  You probably failed to set a file suffix, or
    set the suffix to an unsupported type.
```

It most likely means that the test driver is not linking against `ITKFactoryRegistration` library and/or registering the ITK factories. To address this, the test driver should be updated:

- link against `${SlicerExecutionModel_EXTRA_EXECUTABLE_TARGET_LIBRARIES}`
- include `itkFactoryRegistration.h`
- call `itk::itkFactoryRegistration();` in its main function.

For more details, read What is the ITKFactoryRegistration library?.

12.7 Python FAQ

Frequently asked questions about how to write Python scripts for Slicer.

12.7.1 What is the Slicer Python environment?

You can consider each Slicer installation as a virtual Python environment - the same way as you create virtual environments using python or conda.

12.7.2 What is the PythonSlicer executable?

`PythonSlicer` is an executable provided in the `bin` directory of the Slicer installation. It is a Python interpreter that allows access to all of the packages installed in Slicer. This makes it possible to use the Slicer Python environment from outside the application, such as for batch processing, command-line operations, terminal-based interactive session and integration with IDEs.

This means you can use `PythonSlicer` as a replacement for the regular Python interpreter (`python` or `python3`) to take advantage of the installed packages in the Slicer environment.

Warning

Please note that the running Python context is **not** available when using `PythonSlicer`. As a result, many objects, including the application instance, the MRML scene and the loaded modules, are not available.

Tip

To install additional packages, you can use the *`slicer.packaging.pip_install()`* function.

12.7.3 What is the Python Console?

This describes the *Python console* available through the *user interface*.

It allows to access the running Python context including the application instance, the MRML scene and all the loaded modules through these :mod:`slicer` attributes:

- *`slicer.app`*
- *`slicer.mrmlScene`*

- *slicer.modules*
- *slicer.moduleNames*

Hint

Running scripts (or code) using the command-line option `--python-script` (or `python-code`) is equivalent to running code in the Python Console.

Combined with the use of `--no-main-window`, this is useful for implementing batch processing pipelines leveraging capabilities only available in the context of a running Slicer application. For example, this applies to the *Segment Editor effects*.

Tip

To install additional packages, you can use the *slicer.packaging.pip_install()* function.

Changed in version 5.2.0: The `Python interactor` was renamed to `Python Console`. See related discussion.

12.7.4 How to access Slicer's Python API via an external program while Slicer is running?

There are several ways to access Slicer's Python API from an external program while Slicer is running:

- *SlicerWeb*: Exposes Slicer's Python environment as a web service that can respond to http(s) requests with data from the current application state or modify the application state. This is well-suited for applications that already use web requests.
- SlicerOpenIGTLink: A lightweight socket-based protocol for real-time data transfer. This is useful for applications that need to send many requests per second (e.g., continuous data streaming) or for clients that only have access to sockets and prefer to avoid the complexity of protocols like HTTP. In most cases, it performs well for sending requests at a rate of 10 to 100 requests per second.

  A native Python implementation, pyigtl, is available for use outside the Slicer application. It can be used to both stream data from Slicer and stream data to Slicer.
- SlicerJupyter: A protocol for interacting with Slicer using standard Jupyter kernel protocol (simple protocol built on top of ZeroMQ middleware). This is useful for applications that want to offer embedded Python console to Slicer and don't want to implement a Slicer-specific protocol.
- *Python debuggers*: Python debuggers like PyCharm, Visual Studio Code, and Eclipse can be used to visualize and debug Python scripts in Slicer, including setting breakpoints, executing code step-by-step, and viewing variables and the stack.

These approaches offer different trade-offs in terms of complexity, performance, and ease of use, so it's important to choose the one that best fits your needs. For more information, see the linked documentation for each approach.

12.7.5 How to run a CLI module from Python

Here's an example to create a model from a volume using the "Grayscale Model Maker" module:

```
def createModelFromVolume(inputVolumeNode):
  """Create surface mesh from volume node using CLI module"""
  # Set parameters
  parameters = {}
```

(continues on next page)

(continued from previous page)

```
  parameters["InputVolume"] = inputVolumeNode
  outputModelNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLModelNode")
  parameters["OutputGeometry"] = outputModelNode
  # Execute
  grayMaker = slicer.modules.grayscalemodelmaker
  cliNode = slicer.cli.runSync(grayMaker, None, parameters)
  # Process results
  if cliNode.GetStatus() & cliNode.ErrorsMask:
    # error
    errorText = cliNode.GetErrorText()
    slicer.mrmlScene.RemoveNode(cliNode)
    raise ValueError("CLI execution failed: " + errorText)
  # success
  slicer.mrmlScene.RemoveNode(cliNode)
  return outputModelNode
```

To try this, download the MRHead dataset using "Sample Data" module and paste the code above into the Python console and then run this:

```
volumeNode = getNode('MRHead')
modelNode = createModelFromVolume(volumeNode)
```

A complete example for running a CLI module from a scripted module is available here

Get list of parameter names

The following script prints all the parameter names of a CLI parameter node:

```
cliModule = slicer.modules.grayscalemodelmaker
n=cliModule.cliModuleLogic().CreateNode()
for groupIndex in range(n.GetNumberOfParameterGroups()):
  print(f'Group: {n.GetParameterGroupLabel(groupIndex)}')
  for parameterIndex in range(n.GetNumberOfParametersInGroup(groupIndex)):
    print('  {0} [{1}]: {2}'.format(n.GetParameterName(groupIndex, parameterIndex),
      n.GetParameterTag(groupIndex, parameterIndex),n.GetParameterLabel(groupIndex,␣
↪parameterIndex)))
```

Passing markups point list to CLIs

```
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
head = sampleDataLogic.downloadMRHead()
volumesLogic = slicer.modules.volumes.logic()
headLabel = volumesLogic.CreateLabelVolume(slicer.mrmlScene, head, 'head-label')

pointListNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsFiducialNode")
pointListNode.AddControlPoint(vtk.vtkVector3d(1,0,5))
pointListNode.SetName('Seed Point')

params = {'inputVolume': head.GetID(), 'outputVolume': headLabel.GetID(), 'seed' :␣
↪pointListNode.GetID(), 'iterations' : 2}
```

(continues on next page)

(continued from previous page)

```
cliNode = slicer.cli.runSync(slicer.modules.simpleregiongrowingsegmentation, None,␣
↪params)
```

Running CLI in the background

If the CLI module is executed using `slicer.cli.run` method then the CLI module runs in a background thread, so the call to `startProcessing` will return right away and the user interface will not be blocked. The `slicer.cli.run` call returns a cliNode (an instance of vtkMRMLCommandLineModuleNode) which can be used to monitor the progress of the module.

In this example we create a simple callback `onProcessingStatusUpdate` that will be called whenever the cliNode is modified. The status will tell you if the nodes is Pending, Running, or Completed.

```
def startProcessing(inputVolumeNode):
  """Create surface mesh from volume node using CLI module"""
  # Set parameters
  parameters = {}
  parameters["InputVolume"] = inputVolumeNode
  outputModelNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLModelNode")
  parameters["OutputGeometry"] = outputModelNode
  # Start execution in the background
  grayMaker = slicer.modules.grayscalemodelmaker
  cliNode = slicer.cli.run(grayMaker, None, parameters)
  return cliNode

def onProcessingStatusUpdate(cliNode, event):
  print("Got a %s from a %s" % (event, cliNode.GetClassName()))
  if cliNode.IsA('vtkMRMLCommandLineModuleNode'):
    print("Status is %s" % cliNode.GetStatusString())
  if cliNode.GetStatus() & cliNode.Completed:
    if cliNode.GetStatus() & cliNode.ErrorsMask:
      # error
      errorText = cliNode.GetErrorText()
      print("CLI execution failed: " + errorText)
    else:
      # success
      print("CLI execution succeeded. Output model node ID: "+cliNode.
↪GetParameterAsString("OutputGeometry"))

volumeNode = getNode('MRHead')
cliNode = startProcessing(volumeNode)
cliNode.AddObserver('ModifiedEvent', onProcessingStatusUpdate)

# If you need to cancel the CLI, call
# cliNode.Cancel()
```

12.7.6 How to use a loadable module from Python

Both Python scripted loadable modules and C++ loadable modules can be used from Python in multiple ways: by modifying MRML nodes, calling methods of the module logic object, and adding reusable GUI widgets provided by the module.

It may be tempting to use another module via the module's GUI (for example, by simulating button clicks), since Qt allows access to widgets via public methods, signals, and properties. However, a module GUI is not designed to be manipulated programmatically, and doing so could lead to unexpected behavior. Therefore, it is only recommended to interact with a module GUI object for testing and debugging.

Calling methods of a module logic

Functions that a module offers to other modules are made available via the module's logic class.

For most modules, the logic class can be accessed by calling the `slicer.util.getModuleLogic('ModuleName')` convenience function.

- If the type of the returned logic object is a `qSlicerBaseQTCLI.vtkSlicerCLIModuleLogic` then it means that it is a CLI module. Follow the instructions described in the *How to run a CLI module from Python* section.
- If the method returns an error, it means that the module does not instantiate a publicly accessible module logic object. Some Python scripted loadable modules create a logic just in time when it is needed. In this cases, a module logic can be instantiated by importing the corresponding Python module and instantiating classes - see for example how it is done for *DICOM module*.

There are many examples for using a module's logic class in the *Script Repository*. Note that in the examples a module logic may obtained by using `slicer.modules.modulename.logic()`, which only works for C++ loadable modules, but otherwise it returns the same result as `slicer.util.getModuleLogic()`.

Module logic methods can be explored by using the `help` Python function (`help(slicer.util.getModuleLogic('Markups'))`) or using auto-complete in the Python console (type `slicer.util.getModuleLogic('Markups').Get` and hit Tab). Documentation of module logic of C++ classes can be found in the *C++ API documentation*, for Python scripted modules developers currently need to get documentation from the source code.

Adding reusable widgets provided by modules

Most Slicer core modules provide a set of reusable, configurable GUI widgets that can help developers in building their module's user interface.

For example, Volumes module offers a widget that can display and edit basic properties of volume nodes:

```
volumeNode = getNode('MRHead')
w = slicer.qMRMLVolumeInfoWidget()
w.setMRMLScene(slicer.mrmlScene)
w.setVolumeNode(volumeNode)
w.show()
```

While these widgets can be instantiated using Python scripting, it is more convenient (and requires less code) to add them to a module's user interface using Qt designer.

12.7.7 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::SliceResolutionMatc` in Python syntax it is `someSliceNode.SetSliceResolutionMode(slicer.vtkMRMLSliceNode.SliceResolutionMatchVolumes)`. For example, for the red slice node this will be:

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

12.7.8 How to connect GUI widget events to Python code

Slicer uses Qt toolkit for providing graphical user interface (GUI). Qt is made available in Python via PythonQt. Using this Python wrapper, a GUI widget event - it is called a `signal` in Qt - can be connected to a Python function using the syntax: *someQtWidgetObject.signalName*.connect(*slotName*).

For example, a QProgressBar will have these signals such as valueChanged. Therefore a Python function can be connected like this:

```
def printMyNewValue(value):
  print("The progress bar value is now: {}".format(value))

import qt
progress_bar = qt.QProgressBar()
progress_bar.setMaximum(10)
progress_bar.valueChanged.connect(printMyNewValue)
progress_bar.setValue(4)  # will then print "The progress bar value is now: 4"
```

Where to get a list of signals for a widget object? You can find the specification of signals in the header files or the generated documentation. Typically, the first hit on Google search for the class name brings up the documentation page or header file of the class.

Where to get examples? Since Slicer is open-source and there are about 200 extensions to it, mostly developed in Python, hosted on github, there is a very high chance that there are already examples for using the signal you need. You can search in all Python code in entire GitHub for a usage example by typing the name of the class and signal. For example you can search for `ctkPathLineEdit currentPathChanged` in Python code.

12.7.9 How to run an external Python script as a CLI module

A standalone Python script (that does not use any Slicer application features) can run from Slicer as a CLI module. Slicer generates a graphical user interface from the parameter definition XML file. See a complete example here.

12.7.10 How to type file paths in Python

New Python users on Windows often surprised when they enter a path that contain backslash character (\) and it just does not work. Since backslash (\) is an escape character in Python, it requires special attention when used in string literals. For example, this is incorrect:

```
somePath = "F:\someFolder\myfile.nrrd"  # incorrect (\s and \m are interpreted as
↪special characters)
```

The easiest method for using a path that contains backslash character is to declare the text as "raw string" by prepending an `r` character. This is correct:

```
somePath = r"F:\someFolder\myfile.nrrd"
```

It is possible to keep the text as regular string and typing double-backslash instead of . This is correct:

```
somePath = "F:\\someFolder\\myfile.nrrd"
```

In most places, unix-type separators can be used instead of backslash. This is correct:

```
somePath = "F:/someFolder/myfile.nrrd"
```

See more information in Python documentation: https://docs.python.org/3/tutorial/introduction.html?#strings

12.7.11 How to modify a Python scripted module

If `Developer mode` is enabled in the application settings then the `Reload and Test` section is displayed at the top of the user interface of Python scripted modules. This section contains buttons for convenient editing of the module source code (`.py` file) and user interface (`.ui` file). By default, clicking on the `Edit` button opens the module source code in the program associated with `.py` files, as defined in the operating system settings. This behavior can be overridden by specifying a text editor (such as VS Code, Sublime Text, . . .) in the application settings: `Editor for .py files` in the `Python` section.

Tip

On Windows, VS Code text editor is installed by default at:

```
C:/Users/YourUserName/AppData/Local/Programs/Microsoft VS Code/Code.exe
```

12.7.12 How to include Python modules in an extension

Sometimes a Python scripted module grows big and it becomes inconvenient to have all the source code in a single .py file. Since all the .py files in a folder that is listed among "additional module paths" are expected to be Slicer modules, these additional files cannot be simply placed in the same folder as in the Slicer module. Instead, all additional .py files can be put in a subfolder, as a regular Python module.

For example, the files associated with a Slicer module could look like this:

```
.
├── CMakeLists.txt
├── MySlicerModuleLib
│   ├── __init__.py
│   ├── cool_maths.py
│   └── utils.py
└── MySlicerModule.py
```

So that the following code can run within MySlicerModule.py:

```
from MySlicerModuleLib import utils, cool_maths
```

By default, only the Slicer module (MySlicerModule.py) will be included in the package distributed via the *Extensions Manager* (see a related issue on GitHub). To make sure all the necessary files are included in the package, the CMakeLists.txt file associated with the Slicer module needs to be modified. Initially, the second section of CMakeLists.txt will look like this:

```
set(MODULE_PYTHON_SCRIPTS
  ${MODULE_NAME}.py
  )
```

All the necessary files need to be added to the list. In our example:

```
set(MODULE_PYTHON_SCRIPTS
  ${MODULE_NAME}.py
  ${MODULE_NAME}Lib/__init__.py
  ${MODULE_NAME}Lib/cool_maths.py
  ${MODULE_NAME}Lib/utils.py
  )
```

12.7.13 Can I use any Python package in a Slicer module

You can install any Python package within Slicer's built-in Python environment.

Utility functions for dependency management

Added in version 5.11: The slicer.packaging module with pip_ensure, pip_check, load_requirements, and load_pyproject_dependencies.

Slicer provides several utility functions to help manage Python dependencies:

The recommended approach is to use *slicer.packaging.pip_ensure()*, which handles checking, user confirmation, and installation with progress display in one call. It also automatically skips installation when running in testing mode (slicer.app.testingEnabled()).

For direct installation with visual feedback, use *slicer.packaging.pip_install()* with show_progress=True. This displays a modal progress dialog during installation.

For detailed usage examples, see *Install a Python package* in the Script Repository.

Warning

Since installing packages can have side effects on other extensions or the main application, here are some best practices to adhere to:

DO:

- Ask the user before installing. *slicer.packaging.pip_ensure()* does this for you; if you go through the lower-level *slicer.packaging.pip_install()*, write your own confirmation dialog as shown in the linked example.
- Document the dependencies your module relies upon using a requirements.txt file.
- Consider specifying version requirements using >=X.Y to avoid incompatible versions.

- Verify that all Python packages are distributed as Python wheels. This is particularly important for dependencies including compiled code, as installing a wheel eliminates the need for users to install a compiler.

DON'T:

- Do not install any packages in the global scope (outside of all classes and functions) or in the module class constructor. This can significantly slow down application startup, and it may even prevent the module from loading.
- Do not pin to a specific version of the package, as this may generate conflicts with other package versions, leading to unexpected environment modifications. Pinning dependencies should be considered only in the context of custom applications where the deployment environment is tightly controlled.

12.8 Script repository

Note

Usage: Copy-paste the code lines displayed below or the linked `.py` file contents into Python console in Slicer. Or save them to a `.py` file and run them using `execfile`.

To run a Python code snippet automatically at each application startup, add it to the *.slicerrc.py file*.

Note

More reference code: The Slicer source code has Python scripted modules and scripted Segmentation Editor effects that can be used as working examples.

- Most Slicer Extensions are written in Python to address specific use cases. Looking at their source code can be informative.
- The Slicer Discourse forum has many code snippets and discussions.

12.8.1 Install Slicer

There are different approaches to install Slicer and extensions programmatically:

- Install Slicer manually and install extensions by using `slicer.app.extensionsManagerModel()`. See example *below* and in install-slicer-extension.py
- Directly interact with the REST API endpoints of https://slicer-packages.kitware.com using `curl` and `jq`. See slicer-download.sh

12.8.2 Launch Slicer

Open a file with Slicer at the command line

Open Slicer to view the `c:\some\folder\MRHead.nrrd` image file:

```
"c:\Users\myusername\AppData\Local\NA-MIC\Slicer 4.11.20210226\Slicer.exe" c:\some\
↪folder\MRHead.nrrd
```

Note

It is necessary to specify full path to the Slicer executable and to the file that needs to be loaded.

On Windows, the installer registers the `Slicer` application during install. This makes it possible to use `start` command to launch the most recently installed Slicer. For example, this command on the command-line starts Slicer and loads an image:

```
start Slicer c:\some\folder\MRHead.nrrd
```

To load a file with non-default options, you can use `--python-code` option to run `slicer.util.load...` commands.

Open an .mrb file with Slicer at the command line

```
Slicer.exe --python-code "slicer.util.loadScene('f:/2013-08-23-Scene.mrb')"
```

Run Python commands in the Slicer environment

Run Python commands, without showing any graphical user interface:

```
Slicer.exe --python-code "doSomething; doSomethingElse; etc." --testing --no-splash --no-
↪main-window
```

Slicer exits when the commands are completed because `--testing` options is specified.

Run a Python script file in the Slicer environment

Run a Python script on Windows (stored in script file), without showing any graphical user interface:

```
Slicer.exe --python-script "/full/path/to/myscript.py" --no-splash --no-main-window
```

Run a Python script on MacOS (stored in script file), without showing any graphical user interface:

```
/Applications/Slicer.app/Contents/MacOS/Slicer --no-splash --no-main-window --python-
↪script "/full/path/to/myscript.py"
```

To make Slicer exit when the script execution is completed, call `sys.exit(errorCode)` (where `errorCode` is set 0 for success and other value to indicate error).

Launch Slicer directly from a web browser

Slicer can be associated with the `slicer:` custom URL protocol in the operating system or web browser. This is done automatically in the Windows installer and can be set up on other operating systems manually. After this when the user clicks on a `slicer://...` URL in the web browser, Slicer will start and the `slicer.app` object emits a signal with the URL that modules can process. The DICOM module processes DICOMweb URLs, but any module can specify additional actions.

For example, this module will download and view any image if the user clicks on an URL like this in the web browser:

```
slicer://viewer/?download=https%3A%2F%2Fgithub.com%2Frbumm%2FSlicerLungCTAnalyzer
↪%2Freleases%2Fdownload%2FSampleData%2FLungCTAnalyzerChestCT.nrrd
```

For reference, the DICOM module downloads a study from a DICOMweb server and shows it when providing a URL like this (which is used for example in the Kheops DICOM data sharing platform):

```
slicer://viewer/?studyUID=2.16.840.1.113669.632.20.121711.10000158860
    &access_token=k0zR6WAPpNbVguQ8gGUHp6
    &dicomweb_endpoint=http%3A%2F%2Fdemo.kheops.online%2Fapi
    &dicomweb_uri_endpoint=%20http%3A%2F%2Fdemo.kheops.online%2Fapi%2Fwado
```

12.8.3 MRML scene

Get MRML node from the scene

Get markups point list node named F (useful for quickly getting access to a MRML node in the Python console):

```
pointListNode = getNode('F')
# do something with the node... let's remove the first control point in it
pointListNode.RemoveNthControlPoint(0)
```

Getting the first volume node without knowing its name (useful if there is only one volume loaded):

```
volumeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")
# do something with the node... let's change its display window/level
volumeNode.GetDisplayNode().SetAutoWindowLevel(False)
volumeNode.GetDisplayNode().SetWindowLevelMinMax(100, 200)
```

Note

- *slicer.util.getNode()* is recommended **only for interactive debugging** in the Python console/Jupyter notebook
  - its input is intentionally defined vaguely (it can be either node ID or name and you can use wildcards such as *), which is good because it make it simpler to use, but the uncertain behavior is not good for general-purpose use in a module
  - throws an exception so that the developer knows immediately that there was a typo or other unexpected error
- `slicer.mrmlScene.GetNodeByID()` is more appropriate when a module needs to access a MRML node:
  - its behavior is more predictable: it only accepts node ID as input. `slicer.mrmlScene.GetFirstNodeByName()` can be used to get a node by its name, but since multiple nodes in the scene can have the same name, it is not recommended to keep reference to a node by its name. Since node IDs may change when a scene is saved and reloaded, node ID should not be stored persistently, but *node references* must be used instead
  - if node is not found it returns `None` (instead of throwing an exception), because this is often not considered an error in module code (it is just used to check existence of a node) and using return value for not-found nodes allows simpler syntax

Clone a node

This example shows how to make a copy of any node that appears in Subject Hierarchy (in Data module).

```
# Get a node from SampleData that we will clone
import SampleData
nodeToClone = SampleData.SampleDataLogic().downloadMRHead()
```

(continues on next page)

(continued from previous page)

```
# Clone the node
shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
itemIDToClone = shNode.GetItemByDataNode(nodeToClone)
clonedItemID = slicer.modules.subjecthierarchy.logic().CloneSubjectHierarchyItem(shNode,␣
↪itemIDToClone)
clonedNode = shNode.GetItemDataNode(clonedItemID)
```

Save a node to file

Save a transform node to file (should work with any other node type, if file extension is set to a supported one):

```
myNode = getNode("LinearTransform_3")

myStorageNode = myNode.CreateDefaultStorageNode()
myStorageNode.SetFileName("c:/tmp/something.tfm")
myStorageNode.WriteData(myNode)
```

Save the scene into a single MRB file

```
# Generate file name
import time
sceneSaveFilename = slicer.app.temporaryPath + "/saved-scene-" + time.strftime("%Y%m%d-%H
↪%M%S") + ".mrb"

# Save scene
if slicer.util.saveScene(sceneSaveFilename):
  logging.info("Scene saved to: {0}".format(sceneSaveFilename))
else:
  logging.error("Scene saving failed")
```

Save the scene into a new directory

```
# Create a new directory where the scene will be saved into
import time
sceneSaveDirectory = slicer.app.temporaryPath + "/saved-scene-" + time.strftime("%Y%m%d-
↪%H%M%S")
if not os.access(sceneSaveDirectory, os.F_OK):
  os.makedirs(sceneSaveDirectory)

# Save the scene
if slicer.app.applicationLogic().SaveSceneToSlicerDataBundleDirectory(sceneSaveDirectory,
↪ None):
  logging.info("Scene saved to: {0}".format(sceneSaveDirectory))
else:
  logging.error("Scene saving failed")
```

Override default scene save dialog

Place this class in the scripted module file to override

```
class MyModuleFileDialog ():
  """This specially named class is detected by the scripted loadable
  module and is the target for optional drag and drop operations.
  See: Base/QTGUI/qSlicerScriptedFileDialog.h.

  This class is used for overriding default scene save dialog
  with simple saving the scene without asking anything.
  """

  def __init__(self,qSlicerFileDialog ):
    self.qSlicerFileDialog = qSlicerFileDialog
    qSlicerFileDialog.fileType = "NoFile"
    qSlicerFileDialog.description = "Save scene"
    qSlicerFileDialog.action = slicer.qSlicerFileDialog.Write

  def execDialog(self):
    # Implement custom scene save operation here.
    # Return True if saving completed successfully,
    # return False if saving was cancelled.
    ...
    return saved
```

Override application close behavior

When application close is requested then by default confirmation popup is displayed. To customize this behavior (for example, allow application closing without displaying default confirmation popup) an event filter can be installed for the close event on the main window:

```
class CloseApplicationEventFilter(qt.QWidget):
  def eventFilter(self, object, event):
    if event.type() == qt.QEvent.Close:
      event.accept()
      return True
    return False

filter = CloseApplicationEventFilter()
slicer.util.mainWindow().installEventFilter(filter)
```

Change default output file type for new nodes

This script changes default output file format for nodes that have not been saved yet (do not have storage node yet).

Default node can be specified that will be used as a basis of all new storage nodes. This can be used for setting default file extension. For example, change file format to PLY for model nodes:

```
defaultModelStorageNode = slicer.vtkMRMLModelStorageNode()
defaultModelStorageNode.SetDefaultWriteFileExtension("ply")
slicer.mrmlScene.AddDefaultNode(defaultModelStorageNode)
```

To permanently change default file extension on your computer, copy-paste the code above into your application startup script (you can find its location in menu: Edit / Application settings / General / Application startup script).

Change file type for saving for existing nodes

This script changes output file types for nodes that have been already saved (they already have storage node).

If it is not necessary to preserve file paths then the simplest is to configure default storage node (as shown in the example above), then delete all existing storage nodes. When save dialog is opened, default storage nodes will be recreated.

```
# Delete existing model storage nodes so that they will be recreated with default␣
↪settings
existingModelStorageNodes = slicer.util.getNodesByClass("vtkMRMLModelStorageNode")
for modelStorageNode in existingModelStorageNodes:
  slicer.mrmlScene.RemoveNode(modelStorageNode)
```

To update existing storage nodes to use new file extension (but keep all other parameters unchanged) you can use this approach (example is for volume storage):

```
requiredFileExtension = ".nia"
originalFileExtension = ".nrrd"
volumeNodes = slicer.util.getNodesByClass("vtkMRMLScalarVolumeNode")
for volumeNode in volumeNodes:
  volumeStorageNode = volumeNode.GetStorageNode()
  if not volumeStorageNode:
    volumeNode.AddDefaultStorageNode()
    volumeStorageNode = volumeNode.GetStorageNode()
    volumeStorageNode.SetFileName(volumeNode.GetName()+requiredFileExtension)
  else:
    volumeStorageNode.SetFileName(volumeStorageNode.GetFileName().
↪replace(originalFileExtension, requiredFileExtension))
```

To set all volume nodes to save uncompressed by default (add this to *slicerrc.py file* so it takes effect for the whole session):

```
# Set the default volume storage to not compress by default
defaultVolumeStorageNode = slicer.vtkMRMLVolumeArchetypeStorageNode()
defaultVolumeStorageNode.SetUseCompression(0)
slicer.mrmlScene.AddDefaultNode(defaultVolumeStorageNode)
logging.info("Volume nodes will be stored uncompressed by default")
```

Same thing as above, but applied to all segmentations instead of volumes:

```
# Set the default segmentation storage to not compress by default
defaultSegmentationStorageNode = slicer.vtkMRMLSegmentationStorageNode()
defaultSegmentationStorageNode.SetUseCompression(0)
slicer.mrmlScene.AddDefaultNode(defaultSegmentationStorageNode)
logging.info("Segmentation nodes will be stored uncompressed by default")
```

12.8.4 Module selection

Switch to a different module

This utility function can be used to open a different module:

```
slicer.util.selectModule("DICOM")
```

Set a new default module at startup

Instead of the default Welcome module:

```
qt.QSettings().setValue("Modules/HomeModule", "Data")
```

12.8.5 Views

Display text in a 3D view or slice view

The easiest way to show information overlaid on a viewer is to use corner annotations.

```
view=slicer.app.layoutManager().threeDWidget(0).threeDView()
# Set text to "Something"
view.cornerAnnotation().SetText(vtk.vtkCornerAnnotation.UpperRight,"Something")
# Set color to red
view.cornerAnnotation().GetTextProperty().SetColor(1,0,0)
# Update the view
view.forceRender()
```

To display text in slice views, replace the first line by this line (and consider hiding slice view annotations, to prevent them from overwriting the text you place there):

```
view=slicer.app.layoutManager().sliceWidget("Red").sliceView()
```

Activate hanging protocol by keyboard shortcut

This code snippet shows how to specify a hanging protocol for PET/CT with the following properties:

- window/level and colormap is set to standardized values
- any acquisition transforms hardened on the images (these transforms are created for example when the image is acquired with varying slice spacing)
- show PET/CT images fused in slice views
- show PET image and fused image slices in 3D view

The hanging protocol can be activated using the Ctrl+9 keyboard shortcut.

```
def useHangingProtocolPetCt():
    ctImage = None
    petImage = None

    shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
    petColor = slicer.mrmlScene.GetFirstNodeByName('PET-Heat')
    for imageNode in slicer.util.getNodesByClass('vtkMRMLScalarVolumeNode'):
        # Harden any transform (in case the image is stored non-uniform spacing, etc.
        # hardening the acquisition transforms creates a single Cartesian volume)
        imageNode.HardenTransform()

        # Set window/level and colormap for recognized image types
        imageItem = shNode.GetItemByDataNode(imageNode)
        modality = shNode.GetItemAttribute(imageItem, 'DICOM.Modality')
        if modality == "CT":
            ctImage = imageNode
            ctImage.GetVolumeDisplayNode().SetAndObserveColorNodeID(petColor.GetID())
```

(continues on next page)

(continued from previous page)

```
            slicer.modules.volumes.logic().ApplyVolumeDisplayPreset(ctImage.
↪GetVolumeDisplayNode(), "CT_ABDOMEN")
        elif modality == "PT":
            petImage = imageNode
            petImage.GetVolumeDisplayNode().SetAndObserveColorNodeID(petColor.GetID())
            petImage.GetVolumeDisplayNode().SetWindowLevelMinMax(0, 20)

    # Set up view layout and content
    slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.SlicerLayoutFourUpView)
    slicer.util.setSliceViewerLayers(background=ctImage, foreground=petImage,␣
↪foregroundOpacity=0.3, fit=True)

    # Show the PET image in 3D view using volume rendering
    vrLogic = slicer.modules.volumerendering.logic()
    vrDisplayNode = vrLogic.CreateDefaultVolumeRenderingNodes(petImage)
    vrDisplayNode.SetVisibility(True)
    # Use the same window/level and colormap settings for volume rendering as for slice␣
↪display
    vrDisplayNode.SetFollowVolumeDisplayNode(True)

    # Show slice views in 3D view
    layoutManager = slicer.app.layoutManager()
    for sliceViewName in layoutManager.sliceViewNames():
        controller = layoutManager.sliceWidget(sliceViewName).sliceController()
        controller.setSliceVisible(True)

    # Center and fit displayed content in 3D view
    layoutManager = slicer.app.layoutManager()
    threeDWidget = layoutManager.threeDWidget(0)
    threeDView = threeDWidget.threeDView()
    threeDView.rotateToViewAxis(3)  # look from anterior direction
    threeDView.resetFocalPoint()  # reset the 3D view cube size and center it
    threeDView.resetCamera()  # reset camera zoom

    return [ctImage, petImage]

# Register keyboard shortcut
shortcut = qt.QShortcut(slicer.util.mainWindow())
shortcut.setKey(qt.QKeySequence("Ctrl+9"))
shortcut.connect( "activated()", useHangingProtocolPetCt)
```

Show orientation marker in all views

```
viewNodes = slicer.util.getNodesByClass("vtkMRMLAbstractViewNode")
for viewNode in viewNodes:
  viewNode.SetOrientationMarkerType(slicer.vtkMRMLAbstractViewNode.
↪OrientationMarkerTypeAxes)
```

Change view axis labels

```
labels = ["x", "X", "y", "Y", "z", "Z"]
viewNode = slicer.app.layoutManager().threeDWidget(0).mrmlViewNode()
# for slice view:
# viewNode = slicer.app.layoutManager().sliceWidget("Red").mrmlSliceNode()
for index, label in enumerate(labels):
  viewNode.SetAxisLabel(index, label)
```

Hide view controller bars

```
slicer.app.layoutManager().threeDWidget(0).threeDController().setVisible(False)
slicer.app.layoutManager().sliceWidget("Red").sliceController().setVisible(False)
slicer.app.layoutManager().plotWidget(0).plotController().setVisible(False)
slicer.app.layoutManager().tableWidget(0).tableController().setVisible(False)
```

Hide Slicer logo from main window

This script increases vertical space available in the module panel by hiding the Slicer application logo.

```
slicer.util.findChild(slicer.util.mainWindow(), "LogoLabel").visible = False
```

Customize widgets in view controller bars

```
sliceController = slicer.app.layoutManager().sliceWidget("Red").sliceController()

# hide what is not needed
sliceController.pinButton().hide()
#sliceController.viewLabel().hide()
sliceController.fitToWindowToolButton().hide()
sliceController.sliceOffsetSlider().hide()

# add custom widgets
myButton = qt.QPushButton("My custom button")
sliceController.barLayout().addWidget(myButton)
```

Display a node in only some views

Show a displayable node in a predefined set of views. Here, show a markups line only in `Red+`, `Green+` and `Yellow+` views.

```
displayableNode = getNode("L")

viewNodeIDs = [
  slicer.app.layoutManager().sliceWidget(viewName).mrmlSliceNode().GetID()
  for viewName in ["Red+", "Green+", "Yellow+"]
]

displayableNode.GetDisplayNode().SetViewNodeIDs(viewNodeIDs)
```

Get current mouse coordinates in a slice view

You can get 3D (RAS) coordinates of the current mouse cursor from the crosshair singleton node as shown in the example below:

```
def onMouseMoved(observer,eventid):
  ras=[0,0,0]
  crosshairNode.GetCursorPositionRAS(ras)
  print(ras)

crosshairNode=slicer.util.getNode("Crosshair")
crosshairNode.AddObserver(slicer.vtkMRMLCrosshairNode.CursorPositionModifiedEvent,␣
↪onMouseMoved)
```

Display crosshair at a 3D position

```
position_RAS = [23.4, 5.6, 78.9]
crosshairNode = slicer.util.getNode("Crosshair")
# Set crosshair position
crosshairNode.SetCrosshairRAS(position_RAS)
# Center the position in all slice views
slicer.vtkMRMLSliceNode.JumpAllSlices(slicer.mrmlScene, *position_RAS, slicer.
↪vtkMRMLSliceNode.CenteredJumpSlice)
# Make the crosshair visible
crosshairNode.SetCrosshairMode(slicer.vtkMRMLCrosshairNode.ShowBasic)
```

Note

Crosshair node stores two positions: Cursor position is the current position of the mouse pointer in a slice or 3D view (modules should only read this position). Crosshair position is the location of the visible crosshair in views (modules can read or write this position).

Change the crosshair color

```
# Get the crosshair node
crosshairNode = slicer.util.getNode("Crosshair")
# Set the crosshair color to Red
crosshairNode.SetCrosshairColor(1.0, 0.0, 0.0)
```

Display mouse pointer coordinates in alternative coordinate system

The Data probe only shows coordinate values in the world coordinate system. You can make the world coordinate system mean anything you want (e.g., MNI) by applying a transform to the volume that transforms it into that space. See more details in here .

```
def onMouseMoved(observer,eventid):
  mniToWorldTransformNode = getNode("LinearTransform_3")  # replace this by the name of␣
↪your actual MNI to world transform
  worldToMniTransform = vtk.vtkGeneralTransform()
  mniToWorldTransformNode.GetTransformToWorld(worldToMniTransform)
  ras=[0,0,0]
  mni=[0,0,0]
```

(continues on next page)

(continued from previous page)

```
  crosshairNode.GetCursorPositionRAS(ras)
  worldToMniTransform.TransformPoint(ras, mni)
  _ras = "; ".join([str(k) for k in ras])
  _mni = "; ".join([str(k) for k in mni])
  slicer.util.showStatusMessage(f"RAS={_ras}   MNI={_mni}")

crosshairNode=slicer.util.getNode("Crosshair")
observationId = crosshairNode.AddObserver(slicer.vtkMRMLCrosshairNode.
↪CursorPositionModifiedEvent, onMouseMoved)

# Run this to stop displaying values:
# crosshairNode.RemoveObserver(observationId)
```

Get 3D coordinates from 2D display coordinates

If 2D display position (in pixels) of a model's surface point is known then this code snippet can compute its position in 3D (in world coordinate system).

```
# Display position is in pixels, origin is top-left corner
displayPosition = [10, 12]

# Get model displayable manager
threeDViewNode = slicer.mrmlScene.GetNodeByID("vtkMRMLViewNode1")
appLogic = slicer.app.applicationLogic()
modelDisplayableManager = appLogic.GetViewDisplayableManagerByClassName(threeDViewNode,
↪"vtkMRMLModelDisplayableManager")

# Use model displayable manager's point picker
if modelDisplayableManager.Pick(displayPosition[0], displayPosition[1]) and␣
↪modelDisplayableManager.GetPickedNodeID():
    rasPosition = modelDisplayableManager.GetPickedRAS()
    print(rasPosition)
else:
    print(f"No model is visible at {displayPosition}")
```

Get DataProbe text

You can get the mouse location in pixel coordinates along with the pixel value at the mouse by hitting the `.` (period) key in a slice view after pasting in the following code.

```
def printDataProbe():
  infoWidget = slicer.modules.DataProbeInstance.infoWidget
  for layer in ("B", "F", "L"):
    print(infoWidget.layerNames[layer].text, infoWidget.layerIJKs[layer].text,␣
↪infoWidget.layerValues[layer].text)

s = qt.QShortcut(qt.QKeySequence("."), mainWindow())
s.connect("activated()", printDataProbe)
```

Create custom color table

This example shows how to create a new color table, for example with inverted color range from the default Ocean color table.

```
invertedocean = slicer.vtkMRMLColorTableNode()
invertedocean.SetTypeToUser()
invertedocean.SetNumberOfColors(256)
invertedocean.SetName("InvertedOcean")

for i in range(0,255):
  invertedocean.SetColor(i, 0.0, 1 - (i+1e-16)/255.0, 1.0, 1.0)

slicer.mrmlScene.AddNode(invertedocean)
```

Show color legend for a volume node

Display color legend for a volume node in slice views (and in 3D views, if the slice is displayed in 3D):

```
volumeNode = getNode('MRHead')
colorLegendDisplayNode = slicer.modules.colors.logic().
↪AddDefaultColorLegendDisplayNode(volumeNode)
```

Create custom color map and display color legend

```
modelNode = getNode('MyModel')  # color legend requires a displayable node
colorTableRangeMm = 40
title ="Radial\nCompression\n"
labelFormat = "%4.1f mm"

# Create color node
colorNode = slicer.mrmlScene.CreateNodeByClass("vtkMRMLProceduralColorNode")
colorNode.UnRegister(None)  # to prevent memory leaks
colorNode.SetName(slicer.mrmlScene.GenerateUniqueName("MyColormap"))
colorNode.SetAttribute("Category", "MyModule")
# The color node is a procedural color node, which is saved using a storage node.
# Hidden nodes are not saved if they use a storage node, therefore
# the color node must be visible.
colorNode.SetHideFromEditors(False)
slicer.mrmlScene.AddNode(colorNode)

# Specify colormap
colorMap = colorNode.GetColorTransferFunction()
colorMap.RemoveAllPoints()
colorMap.AddRGBPoint(colorTableRangeMm * 0.0, 0.0, 0.0, 1.0)
colorMap.AddRGBPoint(colorTableRangeMm * 0.2, 0.0, 1.0, 1.0)
colorMap.AddRGBPoint(colorTableRangeMm * 0.5, 1.0, 1.0, 0.0)
colorMap.AddRGBPoint(colorTableRangeMm * 1.0, 1.0, 0.0, 0.0)

# Display color legend
modelNode.GetDisplayNode().SetAndObserveColorNodeID(colorNode.GetID())
colorLegendDisplayNode = slicer.modules.colors.logic().
↪AddDefaultColorLegendDisplayNode(modelNode)
```

(continues on next page)

(continued from previous page)

```
colorLegendDisplayNode.SetTitleText(title)
colorLegendDisplayNode.SetLabelFormat(labelFormat)
```

Customize view layout

Show a custom layout of a 3D view on top of the red slice view:

```
customLayout = """
<layout type="vertical" split="true">
  <item>
  <view class="vtkMRMLViewNode" singletontag="1">
    <property name="viewlabel" action="default">1</property>
  </view>
  </item>
  <item>
  <view class="vtkMRMLSliceNode" singletontag="Red">
    <property name="orientation" action="default">Axial</property>
    <property name="viewlabel" action="default">R</property>
    <property name="viewcolor" action="default">#F34A33</property>
  </view>
  </item>
</layout>
"""

# Built-in layout IDs are all below 100, so you can choose any large random number
# for your custom layout ID.
customLayoutId=501

layoutManager = slicer.app.layoutManager()
layoutManager.layoutLogic().GetLayoutNode().AddLayoutDescription(customLayoutId,␣
↪customLayout)

# Switch to the new custom layout
layoutManager.setLayout(customLayoutId)
```

See description of standard layouts (that can be used as examples) here: https://github.com/Slicer/Slicer/blob/main/Libs/MRML/Logic/vtkMRMLLayoutLogic.cxx

You can use this code snippet to add a button to the layout selector toolbar:

```
# Add button to layout selector toolbar for this custom layout
viewToolBar = mainWindow().findChild("QToolBar", "ViewToolBar")
layoutMenu = viewToolBar.widgetForAction(viewToolBar.actions()[0]).menu()
layoutSwitchActionParent = layoutMenu  # use `layoutMenu` to add inside layout list, use␣
↪`viewToolBar` to add next the standard layout list
layoutSwitchAction = layoutSwitchActionParent.addAction("My view") # add inside layout␣
↪list
layoutSwitchAction.setData(customLayoutId)
layoutSwitchAction.setIcon(qt.QIcon(":Icons/Go.png"))
layoutSwitchAction.setToolTip("3D and slice view")
```

Turn on slice intersections

```
sliceDisplayNodes = slicer.util.getNodesByClass("vtkMRMLSliceDisplayNode")
for sliceDisplayNode in sliceDisplayNodes:
  sliceDisplayNode.SetIntersectingSlicesVisibility(1)

# Workaround to force visual update (see https://github.com/Slicer/Slicer/issues/6338)
sliceNodes = slicer.util.getNodesByClass('vtkMRMLSliceNode')
for sliceNode in sliceNodes:
    sliceNode.Modified()
```

Note

How to find code corresponding to a user interface widget?

For this one I searched for "slice intersections" text in the whole Slicer source code, found that the function is implemented in `Base\QTGUI\qSlicerViewersToolBar.cxx`, then translated the `qSlicerViewersToolBarPrivate::setSliceIntersectionVisible(bool visible)` method to Python.

Hide slice view annotations

This script can hide node name, patient information displayed in corners of slice views (managed by DataProbe module).

```
# Disable slice annotations immediately
sliceAnnotations = slicer.modules.DataProbeInstance.infoWidget.sliceAnnotations
sliceAnnotations.sliceViewAnnotationsEnabled=False
sliceAnnotations.updateSliceViewFromGUI()
# Disable slice annotations persistently (after Slicer restarts)
settings = qt.QSettings()
settings.setValue("DataProbe/sliceViewAnnotations.enabled", 0)
```

Change slice offset

Equivalent to moving the slider in slice view controller.

```
layoutManager = slicer.app.layoutManager()
red = layoutManager.sliceWidget("Red")
redLogic = red.sliceLogic()
# Print current slice offset position
print(redLogic.GetSliceOffset())
# Change slice position
redLogic.SetSliceOffset(20)
```

Change slice orientation

Get `Red` slice node and rotate around `X` and `Y` axes.

```
sliceNode = slicer.app.layoutManager().sliceWidget("Red").mrmlSliceNode()
sliceToRas = sliceNode.GetSliceToRAS()
transform=vtk.vtkTransform()
transform.SetMatrix(sliceToRas)
transform.RotateX(20)
transform.RotateY(15)
```

(continues on next page)

(continued from previous page)

```
sliceToRas.DeepCopy(transform.GetMatrix())
sliceNode.UpdateMatrices()
```

Measure angle between two slice planes

Measure angle between red and yellow slice nodes. Whenever any of the slice nodes are moved, the updated angle is printed on the console.

```
sliceNodeIds = ["vtkMRMLSliceNodeRed", "vtkMRMLSliceNodeYellow"]

# Print angles between slice nodes
def ShowAngle(unused1=None, unused2=None):
  sliceNormalVector = []
  for sliceNodeId in sliceNodeIds:
    sliceToRAS = slicer.mrmlScene.GetNodeByID(sliceNodeId).GetSliceToRAS()
    sliceNormalVector.append([sliceToRAS.GetElement(0,2), sliceToRAS.GetElement(1,2),␣
↪sliceToRAS.GetElement(2,2)])
  angleRad = vtk.vtkMath.AngleBetweenVectors(sliceNormalVector[0], sliceNormalVector[1])
  angleDeg = vtk.vtkMath.DegreesFromRadians(angleRad)
  print("Angle between slice planes = {0:0.3f}".format(angleDeg))

# Observe slice node changes
for sliceNodeId in sliceNodeIds:
  slicer.mrmlScene.GetNodeByID(sliceNodeId).AddObserver(vtk.vtkCommand.ModifiedEvent,␣
↪ShowAngle)

# Print current angle
ShowAngle()
```

Set slice position and orientation from a normal vector and position

This code snippet shows how to display a slice view defined by a normal vector and position in an anatomically sensible way: rotating slice view so that "up" direction (or "right" direction) is towards an anatomical axis.

```
def setSlicePoseFromSliceNormalAndPosition(sliceNode, sliceNormal, slicePosition,␣
↪defaultViewUpDirection=None, backupViewRightDirection=None):
  """
  Set slice pose from the provided plane normal and position. View up direction is␣
↪determined automatically,
  to make view up point towards defaultViewUpDirection.
  :param defaultViewUpDirection Slice view will be spinned in-plane to match point␣
↪approximately this up direction. Default: patient superior.
  :param backupViewRightDirection Slice view will be spinned in-plane to match point␣
↪approximately this right direction
    if defaultViewUpDirection is too similar to sliceNormal. Default: patient left.
  """
  # Fix up input directions
  if defaultViewUpDirection is None:
    defaultViewUpDirection = [0,0,1]
  if backupViewRightDirection is None:
    backupViewRightDirection = [-1,0,0]
  if sliceNormal[1]>=0:
```

(continues on next page)

(continued from previous page)

```
    sliceNormalStandardized = sliceNormal
  else:
    sliceNormalStandardized = [-sliceNormal[0], -sliceNormal[1], -sliceNormal[2]]
  # Compute slice axes
  sliceNormalViewUpAngle = vtk.vtkMath.AngleBetweenVectors(sliceNormalStandardized,␣
↪defaultViewUpDirection)
  angleTooSmallThresholdRad = 0.25 # about 15 degrees
  if sliceNormalViewUpAngle > angleTooSmallThresholdRad and sliceNormalViewUpAngle < vtk.
↪vtkMath.Pi() - angleTooSmallThresholdRad:
    viewUpDirection = defaultViewUpDirection
    sliceAxisY = viewUpDirection
    sliceAxisX = [0, 0, 0]
    vtk.vtkMath.Cross(sliceAxisY, sliceNormalStandardized, sliceAxisX)
  else:
    sliceAxisX = backupViewRightDirection
  # Set slice axes
  sliceNode.SetSliceToRASByNTP(sliceNormalStandardized[0], sliceNormalStandardized[1],␣
↪sliceNormalStandardized[2],
    sliceAxisX[0], sliceAxisX[1], sliceAxisX[2],
    slicePosition[0], slicePosition[1], slicePosition[2], 0)

# Example usage:
sliceNode = getNode("vtkMRMLSliceNodeRed")
transformNode = getNode("Transform_3")
transformMatrix = vtk.vtkMatrix4x4()
transformNode.GetMatrixTransformToParent(transformMatrix)
sliceNormal = [transformMatrix.GetElement(0,2), transformMatrix.GetElement(1,2),␣
↪transformMatrix.GetElement(2,2)]
slicePosition = [transformMatrix.GetElement(0,3), transformMatrix.GetElement(1,3),␣
↪transformMatrix.GetElement(2,3)]
setSlicePoseFromSliceNormalAndPosition(sliceNode, sliceNormal, slicePosition)
```

Show slice views in 3D window

Equivalent to clicking 'eye' icon in the slice view controller.

```
layoutManager = slicer.app.layoutManager()
for sliceViewName in layoutManager.sliceViewNames():
  controller = layoutManager.sliceWidget(sliceViewName).sliceController()
  controller.setSliceVisible(True)
```

Change default slice view orientation

You can left-right "flip" slice view orientation presets (show patient left side on left/right side of the screen) by copy-pasting the script below to your *slicerrc.py file*.

```
# Axial slice axes:
#   1 0 0
#   0 1 0
#   0 0 1
axialSliceToRas=vtk.vtkMatrix3x3()
```

(continues on next page)

(continued from previous page)

```
# Coronal slice axes:
#   1 0 0
#   0 0 -1
#   0 1 0
coronalSliceToRas=vtk.vtkMatrix3x3()
coronalSliceToRas.SetElement(1,1, 0)
coronalSliceToRas.SetElement(1,2, -1)
coronalSliceToRas.SetElement(2,1, 1)
coronalSliceToRas.SetElement(2,2, 0)

# Replace orientation presets in all existing slice nodes and in the default slice node
sliceNodes = slicer.util.getNodesByClass("vtkMRMLSliceNode")
sliceNodes.append(slicer.mrmlScene.GetDefaultNodeByClass("vtkMRMLSliceNode"))
for sliceNode in sliceNodes:
  orientationPresetName = sliceNode.GetOrientation()
  sliceNode.RemoveSliceOrientationPreset("Axial")
  sliceNode.AddSliceOrientationPreset("Axial", axialSliceToRas)
  sliceNode.RemoveSliceOrientationPreset("Coronal")
  sliceNode.AddSliceOrientationPreset("Coronal", coronalSliceToRas)
  sliceNode.SetOrientation(orientationPresetName)
```

Set all slice views linked by default

You can make slice views linked by default (when application starts or the scene is cleared) by copy-pasting the script below to your *.slicerrc.py file*.

```
# Set linked slice views  in all existing slice composite nodes and in the default node
sliceCompositeNodes = slicer.util.getNodesByClass("vtkMRMLSliceCompositeNode")
defaultSliceCompositeNode = slicer.mrmlScene.GetDefaultNodeByClass(
↪"vtkMRMLSliceCompositeNode")
if not defaultSliceCompositeNode:
  defaultSliceCompositeNode = slicer.mrmlScene.CreateNodeByClass(
↪"vtkMRMLSliceCompositeNode")
  defaultSliceCompositeNode.UnRegister(None)  # CreateNodeByClass is factory method,␣
↪need to unregister the result to prevent memory leaks
  slicer.mrmlScene.AddDefaultNode(defaultSliceCompositeNode)
sliceCompositeNodes.append(defaultSliceCompositeNode)
for sliceCompositeNode in sliceCompositeNodes:
  sliceCompositeNode.SetLinkedControl(True)
```

Set crosshair jump mode to centered by default

You can change default slice jump mode (when application starts or the scene is cleared) by copy-pasting the script below to your *.slicerrc.py file*.

```
crosshair=slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLCrosshairNode")
crosshair.SetCrosshairBehavior(crosshair.CenteredJumpSlice)
```

Set up custom units in slice view ruler

For microscopy or micro-CT images you may want to switch unit to micrometer instead of the default mm. To do that, 1. change the unit in Application settings / Units and 2. update ruler display settings using the script below (it can be copied to your Application startup script):

```
appLogic = slicer.app.applicationLogic()
layoutManagerLogic = slicer.app.layoutManager().layoutLogic()
for viewNode in layoutManagerLogic.GetViewNodes():
  if type(viewNode) is not slicer.vtkMRMLSliceNode:
    continue
  displayableManager = appLogic.GetViewDisplayableManagerByClassName(viewNode,
↪"vtkMRMLRulerDisplayableManager")
  displayableManager.RemoveAllRulerScalePresets()
  displayableManager.AddRulerScalePreset(   0.001, 5, 2, "nm", 1000.0)
  displayableManager.AddRulerScalePreset(   0.010, 5, 2, "nm", 1000.0)
  displayableManager.AddRulerScalePreset(   0.100, 5, 2, "nm", 1000.0)
  displayableManager.AddRulerScalePreset(   0.500, 5, 1, "nm", 1000.0)
  displayableManager.AddRulerScalePreset(   1.0,   5, 2, "um",    1.0)
  displayableManager.AddRulerScalePreset(   5.0,   5, 1, "um",    1.0)
  displayableManager.AddRulerScalePreset(  10.0,   5, 2, "um",    1.0)
  displayableManager.AddRulerScalePreset(  50.0,   5, 1, "um",    1.0)
  displayableManager.AddRulerScalePreset( 100.0,   5, 2, "um",    1.0)
  displayableManager.AddRulerScalePreset( 500.0,   5, 1, "um",    1.0)
  displayableManager.AddRulerScalePreset(1000.0,   5, 2, "mm",    0.001)
```

Center the 3D view on the scene

```
layoutManager = slicer.app.layoutManager()
threeDWidget = layoutManager.threeDWidget(0)
threeDView = threeDWidget.threeDView()
threeDView.resetFocalPoint()
```

Rotate the 3D View

```
layoutManager = slicer.app.layoutManager()
threeDWidget = layoutManager.threeDWidget(0)
threeDView = threeDWidget.threeDView()
threeDView.yaw()
```

Change 3D view background color

```
viewNode = slicer.app.layoutManager().threeDWidget(0).mrmlViewNode()
viewNode.SetBackgroundColor(1,0,0)
viewNode.SetBackgroundColor2(1,0,0)
```

Change box color in 3D view

```
viewNode = slicer.app.layoutManager().threeDWidget(0).mrmlViewNode()
viewNode.SetBoxColor(1,0,0)
```

Show a slice view outside the view layout

```
# layout name is used to create and identify the underlying slice node and  should be␣
↪set to a value that is not used in any of the layouts owned by the layout manager
layoutName = "TestSlice1"
layoutLabel = "TS1"
layoutColor = [1.0, 1.0, 0.0]
# ownerNode manages this view instead of the layout manager (it can be any node in the␣
↪scene)
viewOwnerNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScriptedModuleNode")

# Create MRML nodes
viewLogic = slicer.vtkMRMLSliceLogic()
viewLogic.SetMRMLScene(slicer.mrmlScene)
viewNode = viewLogic.AddSliceNode(layoutName)
viewNode.SetLayoutLabel(layoutLabel)
viewNode.SetLayoutColor(layoutColor)
viewNode.SetAndObserveParentLayoutNodeID(viewOwnerNode.GetID())

# Create widget
viewWidget = slicer.qMRMLSliceWidget()
viewWidget.setMRMLScene(slicer.mrmlScene)
viewWidget.setMRMLSliceNode(viewNode)
sliceLogics = slicer.app.applicationLogic().GetSliceLogics()
viewWidget.setSliceLogics(sliceLogics)
sliceLogics.AddItem(viewWidget.sliceLogic())
viewWidget.show()
```

Show a 3D view outside the view layout

```
# layout name is used to create and identify the underlying view node and  should be set␣
↪to a value that is not used in any of the layouts owned by the layout manager
layoutName = "Test3DView"
layoutLabel = "T3"
layoutColor = [1.0, 1.0, 0.0]
# ownerNode manages this view instead of the layout manager (it can be any node in the␣
↪scene)
viewOwnerNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScriptedModuleNode")

# Create MRML node
viewLogic = slicer.vtkMRMLViewLogic()
viewLogic.SetMRMLScene(slicer.mrmlScene)
viewNode = viewLogic.AddViewNode(layoutName)
viewNode.SetLayoutLabel(layoutLabel)
viewNode.SetLayoutColor(layoutColor)
viewNode.SetAndObserveParentLayoutNodeID(viewOwnerNode.GetID())

# Create widget
viewWidget = slicer.qMRMLThreeDWidget()
viewWidget.setMRMLScene(slicer.mrmlScene)
viewWidget.setMRMLViewNode(viewNode)
viewWidget.show()
```

12.8.6 Access VTK rendering classes

Access VTK views, renderers, and cameras

Iterate through all 3D views in current layout:

```
layoutManager = slicer.app.layoutManager()
for threeDViewIndex in range(layoutManager.threeDViewCount) :
  view = layoutManager.threeDWidget(threeDViewIndex).threeDView()
  threeDViewNode = view.mrmlViewNode()
  cameraNode = slicer.modules.cameras.logic().GetViewActiveCameraNode(threeDViewNode)
  print("View node for 3D widget " + str(threeDViewIndex))
  print("  Name: " + threeDViewNode .GetName())
  print("  ID: " + threeDViewNode .GetID())
  print("  Camera ID: " + cameraNode.GetID())
```

Iterate through all slice views in current layout:

```
layoutManager = slicer.app.layoutManager()
for sliceViewName in layoutManager.sliceViewNames():
  view = layoutManager.sliceWidget(sliceViewName).sliceView()
  sliceNode = view.mrmlSliceNode()
  sliceLogic = slicer.app.applicationLogic().GetSliceLogic(sliceNode)
  compositeNode = sliceLogic.GetSliceCompositeNode()
  print("Slice view " + str(sliceViewName))
  print("  Name: " + sliceNode.GetName())
  print("  ID: " + sliceNode.GetID())
  print("  Background volume: {0}".format(compositeNode.GetBackgroundVolumeID()))
  print("  Foreground volume: {0} (opacity: {1})".format(compositeNode.
↪GetForegroundVolumeID(), compositeNode.GetForegroundOpacity()))
  print("  Label volume: {0} (opacity: {1})".format(compositeNode.GetLabelVolumeID(),␣
↪compositeNode.GetLabelOpacity()))
```

For low-level manipulation of views, it is possible to access VTK render windows, renderers and cameras of views in the current layout.

```
renderWindow = view.renderWindow()
renderers = renderWindow.GetRenderers()
renderer = renderers.GetItemAsObject(0)
camera = cameraNode.GetCamera()
```

Get displayable manager of a certain type for a certain view

Displayable managers are responsible for creating VTK filters, mappers, and actors to display MRML nodes in renderers. Input to filters and mappers are VTK objects stored in MRML data nodes. Filter and actor properties are set based on display options specified in MRML display nodes.

Accessing displayable managers is useful for troubleshooting or for testing new features that are not exposed via MRML classes yet, as they provide usually allow low-level access to VTK actors.

```
threeDViewNode = slicer.mrmlScene.GetNodeByID("vtkMRMLViewNode1")
appLogic = slicer.app.applicationLogic()
modelDisplayableManager = appLogic.GetViewDisplayableManagerByClassName(threeDViewNode,
↪"vtkMRMLModelDisplayableManager")
if modelDisplayableManager is None:
  logging.error("Failed to find the model displayable manager")
```

Access VTK actor properties

This example shows how to access and modify VTK actor properties to experiment with physically-based rendering.

```
modelNode = slicer.util.getNode("MyModel")

threeDViewNode = slicer.mrmlScene.GetNodeByID("vtkMRMLViewNode1")
appLogic = slicer.app.applicationLogic()
modelDisplayableManager = appLogic.GetViewDisplayableManagerByClassName(threeDViewNode,
↪"vtkMRMLModelDisplayableManager")
actor=modelDisplayableManager.GetActorByID(modelNode.GetDisplayNode().GetID())
property=actor.GetProperty()
property.SetInterpolationToPBR()
property.SetMetallic(0.5)
property.SetRoughness(0.5)
property.SetColor(0.5,0.5,0.9)
slicer.util.forceRenderAllViews()
```

See more information on physically based rendering in VTK here: https://blog.kitware.com/vtk-pbr/

12.8.7 Keyboard shortcuts and mouse gestures

Customize keyboard shortcuts

Keyboard shortcuts can be specified for activating any Slicer feature by adding a couple of lines to your *.slicerrc.py file*.

For example, this script registers Ctrl+b, Ctrl+n, Ctrl+m, Ctrl+, keyboard shortcuts to switch between red, yellow, green, and 4-up view layouts.

```
shortcuts = [
  ("Ctrl+b", lambda: slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.
↪SlicerLayoutOneUpRedSliceView)),
  ("Ctrl+n", lambda: slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.
↪SlicerLayoutOneUpYellowSliceView)),
  ("Ctrl+m", lambda: slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.
↪SlicerLayoutOneUpGreenSliceView)),
  ("Ctrl+,", lambda: slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.
↪SlicerLayoutFourUpView))
  ]

for (shortcutKey, callback) in shortcuts:
  shortcut = qt.QShortcut(slicer.util.mainWindow())
  shortcut.setKey(qt.QKeySequence(shortcutKey))
  shortcut.connect( "activated()", callback)
```

Here's an example for cycling through Segment Editor effects (requested on the Slicer forum for the SlicerMorph project).

```
def cycleEffect(delta=1):
  try:
    orderedNames = list(slicer.modules.SegmentEditorWidget.editor.effectNameOrder())
    allNames = slicer.modules.SegmentEditorWidget.editor.availableEffectNames()
    for name in allNames:
      try:
```

(continues on next page)

(continued from previous page)

```
        orderedNames.index(name)
      except ValueError:
        orderedNames.append(name)
    orderedNames.insert(0, None)
    activeEffect = slicer.modules.SegmentEditorWidget.editor.activeEffect()
    if activeEffect:
      activeName = slicer.modules.SegmentEditorWidget.editor.activeEffect().name
    else:
      activeName = None
    newIndex = (orderedNames.index(activeName) + delta) % len(orderedNames)
    slicer.modules.SegmentEditorWidget.editor.
↪setActiveEffectByName(orderedNames[newIndex])
  except AttributeError:
    # module not active
    pass

shortcuts = [
  ("`", lambda: cycleEffect(-1)),
  ("~", lambda: cycleEffect(1)),
  ]

for (shortcutKey, callback) in shortcuts:
  shortcut = qt.QShortcut(slicer.util.mainWindow())
  shortcut.setKey(qt.QKeySequence(shortcutKey))
  shortcut.connect( "activated()", callback)
```

Customize keyboard/mouse gestures in viewers

Make the 3D view rotate using right-click-and-drag

```
threeDViewNode = slicer.mrmlScene.GetNodeByID("vtkMRMLViewNode1")
appLogic = slicer.app.applicationLogic()
cameraDisplayableManager = appLogic.GetViewDisplayableManagerByClassName(threeDViewNode,
↪"vtkMRMLCameraDisplayableManager")
cameraWidget = cameraDisplayableManager.GetCameraWidget()

# Remove old mapping from right-click-and-drag
cameraWidget.SetEventTranslationClickAndDrag(cameraWidget.WidgetStateIdle, vtk.
↪vtkCommand.RightButtonPressEvent, vtk.vtkEvent.NoModifier,
  cameraWidget.WidgetStateRotate, vtk.vtkWidgetEvent.NoEvent, vtk.vtkWidgetEvent.NoEvent)

# Make right-click-and-drag rotate the view
cameraWidget.SetEventTranslationClickAndDrag(cameraWidget.WidgetStateIdle, vtk.
↪vtkCommand.RightButtonPressEvent, vtk.vtkEvent.NoModifier,
  cameraWidget.WidgetStateRotate, cameraWidget.WidgetEventRotateStart, cameraWidget.
↪WidgetEventRotateEnd)
```

Custom shortcut for moving crosshair in a slice view

```
# Red slice view
sliceViewLabel = "Red"
sliceViewNode = slicer.mrmlScene.GetNodeByID(f"vtkMRMLSliceNode{sliceViewLabel}")
appLogic = slicer.app.applicationLogic()
displayableManager = appLogic.GetViewDisplayableManagerByClassName(sliceViewNode,
↪"vtkMRMLCrosshairDisplayableManager")

widget = displayableManager.GetSliceIntersectionWidget()

# Set crosshair position by left-click
widget.SetEventTranslation(widget.WidgetStateIdle, slicer.vtkMRMLInteractionEventData.
↪LeftButtonClickEvent, vtk.vtkEvent.NoModifier, widget.WidgetEventSetCrosshairPosition)
widget.SetEventTranslation(widget.WidgetStateIdle, slicer.vtkMRMLInteractionEventData.
↪LeftButtonClickEvent, vtk.vtkEvent.NoModifier, widget.WidgetEventSetCrosshairPosition)

# Move crosshair by Alt+left-click-and-drag
widget.SetEventTranslationClickAndDrag(widget.WidgetStateIdle, vtk.vtkCommand.
↪LeftButtonPressEvent, vtk.vtkEvent.AltModifier,
  widget.WidgetStateMoveCrosshair, widget.WidgetEventMoveCrosshairStart, widget.
↪WidgetEventMoveCrosshairEnd)
```

Custom shortcut for moving crosshair in a 3D view

```
# 3D view
threeDViewNode = slicer.mrmlScene.GetNodeByID("vtkMRMLViewNode1")
appLogic = slicer.app.applicationLogic()
cameraDisplayableManager = appLogic.GetViewDisplayableManagerByClassName(threeDViewNode,
↪"vtkMRMLCameraDisplayableManager")
widget = cameraDisplayableManager.GetCameraWidget()

# Set crosshair position by left-click
widget.SetEventTranslation(widget.WidgetStateIdle, slicer.vtkMRMLInteractionEventData.
↪LeftButtonClickEvent, vtk.vtkEvent.NoModifier, widget.WidgetEventSetCrosshairPosition)

# Move crosshair by Alt+left-click-and-drag
widget.SetEventTranslationClickAndDrag(widget.WidgetStateIdle, vtk.vtkCommand.
↪LeftButtonPressEvent, vtk.vtkEvent.AltModifier,
  widget.WidgetStateMoveCrosshair, widget.WidgetEventMoveCrosshairStart, widget.
↪WidgetEventMoveCrosshairEnd)
```

Add shortcut to adjust window/level in any mouse mode

Makes Ctrl + Right-click-and-drag gesture adjust window/level in red slice view. This gesture works even when not in "Adjust window/level" mouse mode.

```
sliceViewLabel = "Red"
sliceViewNode = slicer.mrmlScene.GetNodeByID(f"vtkMRMLSliceNode{sliceViewLabel}")
appLogic = slicer.app.applicationLogic()
displayableManager = appLogic.GetViewDisplayableManagerByClassName(sliceViewNode,
↪"vtkMRMLScalarBarDisplayableManager")
```

(continues on next page)

(continued from previous page)

```
w = displayableManager.GetWindowLevelWidget()
w.SetEventTranslationClickAndDrag(w.WidgetStateIdle,
  vtk.vtkCommand.RightButtonPressEvent, vtk.vtkEvent.ControlModifier,
  w.WidgetStateAdjustWindowLevel, w.WidgetEventAlwaysOnAdjustWindowLevelStart, w.
↪WidgetEventAlwaysOnAdjustWindowLevelEnd)
```

Disable certain user interactions in slice views

For example, disable slice browsing using mouse wheel and keyboard shortcuts in the red slice viewer:

```
interactorObserver = slicer.app.layoutManager().sliceWidget("Red").sliceView().
↪interactorObserver()
interactorObserver.SetActionEnabled(interactorStyle.BrowseSlice, False)
```

Hide all slice view controllers:

```
lm = slicer.app.layoutManager()
for sliceViewName in lm.sliceViewNames():
  lm.sliceWidget(sliceViewName).sliceController().setVisible(False)
```

Hide all 3D view controllers:

```
lm = slicer.app.layoutManager()
for viewIndex in range(slicer.app.layoutManager().threeDViewCount):
  lm.threeDWidget(0).threeDController().setVisible(False)
```

Add keyboard shortcut to jump to center or world coordinate system

You can copy-paste this into the Python console to jump slice views to (0,0,0) position on (Ctrl+e):

```
shortcut = qt.QShortcut(qt.QKeySequence("Ctrl+e"), slicer.util.mainWindow())
shortcut.connect("activated()",
  lambda: slicer.modules.markups.logic().JumpSlicesToLocation(0,0,0, True))
```

12.8.8 Launch external applications

How to run external applications from Slicer.

Launch external process in startup environment

When a process is launched from Slicer then by default Slicer"s ITK, VTK, Qt, etc. libraries are used. If an external application has its own version of these libraries, then the application is expected to crash. To prevent crashing, the application must be run in the environment where Slicer started up (without all Slicer-specific library paths). This startup environment can be retrieved using *slicer.util.startupEnvironment()*.

Example: run Python3 script from Slicer:

```
command_to_execute = ["/usr/bin/python3", "-c", "print("hola")"]
from subprocess import check_output
check_output(
  command_to_execute,
  env=slicer.util.startupEnvironment()
  )
```

will output:

```
"hola\n"
```

On some systems, *shell=True* must be specified as well.

12.8.9 Manage extensions

Download and install extension

Added in version 5.4: Slicer introduces `slicer.app.installExtensionFromServer()`, which simplifies the process of downloading and installing extensions. The updated approach is as follows:

```
extensionName = 'SlicerIGT'
em = slicer.app.extensionsManagerModel()
em.interactive = False  # prevent display of popups
restart = True
if not em.installExtensionFromServer(extensionName, restart):
  raise ValueError(f"Failed to install {extensionName} extension")
```

Install a module directly from a git repository

This code snippet can be useful for sharing code in development without requiring a restart of Slicer.

Install a Python package

It is recommended to only install a package at runtime when it is actually needed (not at startup, not even when the user opens a module, but just before that Python package is used the first time), and ask the user about it. For more comprehensive guidelines, refer to the *best practices*.

Recommended approach using pip_ensure

Added in version 5.11: The `slicer.packaging` module and the `pip_ensure` workflow described below.

The `slicer.packaging.pip_ensure()` function handles checking, prompting, and installing in one call with a progress dialog:

```
import slicer.packaging

class MyModuleWidget(ScriptedLoadableModuleWidget):

    def onApplyButton(self):
        # For a single package (or space-separated list):
        slicer.packaging.pip_ensure("flywheel-sdk>=1.0", requester="MyModule")

        # Or load requirements from file (recommended for multiple dependencies):
        # reqs = slicer.packaging.load_requirements(self.resourcePath("requirements.txt
↪"))
        # slicer.packaging.pip_ensure(reqs, requester="MyModule")

        import flywheel
        # Now safe to use flywheel
```

Key features of `pip_ensure()`:

- Shows a confirmation dialog before installing (controlled by `prompt_install`, default `True`)

- Shows a progress dialog during installation with collapsible log details
- Detects when an updated package was already imported in the current session and offers a restart prompt (controlled by `prompt_restart`, default `True`)
- Automatically skips installation in testing mode (`slicer.app.testingEnabled()`)

Alternative: pip_install with progress dialog

For direct installation with visual feedback without using `pip_ensure`. Note that `pip_install` now shows a progress dialog by default:

```
try:
  import flywheel
except ModuleNotFoundError:
  if slicer.util.confirmOkCancelDisplay("This module requires 'flywheel-sdk' Python␣
↪package. Click OK to install it now."):
    slicer.packaging.pip_install("flywheel-sdk", requester="MyModule")
    import flywheel
```

Simple approach (no progress dialog)

For scripting or automation where no visual feedback is needed:

```
try:
  import flywheel
except ModuleNotFoundError:
  if slicer.util.confirmOkCancelDisplay("This module requires 'flywheel-sdk' Python␣
↪package. Click OK to install it now."):
    slicer.packaging.pip_install("flywheel-sdk", show_progress=False)
    import flywheel
```

12.8.10 Python package management

The `slicer.packaging` module provides functions for checking, installing, and managing Python packages in the Slicer environment. For best practices on when and how to install packages, see *Can I use any Python package in a Slicer module?*.

Added in version 5.11: The `slicer.packaging` module with `pip_ensure`, `pip_check`, `load_requirements`, and `load_pyproject_dependencies`.

Tip

When writing a Slicer module, prefer *`slicer.packaging.pip_ensure()`* over *`slicer.packaging.pip_install()`*. `pip_ensure` checks whether each requirement is already satisfied (skipping the install if so), prompts the user before modifying the environment, and detects when an updated package was already imported in the current session and offers a restart prompt. `pip_install` is the lower-level building block and does none of those things on its own.

Check if packages are installed

Use `pip_check` to test whether requirements are already satisfied without launching a subprocess:

```python
import slicer.packaging

# Single requirement
if slicer.packaging.pip_check("scipy>=1.0"):
    print("scipy is available")

# Multiple requirements (space-separated)
if slicer.packaging.pip_check("scipy>=1.0 numpy>=1.20"):
    print("All available")

# Multiple requirements from a file
reqs = slicer.packaging.load_requirements("/path/to/requirements.txt")
if not slicer.packaging.pip_check(reqs):
    print("Some requirements are missing")
```

Load requirements from a file

You can keep your dependencies in a `requirements.txt` file and load them with *slicer.packaging.load_requirements()*:

```python
reqs = slicer.packaging.load_requirements(self.resourcePath("requirements.txt"))
slicer.packaging.pip_ensure(reqs, requester="MyExtension")
```

If your extension already has a `pyproject.toml`, you can read the `[project.dependencies]` list with *slicer.packaging.load_pyproject_dependencies()* instead. Both functions return the same `list[Requirement]` type, so the rest of the pipeline (`pip_check`, `pip_ensure`) works identically:

```python
reqs = slicer.packaging.load_pyproject_dependencies(self.resourcePath("pyproject.toml"))
slicer.packaging.pip_ensure(reqs, requester="MyExtension")
```

Install packages with a constraints file

A constraints file limits which versions pip may install without triggering installation on its own. This is useful for ensuring compatible versions across multiple extensions.

```python
import slicer.packaging

# Via pip_ensure (recommended)
reqs = slicer.packaging.load_requirements("/path/to/requirements.txt")
slicer.packaging.pip_ensure(
    reqs,
    constraints="/path/to/constraints.txt",
    requester="MyExtension",
)
```

```python
# Via pip_install (lower level)
slicer.packaging.pip_install("pandas scipy", constraints="/path/to/constraints.txt")
```

Install packages with broken dependency declarations

When a package declares overly strict dependencies that conflict with other packages, use `no_deps_requirements` to install it without its declared dependencies, then install the actual dependencies you need separately:

```
slicer.packaging.pip_install(
    requirements="numpy scipy",
    no_deps_requirements="problematic-pkg==1.0",
)
```

Excluding specific transitive dependencies (skip_packages)

skip_packages installs each requirement with --no-deps, walks its dependency tree recursively, and excludes any package matching the skip list. Package METADATA is scrubbed afterward so pip will not flag the skipped packages as missing.

It exists for the specific case where a package pulls in a transitive dependency that Slicer already provides differently, such as SimpleITK (Slicer bundles a custom build) or torch (must come from SlicerPyTorch for the correct CUDA/CPU combination). Without skip_packages, installing such a package would overwrite or conflict with Slicer's bundled version.

```
import slicer.packaging

skipped = slicer.packaging.pip_ensure(
    "nnunetv2>=2.3",
    skip_packages=["SimpleITK", "torch", "requests"],
    requester="SlicerNNUNet",
)
# skipped contains the requirement strings that were excluded,
# e.g. ["torch>=2.0", "SimpleITK>=2.0.2", "requests"]
```

Warning

skip_packages is a workaround, not a recommended pattern. It hides unwanted requirements from pip, which means real conflicts are no longer reported and pip show no longer reflects the true dependency graph. Skip lists also drift over time as Slicer's bundled packages change.

Where possible, consider asking the upstream maintainer to relax the unnecessary requirement, or use no_deps_requirements if the dependency tree is small enough to enumerate yourself. For packages with deep transitive dependency trees that conflict with Slicer's bundled libraries (e.g. nnUNet pulling in torch and SimpleITK), skip_packages is often the only practical option short of forking the package – which is why it exists.

skip_packages and no_deps_requirements are mutually exclusive.

Non-blocking package installation

Use blocking=False to install packages without blocking the UI. Pip output appears in the status bar by default, and you can provide callbacks for custom handling:

```
def onComplete(returnCode):
    if returnCode == 0:
        import pandas  # Now safe to import
    else:
        slicer.util.errorDisplay("Failed to install packages")

slicer.packaging.pip_install(
    "pandas scipy",
```

(continues on next page)

(continued from previous page)

```
    blocking=False,
    completedCallback=onComplete,
)
# Returns immediately - UI stays responsive
```

Warning

When using `blocking=False`, the user can interact with the application while installation is in progress. Consider showing a modal dialog or disabling relevant UI elements during installation to prevent conflicts.

Check if installation is in progress

Before starting an operation that might conflict with an ongoing pip installation, check the in-progress flag:

```
import slicer.packaging

if slicer.packaging.isPipInstallInProgress():
    slicer.util.warningDisplay("Package installation is in progress. Please wait.")
else:
    slicer.packaging.pip_install("scipy", blocking=False, requester="MyExtension")
```

12.8.11 DICOM

Load DICOM files into the scene from a folder

This code loads all DICOM objects into the scene from a file folder. All the registered plugins are evaluated and the one with the highest confidence will be used to load the data. Files are imported into a temporary DICOM database, so the current Slicer DICOM database is not impacted.

```
dicomDataDir = "c:/my/folder/with/dicom-files"  # input folder with DICOM files
loadedNodeIDs = []  # this list will contain the list of all loaded node IDs

from DICOMLib import DICOMUtils
with DICOMUtils.TemporaryDICOMDatabase() as db:
  DICOMUtils.importDicom(dicomDataDir, db)
  patientUIDs = db.patients()
  for patientUID in patientUIDs:
    loadedNodeIDs.extend(DICOMUtils.loadPatientByUID(patientUID))
```

Import DICOM files into the application's DICOM database

This code snippet uses Slicer DICOM browser built-in indexer to import DICOM files into the database. Images are not loaded into the scene, but they show up in the DICOM browser. After import, data sets can be loaded using DICOMUtils functions (e.g., loadPatientByUID) - see above for an example.

```
# instantiate a new DICOM browser
slicer.util.selectModule("DICOM")
dicomBrowser = slicer.modules.DICOMWidget.browserWidget.dicomBrowser
# use dicomBrowser.ImportDirectoryCopy to make a copy of the files (useful for importing␣
↪data from removable storage)
dicomBrowser.importDirectory(dicomFilesDirectory, dicomBrowser.ImportDirectoryAddLink)
```

(continues on next page)

(continued from previous page)

```
# wait for import to finish before proceeding (optional, if removed then import runs in␣
↪the background)
dicomBrowser.waitForImportFinished()
```

Import DICOM files using DICOMweb

Download and import DICOM data set using DICOMweb from a Picture Archiving and Communications System (PACS) such as Kheops, Google Health API, Orthanc, DCM4CHE, etc.

```
slicer.util.selectModule("DICOM")  # ensure DICOM database is initialized
slicer.app.processEvents()
from DICOMLib import DICOMUtils
DICOMUtils.importFromDICOMWeb(
  dicomWebEndpoint="https://demo.kheops.online/api",
  studyInstanceUID="1.3.6.1.4.1.14519.5.2.1.8421.4009.985792766370191766692237040819")
```

Authenticate with an Access Token

Several PACS solutions support remote access authentication with an access token.

How to obtain your access token:

- Google Cloud: Execute `gcloud auth print-access-token` once you have logged in
- Kheops: create an album, create a sharing link (something like `https://demo.kheops.online/view/TfYXwbKAW7JYbAgZ7MyISf`), the token is the string after the last slash (`TfYXwbKAW7JYbAgZ7MyISf`).

```
slicer.util.selectModule("DICOM")  # ensure DICOM database is initialized and
slicer.app.processEvents()
from DICOMLib import DICOMUtils
DICOMUtils.importFromDICOMWeb(
  dicomWebEndpoint="https://demo.kheops.online/api",
  studyInstanceUID="1.3.6.1.4.1.14519.5.2.1.8421.4009.985792766370191766692237040819",
  accessToken="TfYXwbKAW7JYbAgZ7MyISf")
```

Alternate Authentication Approaches

You can provide expanded authentication information to use in the DICOMweb request. Authentication types extending the Python `requests.auth.AuthBase` are accepted.

In the example below we provide a basic username and password as a `requests.HTTPBasicAuth` instance with the DICOMweb import request.

```
DICOMUtils.importFromDICOMWeb(
  dicomWebEndpoint="https://demo.kheops.online/api",
  studyInstanceUID="1.3.6.1.4.1.14519.5.2.1.8421.4009.985792766370191766692237040819",
  auth=requests.HTTPBasicAuth('<user>','<password>'))
```

See the Python `requests` Authentication documentation for more information.

Configure a Global DICOMweb Authentication

You can set a global username and password combination in your local Slicer application to be remembered across application sessions. `DICOMUtils.getGlobalDICOMAuth` provides a convenient way to create a `HTTPBasicAuth` instance from the global configuration with each call.

```
qt.QSettings().setValue(DICOMUtils.GLOBAL_DICOMWEB_USER_KEY, '<user>')
qt.QSettings().setValue(DICOMUtils.GLOBAL_DICOMWEB_PASSWORD_KEY, '<pwd>')
DICOMUtils.importFromDICOMWeb(
  dicomWebEndpoint="https://remote-url/",
  studyInstanceUID="1.3.6.1.4.1.14519.5.2.1.8421.4009.985792766370191766692237040819",
  auth=DICOMUtils.getGlobalDICOMAuth()
)
```

Access top level tags of DICOM images imported into Slicer

For example, to print the first patient's first study's first series' "0020,0032" field:

```
db = slicer.dicomDatabase
patientList = db.patients()
studyList = db.studiesForPatient(patientList[0])
seriesList = db.seriesForStudy(studyList[0])
fileList = db.filesForSeries(seriesList[0])
# Note, fileValue accesses the database of cached top level tags
# (nested tags are not included)
print(db.fileValue(fileList[0], "0020,0032"))
# Get tag group,number from dicom dictionary
import pydicom as dicom
tagName = "StudyDate"
tagStr = str(dicom.tag.Tag(tagName))[1:-1].replace(" ","")
print(db.fileValue(fileList[0], tagStr))
```

Access DICOM tags nested in a sequence

```
db = slicer.dicomDatabase
patientList = db.patients()
studyList = db.studiesForPatient(patientList[0])
seriesList = db.seriesForStudy(studyList[0])
fileList = db.filesForSeries(seriesList[0])
# Use pydicom to access the full header, which requires
# re-reading the dataset instead of using the database cache
import pydicom
ds = pydicom.dcmread(fileList[0])
ds.CTExposureSequence[0].ExposureModulationType
```

Access tag of a scalar volume loaded from DICOM

Volumes loaded by `DICOMScalarVolumePlugin` store `SOP Instance UIDs` in the volume node's `DICOM.instanceUIDs` attribute. For example, this can be used to get the patient position stored in a volume:

```
volumeName = "2: ENT IMRT"
volumeNode = slicer.util.getNode(volumeName)
instUids = volumeNode.GetAttribute("DICOM.instanceUIDs").split()
```

(continues on next page)

(continued from previous page)

```
filename = slicer.dicomDatabase.fileForInstance(instUids[0])
print(slicer.dicomDatabase.fileValue(filename, "0018,5100"))  # patient position
```

Access tag of an item in the Subject Hierarchy tree

Data sets loaded by various DICOM plugins may not use `DICOM.instanceUIDs` attribute but instead they save the `Series Instance UID` to the subject hierarchy item. The `SOP Instance UIDs` can be retrieved based on the series instance UID, which then can be used to retrieve DICOM tags:

```
volumeName = "2: ENT IMRT"
volumeNode = slicer.util.getNode(volumeName)

# Get series instance UID from subject hierarchy
shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
volumeItemId = shNode.GetItemByDataNode(volumeNode)
seriesInstanceUID = shNode.GetItemUID(volumeItemId, 'DICOM')

# Get patient name (0010,0010) from the first file of the series
instUids = slicer.dicomDatabase.instancesForSeries(seriesInstanceUID)
print(slicer.dicomDatabase.instanceValue(instUids[0], '0010,0010')) # patient name
```

Another example, using referenced instance UIDs to get the content time tag of a structure set:

```
rtStructName = "3: RTSTRUCT: PROS"
rtStructNode = slicer.util.getNode(rtStructName)
shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
rtStructShItemID = shNode.GetItemByDataNode(rtStructNode)
ctSliceInstanceUids = shNode.GetItemAttribute(rtStructShItemID, "DICOM.
↪ReferencedInstanceUIDs").split()
filename = slicer.dicomDatabase.fileForInstance(ctSliceInstanceUids[0])
print(slicer.dicomDatabase.fileValue(filename, "0008,0033"))  # content time
```

Get path and filename of a scalar volume node loaded from DICOM

```
def pathFromNode(node):
  storageNode = node.GetStorageNode()
  if storageNode is not None: # loaded via drag-drop
    filepath = storageNode.GetFullNameFromFileName()
  else: # Loaded via DICOM browser
    instanceUIDs = node.GetAttribute("DICOM.instanceUIDs").split()
    filepath = slicer.dicomDatabase.fileForInstance(instanceUIDs[0])
  return filepath

# Example:
node = slicer.util.getNode("volume1")
path = pathFromNode(node)
print("DICOM path=%s" % path)
```

Convert DICOM to NRRD on the command line

```
/Applications/Slicer-4.6.2.app/Contents/MacOS/Slicer --no-main-window --python-code
→"node=slicer.util.loadVolume('/tmp/series/im0.dcm'); slicer.util.saveNode(node, '/tmp/
→output.nrrd'); exit()"
```

The same can be done on windows by using the top level Slicer.exe. Be sure to use forward slashes in the pathnames within quotes on the command line.

Export a volume to DICOM file format

```
volumeNode = getNode("CTChest")
outputFolder = "c:/tmp/dicom-output"

# Create patient and study and put the volume under the study
shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
# set IDs. Note: these IDs are not specifying DICOM tags, but only the names that appear␣
→in the hierarchy tree
patientItemID = shNode.CreateSubjectItem(shNode.GetSceneItemID(), "test patient")
studyItemID = shNode.CreateStudyItem(patientItemID, "test study")
volumeShItemID = shNode.GetItemByDataNode(volumeNode)
shNode.SetItemParent(volumeShItemID, studyItemID)

import DICOMScalarVolumePlugin
exporter = DICOMScalarVolumePlugin.DICOMScalarVolumePluginClass()
exportables = exporter.examineForExport(volumeShItemID)
for exp in exportables:
  # set output folder
  exp.directory = outputFolder
  # here we set DICOM PatientID and StudyID tags
  exp.setTag('PatientID', "test patient")
  exp.setTag('StudyID', "test study")

exporter.export(exportables)
```

Export a segmentation to DICOM segmentation object

```
segmentationNode = ...
referenceVolumeNode = ...
outputFolder = "c:/tmp/dicom-output"

# Associate segmentation node with a reference volume node
shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
referenceVolumeShItem = shNode.GetItemByDataNode(referenceVolumeNode)
studyShItem = shNode.GetItemParent(referenceVolumeShItem)
segmentationShItem = shNode.GetItemByDataNode(segmentationNode)
shNode.SetItemParent(segmentationShItem, studyShItem)

# Export to DICOM
import DICOMSegmentationPlugin
exporter = DICOMSegmentationPlugin.DICOMSegmentationPluginClass()
exportables = exporter.examineForExport(segmentationShItem)
for exp in exportables:
```

(continues on next page)

(continued from previous page)

```
  exp.directory = outputFolder

exporter.export(exportables)
```

Export DICOM series from the database to research file format

You can export the entire Slicer DICOM database content to nrrd (or nifti, etc.) file format with filtering of data type and naming of the output file based on DICOM tags like this:

```
outputFolder = "c:/tmp/exptest/"

from DICOMLib import DICOMUtils
patientUIDs = slicer.dicomDatabase.patients()
for patientUID in patientUIDs:
    loadedNodeIDs = DICOMUtils.loadPatientByUID(patientUID)
    for loadedNodeID in loadedNodeIDs:
        # Check if we want to save this node
        node = slicer.mrmlScene.GetNodeByID(loadedNodeID)
        # Only export images
        if not node or not node.IsA('vtkMRMLScalarVolumeNode'):
            continue
        # Construct filename
        shNode = slicer.mrmlScene.GetSubjectHierarchyNode()
        seriesItem = shNode.GetItemByDataNode(node)
        studyItem = shNode.GetItemParent(seriesItem)
        patientItem = shNode.GetItemParent(studyItem)
        filename = shNode.GetItemAttribute(patientItem, 'DICOM.PatientID')
        filename += '_' + shNode.GetItemAttribute(studyItem, 'DICOM.StudyDate')
        filename += '_' + shNode.GetItemAttribute(seriesItem, 'DICOM.SeriesNumber')
        filename += '_' + shNode.GetItemAttribute(seriesItem, 'DICOM.Modality')
        filename = slicer.app.ioManager().forceFileNameValidCharacters(filename) + ".nrrd
↪"
        # Save node
        print(f'Write {node.GetName()} to {filename}')
        success = slicer.util.saveNode(node, outputFolder+filename)
    slicer.mrmlScene.Clear()
```

Customize table columns in DICOM browser

Documentation of methods for changing DICOM browser columns: https://github.com/commontk/CTK/blob/master/Libs/DICOM/Core/ctkDICOMDatabase.h#L354-L375

```
# Get browser and database
dicomBrowser = slicer.modules.dicom.widgetRepresentation().self().browserWidget.
↪dicomBrowser
dicomDatabase = dicomBrowser.database()

# Print list of available columns
print(dicomDatabase.patientFieldNames)
print(dicomDatabase.studyFieldNames)
print(dicomDatabase.seriesFieldNames)
```

(continues on next page)

(continued from previous page)

```
# Change column order
dicomDatabase.setWeightForField("Series", "SeriesDescription", 7)
dicomDatabase.setWeightForField("Studies", "StudyDescription", 6)
# Change column visibility
dicomDatabase.setVisibilityForField("Patients", "PatientsBirthDate", False)
dicomDatabase.setVisibilityForField("Patients", "PatientsComments", True)
dicomDatabase.setWeightForField("Patients", "PatientsComments", 8)
# Change column name
dicomDatabase.setDisplayedNameForField("Series", "DisplayedCount", "Number of images")
# Change column width to manual
dicomDatabase.setFormatForField("Series", "SeriesDescription", '{"resizeMode":
↪"interactive"}')
# Customize table manager in DICOM browser
dicomTableManager = dicomBrowser.dicomTableManager()
dicomTableManager.selectionMode = qt.QAbstractItemView.SingleSelection
dicomTableManager.autoSelectSeries = False

# Force database views update
dicomDatabase.closeDatabase()
dicomDatabase.openDatabase(dicomBrowser.database().databaseFilename)
```

Query and retrieve data from a PACS using classic DIMSE DICOM networking

```
# Query
dicomQuery = ctk.ctkDICOMQuery()
dicomQuery.callingAETitle = "SLICER"
dicomQuery.calledAETitle = "ANYAE"
dicomQuery.host = "dicomserver.co.uk"
dicomQuery.port = 11112
# Change filter parameters in the next line if
# query does not find any series (try to use a different letter for "Name", such as "E")
# or there are too many hits (try to make "Name" more specific, such as "An").
dicomQuery.setFilters({"Name":"A", "Modalities":"MR"})
# temporary in-memory database for storing query results
tempDb = ctk.ctkDICOMDatabase()
tempDb.openDatabase("")
dicomQuery.query(tempDb)

# Retrieve
# Enable useCGET to retrieve using the query's connection (using C-GET).
# C-GET is simple, as it does not require configuring a DICOM receiver
# but C-GET is rarely allowed in clinical PACS.
# If useCGET is disabled then retrieve requests the PACS to send the data (using C-STORE)
# to Slicer. Slicer's AE title must be configured in the PACS settings. Slicer must have␣
↪its
# DICOM receiver (C-STORE SCP) running.
useCGET = True
dicomRetrieve = ctk.ctkDICOMRetrieve()
dicomRetrieve.callingAETitle = dicomQuery.callingAETitle
dicomRetrieve.calledAETitle = dicomQuery.calledAETitle
dicomRetrieve.host = dicomQuery.host
```

(continues on next page)

(continued from previous page)

```
dicomRetrieve.port = dicomQuery.port
dicomRetrieve.setDatabase(slicer.dicomDatabase)
for study, series in dicomQuery.studyAndSeriesInstanceUIDQueried:
  print(f"ctkDICOMRetrieveTest: Retrieving {study} - {series}")
  slicer.app.processEvents()
  if useCGET:
    success = dicomRetrieve.getSeries(study, series)
  else:
    dicomRetrieve.moveDestinationAETitle = dicomQuery.callingAETitle
    success = dicomRetrieve.moveSeries(study, series)
  print(f"  - {'success' if success else 'failed'}")

slicer.dicomDatabase.updateDisplayedFields()
```

Query and retrieve data from a PACS using classic DIMSE DICOM networking with the (experimental) ctkDICOMVisualBrowser

```

# Get visual browser instance
visualBrowser = slicer.modules.dicom.widgetRepresentation().self().browserWidget.
↪dicomVisualBrowser
dicomDatabase = visualBrowser.dicomDatabase()

# Disable query/retrieve for all existing servers
for index in range (0, visualBrowser.serversCount()):
  server = visualBrowser.server(index)
  server.queryRetrieveEnabled = False

# Add a new DICOM server
server = ctk.ctkDICOMServer()
server.connectionName = "test"
server.callingAETitle = "SLICER"
server.calledAETitle = "ANYAE"
server.host = "dicomserver.co.uk"
server.port = 104
server.retrieveProtocol = ctk.ctkDICOMServer.CGET

if visualBrowser.addServer(server) == -1:
  raise RuntimeError("Failed to add server")

# Set the filters for the query
visualBrowser.filteringPatientID = "PAT020"
#visualBrowser.filteringPatientName = "Name"
#visualBrowser.filteringStudyDescription = "Study description"
visualBrowser.filteringDate = ctk.ctkDICOMPatientItemWidget.LastYear
#Date options:
#Any,
#Today,
#Yesterday,
#LastWeek,
#LastMonth,
```

(continues on next page)

(continued from previous page)

```
#LastYear
#visualBrowser.filteringSeriesDescription = "Series description"
#visualBrowser.filteringModalities = ["CT", "MR"]

# Run patient query.
# NOTE: this will automatically also start query/retrieve jobs at study and series levels
visualBrowser.onQueryRetrieveOptionToggled(True)
visualBrowser.onQueryPatients()
```

Query and retrieve data from a PACS using classic DIMSE DICOM networking with the (experimental) ctkDICOMScheduler (no UI needed)

```
class Receiver(qt.QObject):
  def __init__(self, scheduler):
    super().__init__()
    self.scheduler = scheduler
    self.scheduler.progressJobDetail.connect(self.onProgressDetails)
    self.scheduler.jobFinished.connect(self.onJobFinished)
    self.scheduler.jobFailed.connect(self.onJobFailed)

  def startQueryRetrieve(self, parameters):
    self.scheduler.setFilters(parameters)
    self.scheduler.queryPatients()

  def onJobFinished(self, details):
    for detail in details:
      if detail.jobType() == ctk.ctkDICOMJobResponseSet.QueryPatients:
        print ("Query patients success. Connection : ", detail.connectionName())
      elif detail.jobType() == ctk.ctkDICOMJobResponseSet.QueryStudies:
        patientID = detail.patientID()
        print ("Query studies success for patientID: ", patientID, ". Connection : ",␣
↪detail.connectionName())
      elif detail.jobType() == ctk.ctkDICOMJobResponseSet.RetrieveStudy:
        patientID = detail.patientID()
        studyInstanceUID = detail.studyInstanceUID()
        print ("Retrieve studies success for studyInstanceUID: ", studyInstanceUID, "␣
↪(patientID: ",patientID, "). Connection : ", detail.connectionName())

  def onJobFailed(self, details):
    for detail in details:
      if detail.jobType() == ctk.ctkDICOMJobResponseSet.QueryPatients:
        print ("Query patients failed. Connection : ", detail.connectionName())
      elif detail.jobType() == ctk.ctkDICOMJobResponseSet.QueryStudies:
        patientID = detail.patientID()
        print ("Query studies failed for patientID: ", patientID, ". Connection : ",␣
↪detail.connectionName())
      elif detail.jobType() == ctk.ctkDICOMJobResponseSet.RetrieveStudy:
        patientID = detail.patientID()
        studyInstanceUID = detail.studyInstanceUID()
        print ("Retrieve studies failed for studyInstanceUID: ", studyInstanceUID, "␣
```

(continues on next page)

(continued from previous page)

```
↪(patientID: ", patientID, "). Connection : ", detail.connectionName())

  def onProgressDetails(self, details):
    for detail in details:
      if detail.jobType() == ctk.ctkDICOMJobResponseSet.QueryPatients:
        patientIDs = detail.queriedPatientIDs()
        for patientID in patientIDs:
          print ("Starting studies query for patient: ", patientID, ". Connection : ",␣
↪detail.connectionName())
          scheduler.queryStudies(patientID)
      elif detail.jobType() == ctk.ctkDICOMJobResponseSet.QueryStudies:
        studyInstanceUIDs = detail.queriedStudyInstanceUIDs()
        for studyInstanceUID in studyInstanceUIDs:
          patientItem = slicer.dicomDatabase.patientForStudy(studyInstanceUID)
          patientID = slicer.dicomDatabase.fieldForPatient("PatientID", patientItem)
          print ("Starting studies retrieve for studyInstanceUID: ", studyInstanceUID, "␣
↪(patientID: ",patientID, "). Connection : ", detail.connectionName())
          scheduler.retrieveStudy(patientID, studyInstanceUID)


# Add a new DICOM server
server = ctk.ctkDICOMServer()
server.connectionName = "test"
server.callingAETitle = "SLICER"
server.calledAETitle = "ANYAE"
server.host = "dicomserver.co.uk"
server.port = 104
server.retrieveProtocol = ctk.ctkDICOMServer.CGET

scheduler = ctk.ctkDICOMScheduler()
scheduler.setDicomDatabase(slicer.dicomDatabase)
scheduler.addServer(server)

receiver = Receiver(scheduler)

# Set the filters for the query
nDays = 325
endDate = qt.QDate().currentDate()
startDate = endDate.addDays(-nDays)
parameters = {
  "ID": "PAT020",
  #"Name": "Name",
  #"Study": "Study description",
  #"Series": "Series description",
  #"Modalities": ["CT", "MR"],
  "StartDate": startDate.toString("yyyyMMdd"),
  "EndDate": endDate.toString("yyyyMMdd")
}

receiver.startQueryRetrieve(parameters)
```

Send data to a PACS using classic DIMSE DICOM networking

```
from DICOMLib import DICOMSender
sender = DICOMSender(
  files=['path/to/0.dcm','path/to/1.dcm'],
  address='dicomserver.co.uk:9999'
  protocol="DIMSE",
  delayed=True
)
sender.send()
```

Send data to a PACS using DICOMweb networking

```
from DICOMLib import DICOMSender
sender = DICOMSender(
  files=['path/to/0.dcm','path/to/1.dcm'],
  address='dicomserver.co.uk:9999'
  protocol="DICOMweb",
  auth=DICOMUtils.getGlobalDICOMAuth(),
  delayed=True
)
sender.send()
```

Convert RT structure set to labelmap NRRD files

SlicerRT batch processing to batch convert RT structure sets to labelmap NRRD files.

Run a DCMTK Command Line Tool

The example below runs the DCMTK `img2dcm` tool to convert a PNG input image to an output DICOM file on disk. `img2dcm` runs in a separate process and Slicer waits until it completes before continuing.

See DCMTK documentation for descriptions of other DCMTK command line application tools.

```
from DICOMLib import DICOMCommand
command = DICOMCommand('img2dcm',['image.png','output.dcm'])
stdout = command.start() # run synchronously, block until img2dcm returns
```

Additional Notes

See the DICOMLib scripted module for additional DICOM utilities.

12.8.12 Markups

Save markups to file

Any markup node can be saved as a *markups json file*:

```
markupsNode = slicer.util.getNode('F')
slicer.util.saveNode(markupsNode, "/path/to/MyMarkups.mrk.json")
```

Generally the markups json file format is recommended for saving all properties of a markups node, but for exporting only control point information (name, position, and basic state) a *control points table can be exported in standard csv file format*:

```
slicer.modules.markups.logic().ExportControlPointsToCSV(markupsNode, "/path/to/
↪MyControlPoints.csv")
```

Load markups from file

Any markup node can be loaded from a *markups json file*:

```
markupsNode = slicer.util.loadMarkups("/path/to/MyMarkups.mrk.json")
```

Control points can be loaded from *control points table csv file*:

```
markupsNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsCurveNode")
slicer.modules.markups.logic().ImportControlPointsFromCSV(markupsNode, "/path/to/
↪MyControlPoints.csv")
```

Load markups point list from file

Markups point list can be loaded from legacy *fcsv file format*. Note that this file format is no longer recommended, as it is not a standard csv file format and can only store a small fraction of information that is in a markups node.

```
slicer.util.loadMarkups("/path/to/list/F.fcsv")
```

Adding control points Programmatically

Markups control points can be added to the currently active point list from the python console by using the following module logic command:

```
slicer.modules.markups.logic().AddControlPoint()
```

The command with no arguments will place a new control point at the origin. You can also pass it an initial location:

```
slicer.modules.markups.logic().AddControlPoint(1.0, -2.0, 3.3)
```

How to draw a curve using control points stored in a numpy array

```
# Create random numpy array to use as input
import numpy as np
pointPositions = np.random.uniform(-50,50,size=[15,3])

# Create curve from numpy array
curveNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsCurveNode")
slicer.util.updateMarkupsControlPointsFromArray(curveNode, pointPositions)
```

Add a button to module GUI to activate control point placement

This code snippet creates a toggle button, which activates control point placement when pressed (and deactivates when released).

The qSlicerMarkupsPlaceWidget widget can automatically activate placement of multiple points and can show buttons for deleting points, changing colors, lock, and hide points.

```
w=slicer.qSlicerMarkupsPlaceWidget()
w.setMRMLScene(slicer.mrmlScene)
markupsNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsCurveNode")
w.setCurrentNode(slicer.mrmlScene.GetNodeByID(markupsNode.GetID()))
# Hide all buttons and only show place button
w.buttonsVisible=False
w.placeButton().show()
w.show()
```

Adding control points via mouse clicks

You can also set the mouse mode into Markups control point placement by calling:

```
placeModePersistence = 1
slicer.modules.markups.logic().StartPlaceMode(placeModePersistence)
```

A lower level way to do this is via the selection and interaction nodes:

```
selectionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLSelectionNodeSingleton")
selectionNode.SetReferenceActivePlaceNodeClassName("vtkMRMLMarkupsFiducialNode")
interactionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLInteractionNodeSingleton")
placeModePersistence = 1
interactionNode.SetPlaceModePersistence(placeModePersistence)
# mode 1 is Place, can also be accessed via slicer.vtkMRMLInteractionNode().Place
interactionNode.SetCurrentInteractionMode(1)
```

To switch back to view transform once you're done placing control points:

```
interactionNode = slicer.mrmlScene.GetNodeByID("vtkMRMLInteractionNodeSingleton")
interactionNode.SwitchToViewTransformMode()
# also turn off place mode persistence if required
interactionNode.SetPlaceModePersistence(0)
```

Access to markups point list Properties

Each vtkMRMLMarkupsFiducialNode has a vector of control points in it which can be accessed from python:

```
pointListNode = getNode("vtkMRMLMarkupsFiducialNode1")
n = pointListNode.AddControlPoint([4.0, 5.5, -6.0])
pointListNode.SetNthControlPointLabel(n, "new label")
# each control point is given a unique id which can be accessed from the superclass level
id1 = pointListNode.GetNthControlPointID(n)
# manually set the position
pointListNode.SetNthControlPointPosition(n, 6.0, 7.0, 8.0)
# set the label
pointListNode.SetNthControlPointLabel(n, "New label")
# set the selected flag, only selected = 1 control points will be passed to CLIs
pointListNode.SetNthControlPointSelected(n, 1)
# set the visibility flag
pointListNode.SetNthControlPointVisibility(n, 0)
```

You can loop over the control points in a list and get the coordinates:

```
pointListNode = slicer.util.getNode("F")
numControlPoints = pointListNode.GetNumberOfControlPoints()
for i in range(numControlPoints):
  ras = vtk.vtkVector3d(0,0,0)
  pointListNode.GetNthControlPointPosition(i,ras)
  # the world position is the RAS position with any transform matrices applied
  world = [0.0, 0.0, 0.0]
  pointListNode.GetNthControlPointPositionWorld(i,world)
  print(i,": RAS =",ras,", world =",world)
```

You can also look at the sample code in the Endoscopy module to see how python is used to access control points from a scripted module.

Define/edit a circular region of interest in a slice viewer

Drop two markups control points on a slice view and copy-paste the code below into the Python console. After this, as you move the control points you'll see a circle following the markups.

```
# Update the sphere from the control points
def UpdateSphere(param1, param2):
  """Update the sphere from the control points
  """
  import math
  pointListNode = slicer.util.getNode("F")
  centerPointCoord = [0.0, 0.0, 0.0]
  pointListNode.GetNthControlPointPosition(0,centerPointCoord)
  circumferencePointCoord = [0.0, 0.0, 0.0]
  pointListNode.GetNthControlPointPosition(1,circumferencePointCoord)
  sphere.SetCenter(centerPointCoord)
  radius=math.sqrt((centerPointCoord[0]-circumferencePointCoord[0])**2+(centerPointCoord[1]-circumferencePointCoord[1])**2+(centerPointCoord[2]-circumferencePointCoord[2])**2)
  sphere.SetRadius(radius)
  sphere.SetPhiResolution(30)
  sphere.SetThetaResolution(30)
  sphere.Update()

# Get point list node from scene
pointListNode = slicer.util.getNode("F")
sphere = vtk.vtkSphereSource()
UpdateSphere(0,0)

# Create model node and add to scene
modelsLogic = slicer.modules.models.logic()
model = modelsLogic.AddModel(sphere.GetOutput())
model.GetDisplayNode().SetSliceIntersectionVisibility(True)
model.GetDisplayNode().SetSliceIntersectionThickness(3)
model.GetDisplayNode().SetColor(1,1,0)

# Call UpdateSphere whenever the control points are changed
pointListNode.AddObserver(slicer.vtkMRMLMarkupsNode.PointModifiedEvent, UpdateSphere, 2)
```

Specify a sphere by multiple control points

Drop multiple markups control points at the boundary of the spherical object and and copy-paste the code below into the Python console to get best-fit sphere. A minimum of 4 control points are required. Tt is recommended to place the control points far away from each other for the most accurate fit.

```
# Get markup node from scene
pointListNode = slicer.util.getNode("F")

from scipy.optimize import least_squares
import numpy

def fit_sphere_least_squares(x_values, y_values, z_values, initial_parameters, bounds=((-
↪numpy.inf, -numpy.inf, -numpy.inf, -numpy.inf),(numpy.inf, numpy.inf, numpy.inf, numpy.
↪inf))):
  """
  Source: https://github.com/thompson318/scikit-surgery-sphere-fitting/blob/master/
↪sksurgeryspherefitting/algorithms/sphere_fitting.py
  Uses scipy's least squares optimisor to fit a sphere to a set
  of 3D Points
  :return: x: an array containing the four fitted parameters
  :return: ier: int An integer flag. If it is equal to 1, 2, 3 or 4, the
          solution was found.
  :param: (x,y,z) three arrays of equal length containing the x, y, and z
          coordinates.
  :param: an array containing four initial values (centre, and radius)
  """
  return least_squares(_calculate_residual_sphere, initial_parameters, bounds=bounds,␣
↪method="trf", jac="3-point", args=(x_values, y_values, z_values))

def _calculate_residual_sphere(parameters, x_values, y_values, z_values):
  """
  Source: https://github.com/thompson318/scikit-surgery-sphere-fitting/blob/master/
↪sksurgeryspherefitting/algorithms/sphere_fitting.py
  Calculates the residual error for an x,y,z coordinates, fitted
  to a sphere with centre and radius defined by the parameters tuple
  :return: The residual error
  :param: A tuple of the parameters to be optimised, should contain [x_centre, y_centre,␣
↪z_centre, radius]
  :param: arrays containing the x,y, and z coordinates.
  """
  #extract the parameters
  x_centre, y_centre, z_centre, radius = parameters
  #use numpy's sqrt function here, which works by element on arrays
  distance_from_centre = numpy.sqrt((x_values - x_centre)**2 + (y_values - y_centre)**2␣
↪+ (z_values - z_centre)**2)
  return distance_from_centre - radius

# Fit a sphere to the markups fidicual points
markupsPositions = slicer.util.arrayFromMarkupsControlPoints(pointListNode)
import numpy as np
# initial guess
center0 = np.mean(markupsPositions, 0)
radius0 = np.linalg.norm(np.amin(markupsPositions,0)-np.amax(markupsPositions,0))/2.0
```

(continues on next page)

(continued from previous page)

```
fittingResult = fit_sphere_least_squares(markupsPositions[:,0], markupsPositions[:,1],␣
↪markupsPositions[:,2], [center0[0], center0[1], center0[2], radius0])
[centerX, centerY, centerZ, radius] = fittingResult["x"]

# Create a sphere using the fitted parameters
sphere = vtk.vtkSphereSource()
sphere.SetPhiResolution(30)
sphere.SetThetaResolution(30)
sphere.SetCenter(centerX, centerY, centerZ)
sphere.SetRadius(radius)
sphere.Update()

# Add the sphere to the scene
modelsLogic = slicer.modules.models.logic()
model = modelsLogic.AddModel(sphere.GetOutput())
model.GetDisplayNode().SetSliceIntersectionVisibility(True)
model.GetDisplayNode().SetSliceIntersectionThickness(3)
model.GetDisplayNode().SetColor(1,1,0)
```

Fit markups ROI to volume

This code snippet creates a new markups ROI and fits it to a volume node.

```
volumeNode = getNode('MRHead')

# Create a new ROI that will be fit to volumeNode
roiNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsROINode")

cropVolumeParameters = slicer.mrmlScene.AddNewNodeByClass(
↪"vtkMRMLCropVolumeParametersNode")
cropVolumeParameters.SetInputVolumeNodeID(volumeNode.GetID())
cropVolumeParameters.SetROINodeID(roiNode.GetID())
slicer.modules.cropvolume.logic().SnapROIToVoxelGrid(cropVolumeParameters)  # optional␣
↪(rotates the ROI to match the volume axis directions)
slicer.modules.cropvolume.logic().FitROIToInputVolume(cropVolumeParameters)
slicer.mrmlScene.RemoveNode(cropVolumeParameters)
```

Fit markups plane to model

This code snippet fits a plane a model node named `InputModel` and creates a new markups plane node to display this best fit plane.

```
inputModel = getNode('InputModel')

# Compute best fit plane
center = [0.0, 0.0, 0.0]
normal = [0.0, 0.0, 1.0]
vtk.vtkPlane.ComputeBestFittingPlane(inputModel.GetPolyData().GetPoints(), center,␣
↪normal)

# Display best fit plane as a markups plane
planeNode = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLMarkupsPlaneNode')
```

(continues on next page)

(continued from previous page)

```
planeNode.SetCenter(center)
planeNode.SetNormal(normal)
```

Measure angle between two markup planes

Measure angle between two markup plane nodes. Whenever any of the plane nodes are moved, the updated angle is printed on the console.

```
planeNodeNames = ["P", "P_1"]

# Print angles between slice nodes
def ShowAngle(unused1=None, unused2=None):
  planeNormalVectors = []
  for planeNodeName in planeNodeNames:
    planeNode = slicer.util.getFirstNodeByClassByName("vtkMRMLMarkupsPlaneNode",␣
↪planeNodeName)
    planeNormalVector = [0.0, 0.0, 0.0]
    planeNode.GetNormalWorld(planeNormalVector)
    planeNormalVectors.append(planeNormalVector)
  angleRad = vtk.vtkMath.AngleBetweenVectors(planeNormalVectors[0],␣
↪planeNormalVectors[1])
  angleDeg = vtk.vtkMath.DegreesFromRadians(angleRad)
  print("Angle between planes {0} and {1} = {2:0.3f}".format(planeNodeNames[0],␣
↪planeNodeNames[1], angleDeg))

# Observe plane node changes
for planeNodeName in planeNodeNames:
  planeNode = slicer.util.getFirstNodeByClassByName("vtkMRMLMarkupsPlaneNode",␣
↪planeNodeName)
  planeNode.AddObserver(slicer.vtkMRMLMarkupsPlaneNode.PointModifiedEvent, ShowAngle)

# Print current angle
ShowAngle()
```

Measure angle between two markup lines

Measure angle between two markup line nodes that are already added to the scene and have the names L and L_1. Whenever either line is moved, the updated angle is printed on the console. This is for illustration only, for standard angle measurements angle markups can be used.

```
lineNodeNames = ["L", "L_1"]

# Print angles between slice nodes
def ShowAngle(unused1=None, unused2=None):
  import numpy as np
  lineDirectionVectors = []
  for lineNodeName in lineNodeNames:
    lineNode = slicer.util.getFirstNodeByClassByName("vtkMRMLMarkupsLineNode",␣
↪lineNodeName)
    lineStartPos = np.zeros(3)
    lineEndPos = np.zeros(3)
    lineNode.GetNthControlPointPositionWorld(0, lineStartPos)
```

(continues on next page)

(continued from previous page)

```
    lineNode.GetNthControlPointPositionWorld(1, lineEndPos)
    lineDirectionVector = (lineEndPos-lineStartPos)/np.linalg.norm(lineEndPos-
↪lineStartPos)
    lineDirectionVectors.append(lineDirectionVector)
  angleRad = vtk.vtkMath.AngleBetweenVectors(lineDirectionVectors[0],␣
↪lineDirectionVectors[1])
  angleDeg = vtk.vtkMath.DegreesFromRadians(angleRad)
  print("Angle between lines {0} and {1} = {2:0.3f}".format(lineNodeNames[0],␣
↪lineNodeNames[1], angleDeg))

# Observe line node changes
for lineNodeName in lineNodeNames:
  lineNode = slicer.util.getFirstNodeByClassByName("vtkMRMLMarkupsLineNode",␣
↪lineNodeName)
  lineNode.AddObserver(slicer.vtkMRMLMarkupsLineNode.PointModifiedEvent, ShowAngle)

# Print current angle
ShowAngle()
```

Project a line to a plane

Create a new line (`projectedLineNode`) by projecting a line (`lineNode`) to a plane (`planeNode`).

Each control point is projected by computing coordinates in the plane coordinate system, zeroing the z coordinate (distance from plane) then transforming back the coordinates to the world coordinate system.

Transformation require homogeneous coordinates (1.0 appended to the 3D position), therefore 1.0 is added to the position after getting from the line and the 1.0 is removed when the computed point is added to the output line.

```
lineNode = getNode('L')
planeNode = getNode('P')

# Create new node for storing the projected line node
projectedLineNode = slicer.mrmlScene.AddNewNodeByClass(lineNode.GetClassName(), lineNode.
↪GetName()+" projected")

# Get transforms
planeToWorld = vtk.vtkMatrix4x4()
planeNode.GetObjectToWorldMatrix(planeToWorld)
worldToPlane = vtk.vtkMatrix4x4()
vtk.vtkMatrix4x4.Invert(planeToWorld, worldToPlane)

# Project each point
for pointIndex in range(2):
    point_World = [*lineNode.GetNthControlPointPositionWorld(pointIndex), 1.0]
    point_Plane = worldToPlane.MultiplyPoint(point_World)
    projectedPoint_Plane = [point_Plane[0], point_Plane[1], 0.0, 1.0]
    projectedPoint_World = planeToWorld.MultiplyPoint(projectedPoint_Plane)
    projectedLineNode.AddControlPoint(projectedPoint_World[0:3])
```

Measure distances of points from a plane

Place a plane (P) and add points to point list (F) using Markups module in a view and then run the following code snippet to compute distances of the points from the plane.

```
pointListNode = getNode('F')
planeNode = getNode('P')

# Get transformation that gets point coordinates relative to the plane
planeToWorld = vtk.vtkMatrix4x4()
planeNode.GetObjectToWorldMatrix(planeToWorld)
worldToPlane = vtk.vtkMatrix4x4()
vtk.vtkMatrix4x4.Invert(planeToWorld, worldToPlane)

for pointIndex in range(pointListNode.GetNumberOfControlPoints()):
    # Get point position in world coordinate system
    point_World = [*pointListNode.GetNthControlPointPositionWorld(pointIndex), 1.0]
    # Get point position in the plane coordinate system
    point_Plane = worldToPlane.MultiplyPoint(point_World)
    # Third axis in the plane coordinate system is the plane normal direction, therefore␣
↪the third coordinate
    # value is the distance from the plane
    distanceFromPlane = abs(point_Plane[2])
    print(f"Distance of point {pointListNode.GetNthControlPointLabel(pointIndex)} from␣
↪plane: {distanceFromPlane:.2f}")
```

Measure distances of points from a line

Draw a markups line (L) and drop markups point list (F) in a view and then run the following code snippet to compute distances of the points from the line.

```
pointListNode = getNode("F")
lineNode = getNode("L")

# Get point list control point positions and line endpoints as numpy arrays
points = slicer.util.arrayFromMarkupsControlPoints(pointListNode)
line = slicer.util.arrayFromMarkupsControlPoints(lineNode)
# Compute distance of control points from the line
from numpy import cross
from numpy.linalg import norm
for i, point in enumerate(points):
    d = norm(cross(line[1]-line[0],point-line[0])/norm(line[1]-line[0]))
    print(f"Point {i}: Position = {point}. Distance from line = {d}.")
```

Set slice position and orientation from 3 markups control points

Drop 3 markups control points in the scene and copy-paste the code below into the Python console. After this, as you move the control points you'll see the red slice view position and orientation will be set to make it fit to the 3 points.

```
# Update plane from control points
def UpdateSlicePlane(param1=None, param2=None):
  # Get control point positions as numpy array
  import numpy as np
  nOfControlPoints = pointListNode.GetNumberOfControlPoints()
```

(continues on next page)

(continued from previous page)

```
  if nOfControlPoints < 3:
    return  # not enough control points
  points = np.zeros([3,nOfControlPoints])
  for i in range(0, nOfControlPoints):
    pointListNode.GetNthControlPointPosition(i, points[:,i])
  # Compute plane position and normal
  planePosition = points.mean(axis=1)
  planeNormal = np.cross(points[:,1] - points[:,0], points[:,2] - points[:,0])
  planeX = points[:,1] - points[:,0]
  sliceNode.SetSliceToRASByNTP(planeNormal[0], planeNormal[1], planeNormal[2],
    planeX[0], planeX[1], planeX[2],
    planePosition[0], planePosition[1], planePosition[2], 0)

# Get point list node from scene
sliceNode = slicer.app.layoutManager().sliceWidget("Red").mrmlSliceNode()
pointListNode = slicer.util.getNode("F")

# Update slice plane manually
UpdateSlicePlane()

# Update slice plane automatically whenever points are changed
pointListObservation = [pointListNode, pointListNode.AddObserver(slicer.
↪vtkMRMLMarkupsNode.PointModifiedEvent, UpdateSlicePlane, 2)]
```

To stop automatic updates, run this:

```
pointListObservation[0].RemoveObserver(pointListObservation[1])
```

Switching to markups control point placement mode

To activate control point placement mode for a point list, both interaction mode has to be set and a point list node has to be selected:

```
interactionNode = slicer.app.applicationLogic().GetInteractionNode()
selectionNode = slicer.app.applicationLogic().GetSelectionNode()
selectionNode.SetReferenceActivePlaceNodeClassName("vtkMRMLMarkupsFiducialNode")
pointListNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsFiducialNode")
selectionNode.SetActivePlaceNodeID(pointListNode.GetID())
interactionNode.SetCurrentInteractionMode(interactionNode.Place)
```

Alternatively, *qSlicerMarkupsPlaceWidget* widget can be used to initiate markup placement:

```
# Temporary markups point list node
pointListNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsFiducialNode")

def placementModeChanged(active):
  print("Placement: " +("active" if active else "inactive"))
  # You can inspect what is in the markups node here, delete the temporary markup point␣
↪list node, etc.

# Create and set up widget that contains a single "place control point" button. The␣
↪widget can be placed in the module GUI.
```

(continues on next page)

(continued from previous page)

```
placeWidget = slicer.qSlicerMarkupsPlaceWidget()
placeWidget.setMRMLScene(slicer.mrmlScene)
placeWidget.setCurrentNode(pointListNode)
placeWidget.buttonsVisible=False
placeWidget.placeButton().show()
placeWidget.connect("activeMarkupsFiducialPlaceModeChanged(bool)", placementModeChanged)
placeWidget.show()
```

Change markup point list display properties

Display properties are stored in display node(s) associated with the point list node.

```
pointListNode = getNode("F")
pointListDisplayNode = pointListNode.GetDisplayNode()
pointListDisplayNode.SetVisibility(False) # Hide all points
pointListDisplayNode.SetVisibility(True) # Show all points
pointListDisplayNode.SetSelectedColor(1,1,0) # Set color to yellow
pointListDisplayNode.SetViewNodeIDs(["vtkMRMLSliceNodeRed", "vtkMRMLViewNode1"]) # Only␣
↪show in red slice view and first 3D view
```

Get a notification if a markup control point position is modified

Event management of Slicer-4.11 version is still subject to change. The example below shows how control point manipulation can be observed now.

```
def onMarkupChanged(caller,event):
  markupsNode = caller
  sliceView = markupsNode.GetAttribute(slicer.vtkMRMLMarkupsDisplayNode.
↪GetMovingInSliceViewAttributeName())
  movingMarkupIndex = markupsNode.GetDisplayNode().GetActiveControlPoint()
  if movingMarkupIndex >= 0:
    pos = [0,0,0]
    markupsNode.GetNthControlPointPosition(movingMarkupIndex, pos)
    isPreview = markupsNode.GetNthControlPointPositionStatus(movingMarkupIndex) ==␣
↪slicer.vtkMRMLMarkupsNode.PositionPreview
    if isPreview:
      logging.info("Point {0} is previewed at {1} in slice view {2}".
↪format(movingMarkupIndex, pos, sliceView))
    else:
      logging.info("Point {0} was moved {1} in slice view {2}".format(movingMarkupIndex,␣
↪pos, sliceView))
  else:
    logging.info("Points modified: slice view = {0}".format(sliceView))

def onMarkupStartInteraction(caller, event):
  markupsNode = caller
  sliceView = markupsNode.GetAttribute(slicer.vtkMRMLMarkupsDisplayNode.
↪GetMovingInSliceViewAttributeName())
  movingMarkupIndex = markupsNode.GetDisplayNode().GetActiveControlPoint()
  logging.info("Start interaction: point ID = {0}, slice view = {1}".
↪format(movingMarkupIndex, sliceView))
```

(continues on next page)

(continued from previous page)

```
def onMarkupEndInteraction(caller, event):
  markupsNode = caller
  sliceView = markupsNode.GetAttribute(slicer.vtkMRMLMarkupsDisplayNode.
→GetMovingInSliceViewAttributeName())
  movingMarkupIndex = markupsNode.GetDisplayNode().GetActiveControlPoint()
  logging.info("End interaction: point ID = {0}, slice view = {1}".
→format(movingMarkupIndex, sliceView))

pointListNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsFiducialNode")
pointListNode.AddControlPoint([0,0,0])
pointListNode.AddObserver(slicer.vtkMRMLMarkupsNode.PointModifiedEvent, onMarkupChanged)
pointListNode.AddObserver(slicer.vtkMRMLMarkupsNode.PointStartInteractionEvent,␣
→onMarkupStartInteraction)
pointListNode.AddObserver(slicer.vtkMRMLMarkupsNode.PointEndInteractionEvent,␣
→onMarkupEndInteraction)
```

Write markup control point positions to JSON file

```
pointListNode = getNode("F")
outputFileName = "c:/tmp/test.json"

# Get markup positions
data = []
for fidIndex in range(pointListNode.GetNumberOfControlPoints()):
  coords=[0,0,0]
  pointListNode.GetNthControlPointPosition(fidIndex,coords)
  data.append({"label": pointListNode.GetNthControlPointLabel(), "position": coords})

import json
with open(outputFileName, "w") as outfile:
  json.dump(data, outfile)
```

Write markup ROI to JSON file

```
roiNode = getNode("R")
outputFileName = "c:/tmp/test.json"

# Get ROI data
center = [0,0,0]
size = [0,0,0]
roiNode.GetCenterWorld(center)
roiNode.GetSizeWorld(size)
data = {"center": center, "size": size}

# Write to json file
import json
with open(outputFileName, "w") as outfile:
  json.dump(data, outfile)
```

Create markup plane JSON file - outside Slicer

This code can be used in any Python environment to create a markup json file containing a simple plane.

If you specify a plane by a single point (`"planeType": "pointNormal"`) then you only need to specify:

- `plane position`: position of the one and only control point
- `plane orientation`: `baseToNode` matrix. Note that a plane in mathematical sense is specified by a position and a normal, but if you want to display a plane then you need to specify its rotation around that normal. `baseToNode` contains the two axis of the plane and the plane normal. If you don't care about the rotation then you can choose an arbitrary orientation and use cross product with the normal vector to make it orthogonal to the normal vector. Translation component of baseToNode matrix is ignored (because position is set by the control point), so it can be set to `0.0, 0.0, 0.0`.
- `plane bounds`: specifies how far the plane extends in 4 directions (-x, +x, -y, +y) from the plane position

```
outputFileName = "path/to/MyPlane.mrk.json"
center = [3.6764886856933536, -52.2593679682938, 41.715845278879044]
normal = [-0.9552783445937983, 0.2957081066696218, 0.0]
plane_bounds = [-50.0, 50.0, -50.0, 50.0]

import numpy as np
import json

normal /= np.linalg.norm(normal)

# Choose an arbitrary vector direction (x) that is not parallel to the normal
axis1 = np.array([1, 0, 0])
if np.linalg.norm(np.cross(normal, axis1)) < 0.1:
  # Almost parallel to the x axis, use another direction (y)
  axis1 = np.array([0, 1, 0])

# Calculate a third axis of the plane coordinate system (orthogonal to the other two)
axis2 = np.cross(normal, axis1)
axis2 /= np.linalg.norm(axis2)

# Get axis1 that is orthogonal to axis2 and normal
axis1 = np.cross(axis2, normal)
axis1 /= np.linalg.norm(axis1)

# Construct the rotation-translation matrix baseToNode
base_to_node_matrix = np.eye(4)
base_to_node_matrix[0:3, 0] = axis1
base_to_node_matrix[0:3, 1] = axis2
base_to_node_matrix[0:3, 2] = normal

data = {
    "@schema": "https://raw.githubusercontent.com/slicer/slicer/master/Modules/Loadable/
↪Markups/Resources/Schema/markups-schema-v1.0.3.json#",
    "markups": [
        {
            "type": "Plane",
            "coordinateSystem": "LPS",
            "coordinateUnits": "mm",
            "planeType": "pointNormal",
```

(continues on next page)

(continued from previous page)

```
            "sizeMode": "auto",
            "baseToNode": list(base_to_node_matrix.reshape(16)),
            "planeBounds": plane_bounds,
            "controlPoints": [{ "id": "1", "position": center }]
        }
    ]
}

with open(outputFileName, "w") as outfile:
  json.dump(data, outfile)
```

Fit slice plane to markup control points

```
sliceNode = slicer.mrmlScene.GetNodeByID("vtkMRMLSliceNodeRed")
pointListNode = slicer.mrmlScene.GetFirstNodeByName("F")
# Get markup point positions as numpy arrays
import numpy as np
p1 = np.zeros(3)
p2 = np.zeros(3)
p3 = np.zeros(3)
pointListNode.GetNthControlPointPosition(0, p1)
pointListNode.GetNthControlPointPosition(1, p2)
pointListNode.GetNthControlPointPosition(2, p3)
# Get plane axis directions
n = np.cross(p2-p1, p2-p3) # plane normal direction
n = n/np.linalg.norm(n)
t = np.cross([0.0, 0.0, 1], n) # plane transverse direction
t = t/np.linalg.norm(t)
# Set slice plane orientation and position
sliceNode.SetSliceToRASByNTP(n[0], n[1], n[2], t[0], t[1], t[2], p1[0], p1[1], p1[2], 0)
```

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 must be used to change markups node color.

Display list of control points in my module's GUI

The qSlicerSimpleMarkupsWidget can be integrated into module widgets to display list of markups control points and initiate placement. An example of this use is in Gel Dosimetry module.

Pre-populate the scene with measurements

This code snippet creates a set of predefined line markups (named A, B, C, D) in the scene when the user hits Ctrl+N. How to use this:

1. Customize the code (replace A, B, C, D with your measurement names) and copy-paste the code into the Python console. This has to be done only once after Slicer is started. Add it to *.slicerrc.py file* so that it persists even if Slicer is restarted.
2. Load the data set that has to be measured
3. Hit Ctrl+N to create all the measurements
4. Go to Markups module to see the list of measurements

5. For each measurement: select it in the data tree, click on the place button on the toolbar then click in slice or 3D views

```
def createMeasurements():
  for nodeName in ['A', 'B', 'C', 'D']:
    lineNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsLineNode", nodeName)
    lineNode.CreateDefaultDisplayNodes()
    dn = lineNode.GetDisplayNode()
    # Use crosshair glyph to allow more accurate point placement
    dn.SetGlyphTypeFromString("CrossDot2D")
    # Hide measurement result while markup up
    lineNode.GetMeasurement('length').SetEnabled(False)

shortcut1 = qt.QShortcut(slicer.util.mainWindow())
shortcut1.setKey(qt.QKeySequence("Ctrl+n"))
shortcut1.connect( 'activated()', createMeasurements)
```

Copy all measurements in the scene to Excel

This code snippet creates a set of predefined line markups (named A, B, C, D) in the scene when the user hits Ctrl+N. How to use this:

1. Copy-paste the code into the Python console. This has to be done only once after Slicer is started. Add it to *.slicerrc.py file* so that it persists even if Slicer is restarted.
2. Load the data set that has to be measured and place line markups (you can use the "Pre-populate the scene with measurements" script above to help with this)
3. Hit Ctrl+M to copy all line measurents to the clipboard
4. Switch to Excel and hit Ctrl+V to paste the results there
5. Save the scene, just in case later you need to review your measurements

```
def copyLineMeasurementsToClipboard():
  measurements = []
  # Collect all line measurements from the scene
  lineNodes = getNodesByClass('vtkMRMLMarkupsLineNode')
  for lineNode in lineNodes:
    if lineNode.GetNumberOfDefinedControlPoints() < 2:
      # incomplete line, skip it
      continue
    # Get node filename that the length was measured on
    try:
      volumeNode = slicer.mrmlScene.GetNodeByID(lineNode.
↪GetNthControlPointAssociatedNodeID(0))
      imagePath = volumeNode.GetStorageNode().GetFileName()
    except:
      imagePath = '(unknown)'
    # Get line node n
    measurementName = lineNode.GetName()
    # Get length measurement
    lineNode.GetMeasurement('length').SetEnabled(True)
    length = str(lineNode.GetMeasurement('length').GetValue())
    # Add fields to results
    measurements.append('\t'.join([imagePath, measurementName, length]))
```

(continues on next page)

(continued from previous page)

```
  # Copy all measurements to clipboard (to be pasted into Excel)
  outputText = "\n".join(measurements) + "\n"
  slicer.app.clipboard().setText(outputText)
  slicer.util.delayDisplay(f"Copied {len(measurements)} length measurements to the␣
↪clipboard.")

shortcut2 = qt.QShortcut(slicer.util.mainWindow())
shortcut2.setKey(qt.QKeySequence("Ctrl+m"))
shortcut2.connect( 'activated()', copyLineMeasurementsToClipboard)
```

To copy all measurement results to a file instead of copying it to the clipboard, replace `slicer.app.clipboard...` line by these lines:

```
with open("c:/tmp/results.csv", "a") as f:
  f.write(outputText)
```

Use markups json files in Python - outside Slicer

The examples below show how to use markups json files outside Slicer, in any Python environment.

To access content of a json file it can be either read as a json document or directly into a pandas dataframe using a single command.

Get a table of control point labels and positions

Get table from the first markups node in the file:

```
import pandas as pd
controlPointsTable = pd.DataFrame.from_dict(pd.read_json(input_json_filename)['markups
↪'][0]['controlPoints'])
```

Result:

```
>>> controlPointsTable
  label                                         position
0   F-1  [-53.388409961685824, -73.33572796934868, 0.0]
1   F-2     [49.8682950191571, -88.58955938697324, 0.0]
2   F-3   [-25.22749042145594, 59.255268199233726, 0.0]
```

Access position of control points positions in separate x, y, z columns

```
controlPointsTable[['x','y','z']] = pd.DataFrame(controlPointsTable['position'].to_
↪list())
del controlPointsTable['position']
```

Write control points to a csv file

```
controlPointsTable.to_csv(output_csv_filename)
```

Resulting csv file:

```
,label,x,y,z
0,F-1,-53.388409961685824,-73.33572796934868,0.0
1,F-2,49.8682950191571,-88.58955938697324,0.0
2,F-3,-25.22749042145594,59.255268199233726,0.0
```

Assign custom actions to markups

Custom actions can be assigned to markups, which can be triggered by any interaction event (mouse or keyboard action). The actions can be detected by adding observers to the markup node's display node.

```
# This example adds an action to the default double-click action on a markup
# and defines two new custom actions. It is done for all existing markups in the first␣
↪3D view.
#
# How to use:
# 1. Create markups nodes.
# 2. Run the script below.
# 3. Double-click on the markup -> this triggers toggleLabelVisibilty.
# 4. Hover the mouse over a markup then pressing `q` and `w` keys -> this triggers␣
↪shrinkControlPoints and growControlPoints.

threeDViewNode = slicer.mrmlScene.GetNodeByID("vtkMRMLViewNode1")
appLogic = slicer.app.applicationLogic()
markupsDisplayableManager = appLogic.GetViewDisplayableManagerByClassName(threeDViewNode,
↪ "vtkMRMLMarkupsDisplayableManager")

def shrinkControlPoints(caller, eventId):
  markupsDisplayNode = caller
  markupsDisplayNode.SetGlyphScale(markupsDisplayNode.GetGlyphScale()/1.1)

def growControlPoints(caller, eventId):
  markupsDisplayNode = caller
  markupsDisplayNode.SetGlyphScale(markupsDisplayNode.GetGlyphScale()*1.1)

def toggleLabelVisibility(caller, eventId):
  markupsDisplayNode = caller
  markupsDisplayNode.SetPointLabelsVisibility(not markupsDisplayNode.
↪GetPointLabelsVisibility())

observations = []  # store the observations so that later can be removed
markupsDisplayNodes = slicer.util.getNodesByClass("vtkMRMLMarkupsDisplayNode")
for markupsDisplayNode in markupsDisplayNodes:
  # Assign keyboard shortcut to trigger custom actions
  markupsWidget = markupsDisplayableManager.GetWidget(markupsDisplayNode)
  # Left double-click interaction event is translated to markupsWidget.WidgetEventAction␣
↪by default,
  # therefore we don't need to add an event translation for that. We just add two␣
↪keyboard event translation for two custom actions
  markupsWidget.SetKeyboardEventTranslation(markupsWidget.WidgetStateOnWidget, vtk.
↪vtkEvent.NoModifier, '\0', 0, "q", markupsWidget.WidgetEventCustomAction1)
  markupsWidget.SetKeyboardEventTranslation(markupsWidget.WidgetStateOnWidget, vtk.
↪vtkEvent.NoModifier, '\0', 0, "w", markupsWidget.WidgetEventCustomAction2)
  # Add observer to custom actions
```

(continues on next page)

(continued from previous page)

```
  observations.append([markupsDisplayNode, markupsDisplayNode.
↪AddObserver(markupsDisplayNode.ActionEvent, toggleLabelVisibility)])
  observations.append([markupsDisplayNode, markupsDisplayNode.
↪AddObserver(markupsDisplayNode.CustomActionEvent1, shrinkControlPoints)])
  observations.append([markupsDisplayNode, markupsDisplayNode.
↪AddObserver(markupsDisplayNode.CustomActionEvent2, growControlPoints)])

# Remove observations when custom actions are not needed anymore by uncommenting these␣
↪lines:
for observedNode, observation in observations:
  observedNode.RemoveObserver(observation)
```

12.8.13 Models

Show a simple surface mesh as a model node

This example shows how to display a simple surface mesh (a box, created by a VTK source filter) as a model node.

```
# Create and set up polydata source
box = vtk.vtkCubeSource()
box.SetXLength(30)
box.SetYLength(20)
box.SetZLength(15)
box.SetCenter(10,20,5)

# Create a model node that displays output of the source
boxNode = slicer.modules.models.logic().AddModel(box.GetOutputPort())

# Adjust display properties
boxNode.GetDisplayNode().SetColor(1,0,0)
boxNode.GetDisplayNode().SetOpacity(0.8)
```

Measure distance of points from surface

This example computes closest distance of points (markups point list `F`) from a surface (model node `mymodel`) and writes results into a table.

```
pointListNode = getNode("F")
modelNode = getNode("mymodel")

# Transform model polydata to world coordinate system
if modelNode.GetParentTransformNode():
  transformModelToWorld = vtk.vtkGeneralTransform()
  slicer.vtkMRMLTransformNode.GetTransformBetweenNodes(modelNode.
↪GetParentTransformNode(), None, transformModelToWorld)
  polyTransformToWorld = vtk.vtkTransformPolyDataFilter()
  polyTransformToWorld.SetTransform(transformModelToWorld)
  polyTransformToWorld.SetInputData(modelNode.GetPolyData())
  polyTransformToWorld.Update()
  surface_World = polyTransformToWorld.GetOutput()
else:
  surface_World = modelNode.GetPolyData()
```

(continues on next page)

(continued from previous page)

```
# Create arrays to store results
indexCol = vtk.vtkIntArray()
indexCol.SetName("Index")
labelCol = vtk.vtkStringArray()
labelCol.SetName("Name")
distanceCol = vtk.vtkDoubleArray()
distanceCol.SetName("Distance")

distanceFilter = vtk.vtkImplicitPolyDataDistance()
distanceFilter.SetInput(surface_World)
nOfControlPoints = pointListNode.GetNumberOfControlPoints()
for i in range(0, nOfControlPoints):
  point_World = [0,0,0]
  pointListNode.GetNthControlPointPositionWorld(i, point_World)
  closestPointOnSurface_World = [0,0,0]
  closestPointDistance = distanceFilter.EvaluateFunctionAndGetClosestPoint(point_World,␣
↪closestPointOnSurface_World)
  indexCol.InsertNextValue(i)
  labelCol.InsertNextValue(pointListNode.GetNthControlPointLabel(i))
  distanceCol.InsertNextValue(closestPointDistance)

# Create a table from result arrays
resultTableNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLTableNode", "Points from␣
↪surface distance")
resultTableNode.AddColumn(indexCol)
resultTableNode.AddColumn(labelCol)
resultTableNode.AddColumn(distanceCol)

# Show table in view layout
slicer.app.layoutManager().setLayout(slicer.vtkMRMLLayoutNode.
↪SlicerLayoutFourUpTableView)
slicer.app.applicationLogic().GetSelectionNode().
↪SetReferenceActiveTableID(resultTableNode.GetID())
slicer.app.applicationLogic().PropagateTableSelection()
```

Add a texture mapped plane to the scene as a model

```
# Create model node
planeSource = vtk.vtkPlaneSource()
planeSource.SetOrigin(-50.0, -50.0, 0.0)
planeSource.SetPoint1(50.0, -50.0, 0.0)
planeSource.SetPoint2(-50.0, 50.0, 0.0)
model = slicer.modules.models.logic().AddModel(planeSource.GetOutputPort())

# Tune display properties
modelDisplay = model.GetDisplayNode()
modelDisplay.SetColor(1,1,0) # yellow
modelDisplay.SetBackfaceCulling(0)

# Add texture (just use image of an ellipsoid)
```

(continues on next page)

(continued from previous page)

```
e = vtk.vtkImageEllipsoidSource()
modelDisplay.SetTextureImageDataConnection(e.GetOutputPort())
```

Note

Model textures are not exposed in the GUI and are not saved in the scene.

Get scalar values at surface of a model

The following script allows getting selected scalar value at a selected position of a model. Position can be selected by moving the mouse while holding down Shift key.

```
modelNode = getNode("sphere")
modelPointValues = modelNode.GetPolyData().GetPointData().GetArray("Normals")
pointListNode = slicer.mrmlScene.GetFirstNodeByName("F")

if not pointListNode:
  pointListNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsFiducialNode","F")

pointsLocator = vtk.vtkPointLocator() # could try using vtk.vtkStaticPointLocator() if␣
↪need to optimize
pointsLocator.SetDataSet(modelNode.GetPolyData())
pointsLocator.BuildLocator()

def onMouseMoved(observer,eventid):
  ras=[0,0,0]
  crosshairNode.GetCursorPositionRAS(ras)
  closestPointId = pointsLocator.FindClosestPoint(ras)
  ras = modelNode.GetPolyData().GetPoint(closestPointId)
  closestPointValue = modelPointValues.GetTuple(closestPointId)
  if pointListNode.GetNumberOfControlPoints() == 0:
    pointListNode.AddControlPoint(ras)
  else:
    pointListNode.SetNthControlPointPosition(0,*ras)
  print(f"RAS={ras}  value={closestPointValue}")

crosshairNode=slicer.util.getNode("Crosshair")
observationId = crosshairNode.AddObserver(slicer.vtkMRMLCrosshairNode.
↪CursorPositionModifiedEvent, onMouseMoved)

# To stop printing of values run this:
# crosshairNode.RemoveObserver(observationId)
```

Apply VTK filter on a model node

```
modelNode = getNode("tip")

# Compute curvature
curv = vtk.vtkCurvatures()
```

(continues on next page)

(continued from previous page)

```
curv.SetInputData(modelNode.GetPolyData())
modelNode.SetPolyDataConnection(curv.GetOutputPort())

# Set up coloring by Curvature
modelNode.GetDisplayNode().SetActiveScalar("Gauss_Curvature", vtk.vtkAssignAttribute.
↪POINT_DATA)
modelNode.GetDisplayNode().SetAndObserveColorNodeID("Viridis")
modelNode.GetDisplayNode().SetScalarVisibility(True)
```

Select cells of a model using markups point list

The following script selects cells of a model node that are closest to positions of markups control points.

```
# Get input nodes
modelNode = slicer.util.getNode("Segment_1") # select cells in this model
pointListNode = slicer.util.getNode("F") # points will be selected at positions␣
↪specified by this markups point list node

# Create scalar array that will store selection state
cellScalars = modelNode.GetMesh().GetCellData()
selectionArray = cellScalars.GetArray("selection")
if not selectionArray:
  selectionArray = vtk.vtkIntArray()
  selectionArray.SetName("selection")
  selectionArray.SetNumberOfValues(modelNode.GetMesh().GetNumberOfCells())
  selectionArray.Fill(0)
  cellScalars.AddArray(selectionArray)

# Set up coloring by selection array
modelNode.GetDisplayNode().SetActiveScalar("selection", vtk.vtkAssignAttribute.CELL_DATA)
modelNode.GetDisplayNode().SetAndObserveColorNodeID("vtkMRMLColorTableNodeWarm1")
modelNode.GetDisplayNode().SetScalarVisibility(True)

# Initialize cell locator
cell = vtk.vtkCellLocator()
cell.SetDataSet(modelNode.GetMesh())
cell.BuildLocator()

def onPointsModified(observer=None, eventid=None):
  global pointListNode, selectionArray
  selectionArray.Fill(0) # set all cells to non-selected by default
  markupPoints = slicer.util.arrayFromMarkupsControlPoints(pointListNode)
  closestPoint = [0.0, 0.0, 0.0]
  cellObj = vtk.vtkGenericCell()
  cellId = vtk.mutable(0)
  subId = vtk.mutable(0)
  dist2 = vtk.mutable(0.0)
  for markupPoint in markupPoints:
    cell.FindClosestPoint(markupPoint, closestPoint, cellObj, cellId, subId, dist2)
    closestCell = cellId.get()
    if closestCell >=0:
      selectionArray.SetValue(closestCell, 100) # set selected cell's scalar value to non-
```

(continues on next page)

(continued from previous page)

```
↪zero
  selectionArray.Modified()

# Initial update
onPointsModified()
# Automatic update each time when a markup point is modified
pointListNodeObserverTag = pointListNode.AddObserver(slicer.vtkMRMLMarkupsNode.
↪PointModifiedEvent, onPointsModified)

# To stop updating selection, run this:
# pointListNode.RemoveObserver(pointListNodeObserverTag)
```

Export entire scene as glTF

glTF is a modern and very efficient file format for surface meshes, which is supported by many web viewers, such as:

- https://3dviewer.net/ (requires a single zip file that contains all the exported files)
- https://gltf-viewer.donmccurdy.com/ (the exported folder can be drag-and-dropped to the webpage)

SlicerOpenAnatomy extension provides rich export of models and segmentations (preserving names, hierarchy, etc.), but for a basic export operation this code snippet can be used:

```
exporter = vtk.vtkGLTFExporter()
exporter.SetRenderWindow(slicer.app.layoutManager().threeDWidget(0).threeDView().
↪renderWindow())
exporter.SetFileName("c:/tmp/newfolder/mymodel.gltf")
exporter.Write()
```

Export entire scene as VRML

Save all surface meshes displayed in the scene (models, markups, etc). Solid colors and coloring by scalar is preserved. Textures are not supported. VRML is a very old general-purpose scene file format, which is still supported by some software.

```
exporter = vtk.vtkVRMLExporter()
exporter.SetRenderWindow(slicer.app.layoutManager().threeDWidget(0).threeDView().
↪renderWindow())
exporter.SetFileName("C:/tmp/something.wrl")
exporter.Write()
```

Export model to Blender, including color by scalar

```
   modelNode = getNode("Model")
   plyFilePath = "c:/tmp/model.ply"

   modelDisplayNode = modelNode.GetDisplayNode()
   triangles = vtk.vtkTriangleFilter()
   triangles.SetInputConnection(modelDisplayNode.GetOutputPolyDataConnection())

   plyWriter = vtk.vtkPLYWriter()
   plyWriter.SetInputConnection(triangles.GetOutputPort())
   lut = vtk.vtkLookupTable()
```

(continues on next page)

(continued from previous page)

```
    lut.DeepCopy(modelDisplayNode.GetColorNode().GetLookupTable())
    lut.SetRange(modelDisplayNode.GetScalarRange())
    plyWriter.SetLookupTable(lut)
    plyWriter.SetArrayName(modelDisplayNode.GetActiveScalarName())

    plyWriter.SetFileName(plyFilePath)
    plyWriter.Write()
```

Show comparison view of all model files a folder

```
# Inputs
modelDir = "c:/some/folder/containing/models"
modelFileExt = "stl"
numberOfColumns = 4

import math
import os
modelFiles = list(f for f in os.listdir(modelDir) if f.endswith("." + modelFileExt))

# Create a custom layout
numberOfRows = int(math.ceil(len(modelFiles)/numberOfColumns))
customLayoutId=567  # we pick a random id that is not used by others
slicer.app.setRenderPaused(True)
customLayout = '<layout type="vertical">'
viewIndex = 0
for rowIndex in range(numberOfRows):
  customLayout += '<item><layout type="horizontal">'
  for colIndex in range(numberOfColumns):
    name = os.path.basename(modelFiles[viewIndex]) if viewIndex < len(modelFiles) else
↪"compare " + str(viewIndex)
    customLayout += '<item><view class="vtkMRMLViewNode" singletontag="'+name
    customLayout += '"><property name="viewlabel" action="default">'+name+'</property></
↪view></item>'
    viewIndex += 1
  customLayout += '</layout></item>'

customLayout += '</layout>'
if not slicer.app.layoutManager().layoutLogic().GetLayoutNode().
↪SetLayoutDescription(customLayoutId, customLayout):
    slicer.app.layoutManager().layoutLogic().GetLayoutNode().
↪AddLayoutDescription(customLayoutId, customLayout)

slicer.app.layoutManager().setLayout(customLayoutId)

# Load and show each model in a view
for modelIndex, modelFile in enumerate(modelFiles):
  # Show only one model in each view
  name = os.path.basename(modelFile)
  viewNode = slicer.mrmlScene.GetSingletonNode(name, "vtkMRMLViewNode")
  viewNode.LinkedControlOn()
  modelNode = slicer.util.loadModel(modelDir + "/" + modelFile)
```

(continues on next page)

(continued from previous page)

```
  modelNode.GetDisplayNode().AddViewNodeID(viewNode.GetID())

slicer.app.setRenderPaused(False)
```

Rasterize a model and save it to a series of image files

This example shows how to generate a stack of image files from an STL file:

```
inputModelFile = "/some/input/folder/SomeShape.stl"
outputDir = "/some/output/folder"
outputVolumeLabelValue = 100
outputVolumeSpacingMm = [0.5, 0.5, 0.5]
outputVolumeMarginMm = [10.0, 10.0, 10.0]

# Read model
inputModel = slicer.util.loadModel(inputModelFile)

# Determine output volume geometry and create a corresponding reference volume
import math
import numpy as np
bounds = np.zeros(6)
inputModel.GetBounds(bounds)
imageData = vtk.vtkImageData()
imageSize = [ int((bounds[axis*2+1]-bounds[axis*2]+outputVolumeMarginMm[axis]*2.0)/
↪outputVolumeSpacingMm[axis]) for axis in range(3) ]
imageOrigin = [ bounds[axis*2]-outputVolumeMarginMm[axis] for axis in range(3) ]
imageData.SetDimensions(imageSize)
imageData.AllocateScalars(vtk.VTK_UNSIGNED_CHAR, 1)
imageData.GetPointData().GetScalars().Fill(0)
referenceVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode")
referenceVolumeNode.SetOrigin(imageOrigin)
referenceVolumeNode.SetSpacing(outputVolumeSpacingMm)
referenceVolumeNode.SetAndObserveImageData(imageData)
referenceVolumeNode.CreateDefaultDisplayNodes()

# Convert model to labelmap
seg = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLSegmentationNode")
seg.SetReferenceImageGeometryParameterFromVolumeNode(referenceVolumeNode)
slicer.modules.segmentations.logic().ImportModelToSegmentationNode(inputModel, seg)
seg.CreateBinaryLabelmapRepresentation()
outputLabelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode
↪")
slicer.modules.segmentations.logic().ExportVisibleSegmentsToLabelmapNode(seg,␣
↪outputLabelmapVolumeNode, referenceVolumeNode)
outputLabelmapVolumeArray = (slicer.util.arrayFromVolume(outputLabelmapVolumeNode) *␣
↪outputVolumeLabelValue).astype("int8")

# Install dependencies
try:
  import imageio
except ModuleNotFoundError:
  slicer.packaging.pip_install("imageio")
```

(continues on next page)

(continued from previous page)

```
  import imageio

# Write labelmap volume to series of TIFF files
for i in range(len(outputLabelmapVolumeArray)):
  imageio.imwrite(f"{outputDir}/image_{i:03}.tiff", outputLabelmapVolumeArray[i])
```

> **Tip**
>
> To learn how to use *slicer.packaging.pip_install()* within a Slicer module, refer to the *Install a Python package* example in the Script Repository.

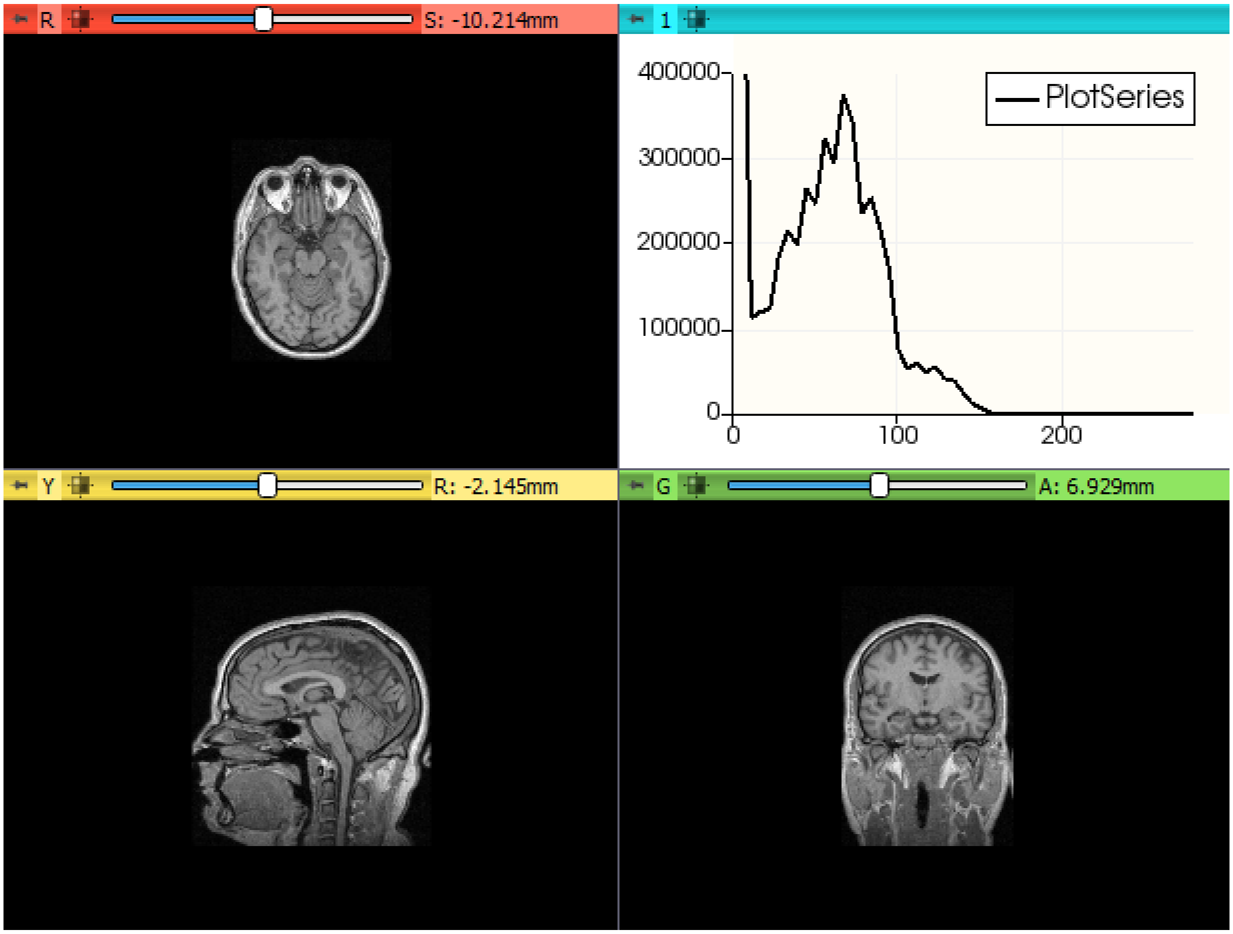
12.8.14 Plots

Slicer plots displayed in view layout

Create histogram plot of a volume and show it embedded in the view layout. More information: https://www.slicer.org/wiki/Documentation/Nightly/Developers/Plots

Using *slicer.util.plot()* utility function

```
# Get a volume from SampleData and compute its histogram
import SampleData
import numpy as np
volumeNode = SampleData.SampleDataLogic().downloadMRHead()
histogram = np.histogram(slicer.util.arrayFromVolume(volumeNode), bins=50)

chartNode = slicer.util.plot(histogram, xColumnIndex = 1)
chartNode.SetYAxisRangeAuto(False)
chartNode.SetYAxisRange(0, 4e5)
```



Using MRML classes only

```
# Get a volume from SampleData
import SampleData
volumeNode = SampleData.SampleDataLogic().downloadMRHead()

# Compute histogram values
import numpy as np
histogram = np.histogram(slicer.util.arrayFromVolume(volumeNode), bins=50)

# Save results to a new table node
tableNode=slicer.mrmlScene.AddNewNodeByClass("vtkMRMLTableNode")
slicer.util.updateTableFromArray(tableNode, histogram)
tableNode.GetTable().GetColumn(0).SetName("Count")
tableNode.GetTable().GetColumn(1).SetName("Intensity")

# Create plot
plotSeriesNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLPlotSeriesNode", volumeNode.
↪GetName() + " histogram")
plotSeriesNode.SetAndObserveTableNodeID(tableNode.GetID())
plotSeriesNode.SetXColumnName("Intensity")
plotSeriesNode.SetYColumnName("Count")
plotSeriesNode.SetPlotType(plotSeriesNode.PlotTypeScatterBar)
```

(continues on next page)

(continued from previous page)

```
plotSeriesNode.SetColor(0, 0.6, 1.0)

# Create chart and add plot
plotChartNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLPlotChartNode")
plotChartNode.AddAndObservePlotSeriesNodeID(plotSeriesNode.GetID())
plotChartNode.YAxisRangeAutoOff()
plotChartNode.SetYAxisRange(0, 500000)

# Show plot in layout
slicer.modules.plots.logic().ShowChartInLayout(plotChartNode)
```

Save a plot as vector graphics (.svg)

```
plotView = slicer.app.layoutManager().plotWidget(0).plotView()
plotView.saveAsSVG("c:/tmp/test.svg")
```

Using matplotlib

Matplotlib may be used from within Slicer, but the default Tk backend locks up and crashes Slicer. However, Matplotlib may still be used through other backends. More details can be found on the MatPlotLib pages.

Non-interactive plot

```
try:
  import matplotlib
except ModuleNotFoundError:
  slicer.packaging.pip_install("matplotlib")
  import matplotlib

matplotlib.use("Agg")
from pylab import *

t1 = arange(0.0, 5.0, 0.1)
t2 = arange(0.0, 5.0, 0.02)
t3 = arange(0.0, 2.0, 0.01)

subplot(211)
plot(t1, cos(2*pi*t1)*exp(-t1), "bo", t2, cos(2*pi*t2)*exp(-t2), "k")
grid(True)
title("A tale of 2 subplots")
ylabel("Damped")

subplot(212)
plot(t3, cos(2*pi*t3), "r--")
grid(True)
xlabel("time (s)")
ylabel("Undamped")
savefig("MatplotlibExample.png")

# Static image view
pm = qt.QPixmap("MatplotlibExample.png")
```

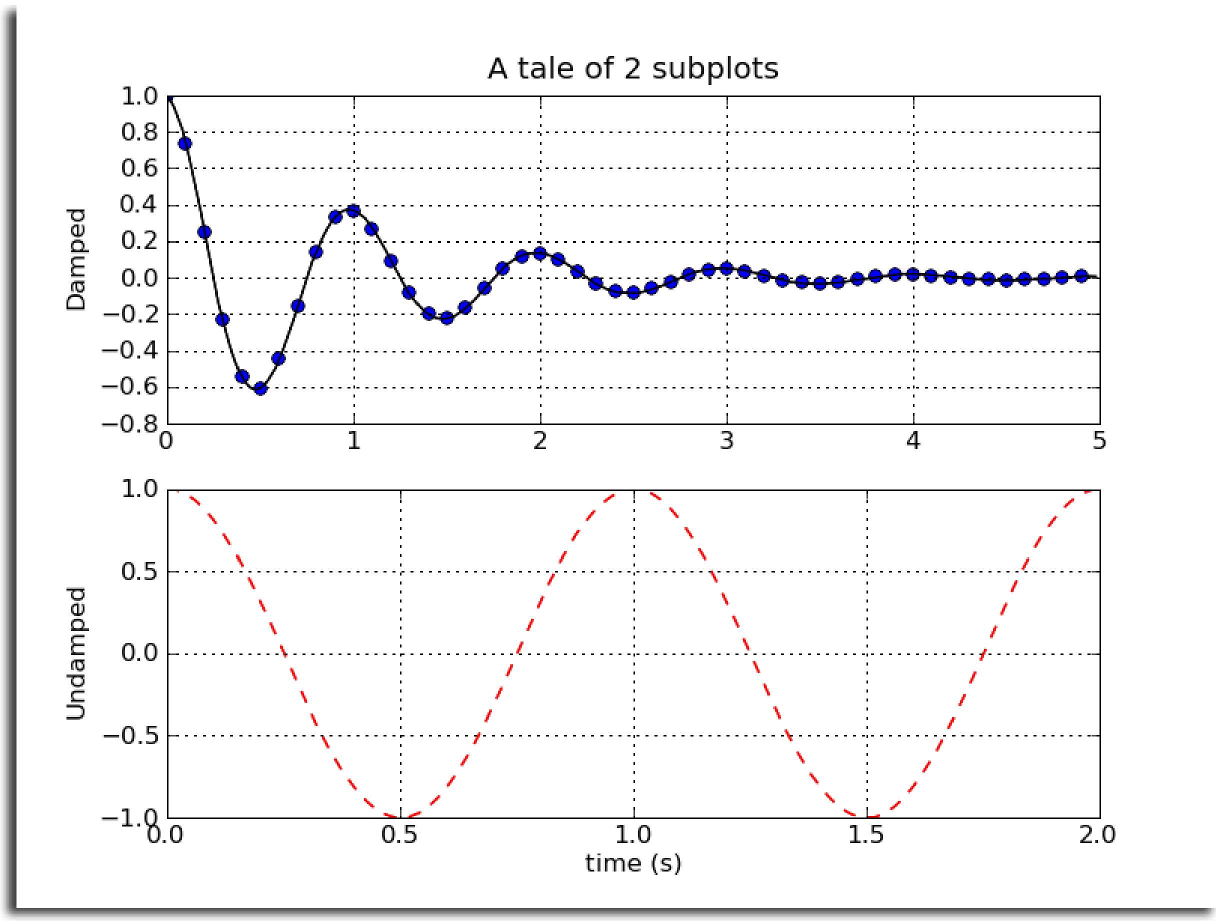
(continues on next page)

(continued from previous page)

```
imageWidget = qt.QLabel()
imageWidget.setPixmap(pm)
imageWidget.setScaledContents(True)
imageWidget.show()
```

Tip

To learn how to use *slicer.packaging.pip_install()* within a Slicer module, refer to the *Install a Python package* example in the Script Repository.



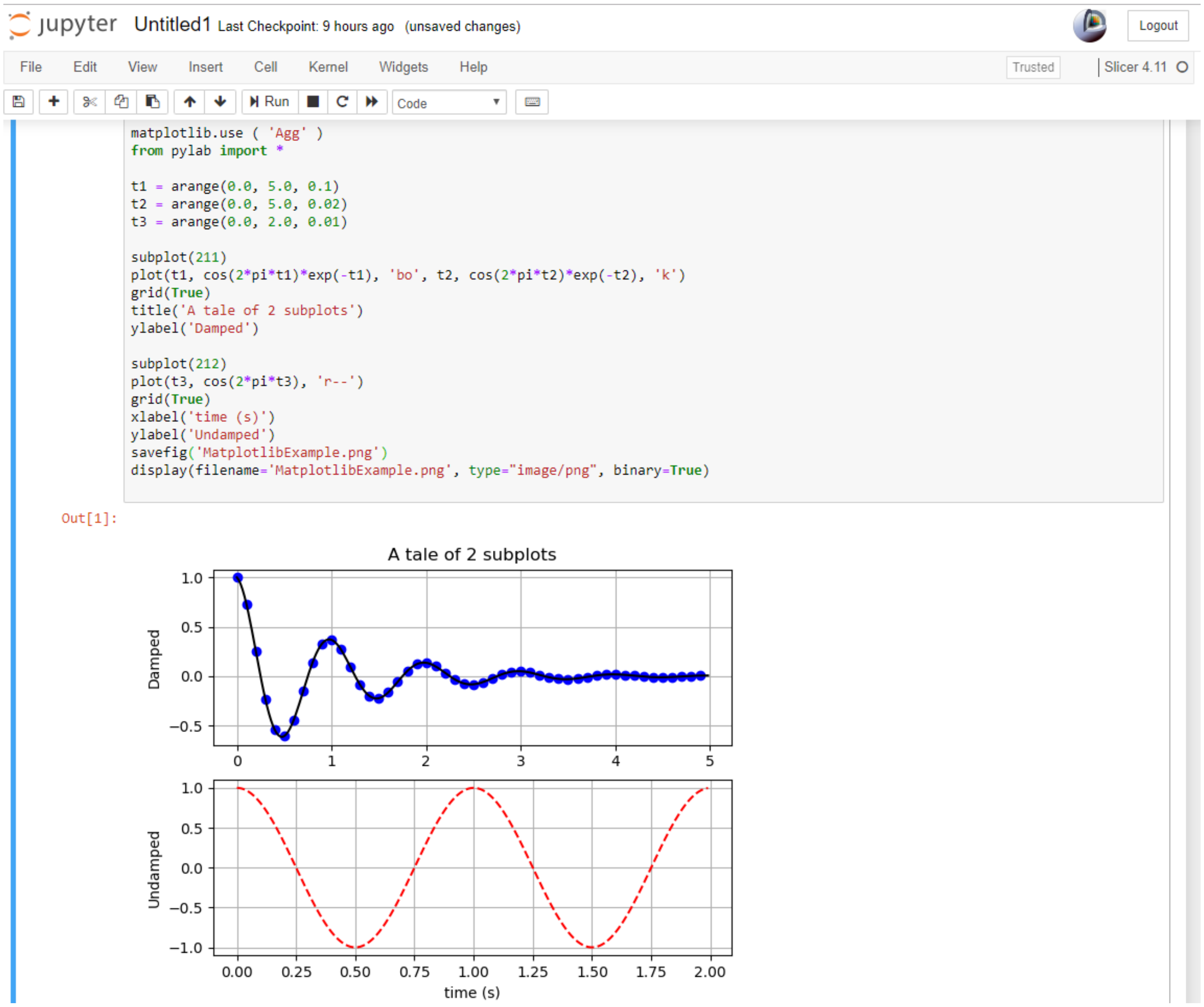
Plot in Slicer Jupyter notebook

```
import JupyterNotebooksLib as slicernb
try:
  import matplotlib
except ModuleNotFoundError:
  slicer.packaging.pip_install("matplotlib")
  import matplotlib
```

(continues on next page)

(continued from previous page)

```
matplotlib.use("Agg")

import matplotlib.pyplot as plt
import numpy as np

def f(t):
  s1 = np.cos(2*np.pi*t)
  e1 = np.exp(-t)
  return s1 * e1

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
t3 = np.arange(0.0, 2.0, 0.01)


fig, axs = plt.subplots(2, 1, constrained_layout=True)
axs[0].plot(t1, f(t1), "o", t2, f(t2), "-")
axs[0].set_title("subplot 1")
axs[0].set_xlabel("distance (m)")
axs[0].set_ylabel("Damped oscillation")
fig.suptitle("This is a somewhat long figure title", fontsize=16)

axs[1].plot(t3, np.cos(2*np.pi*t3), "--")
axs[1].set_xlabel("time (s)")
axs[1].set_title("subplot 2")
axs[1].set_ylabel("Undamped")

slicernb.MatplotlibDisplay(matplotlib.pyplot)
```



Interactive plot using wxWidgets GUI toolkit

```
try:
  import matplotlib
  import wx
except ModuleNotFoundError:
  slicer.packaging.pip_install("matplotlib wxPython")
  import matplotlib

# Get a volume from SampleData and compute its histogram
import SampleData
import numpy as np
volumeNode = SampleData.SampleDataLogic().downloadMRHead()
histogram = np.histogram(slicer.util.arrayFromVolume(volumeNode), bins=50)

# Set matplotlib to use WXAgg backend
import matplotlib
matplotlib.use("WXAgg")
```

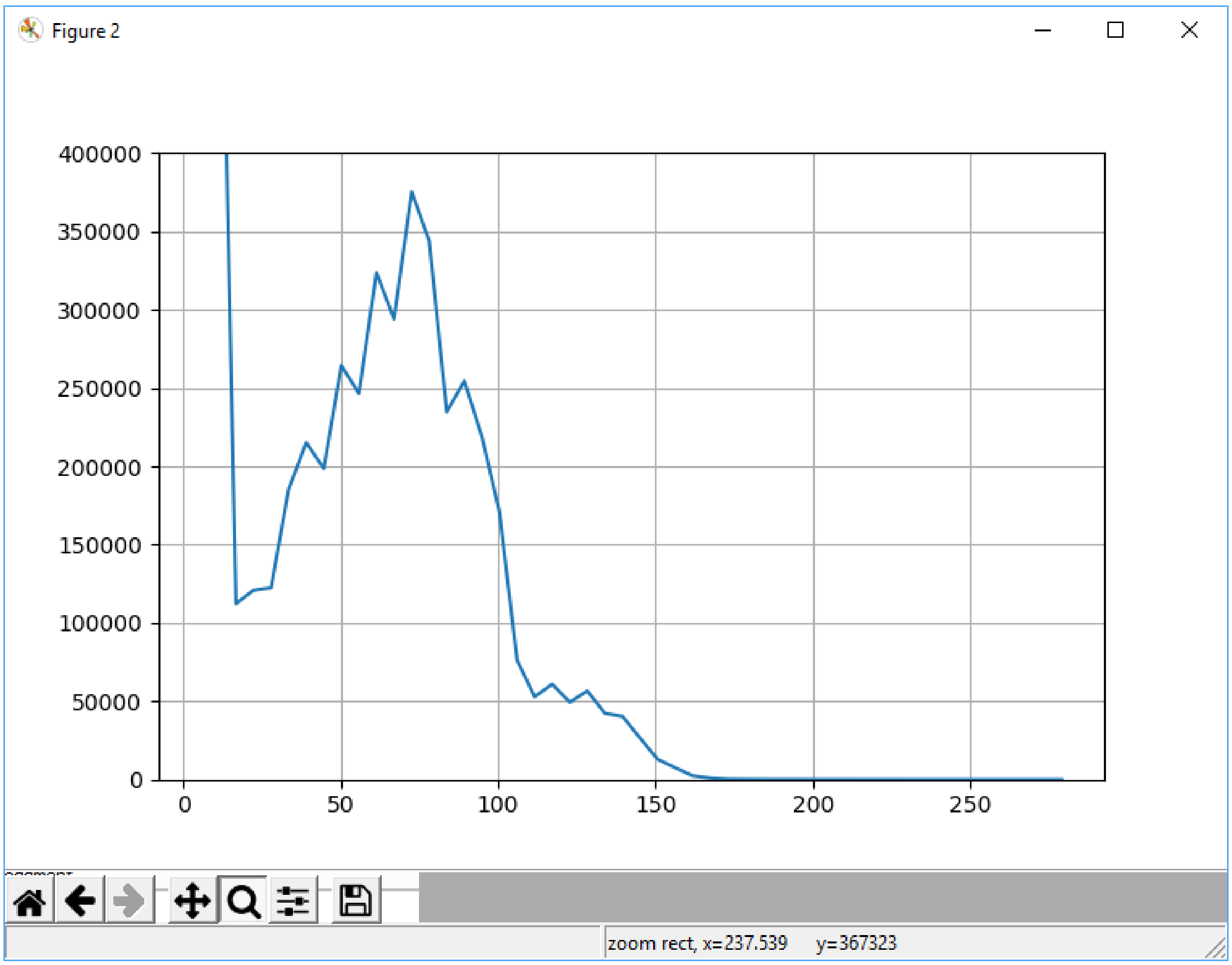
(continues on next page)

(continued from previous page)

```
# Show an interactive plot
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.plot(histogram[1][1:], histogram[0].astype(float))
ax.grid(True)
ax.set_ylim((0, 4e5))
plt.show(block=False)
```

Tip

To learn how to use *slicer.packaging.pip_install()* within a Slicer module, refer to the *Install a Python package* example in the Script Repository.



12.8.15 Screen Capture

Capture the full Slicer screen and save it into a file

```
img = qt.QPixmap.grabWidget(slicer.util.mainWindow()).toImage()
img.save("c:/tmp/test.png")
```

Capture all the views save it into a file

```
import ScreenCapture
cap = ScreenCapture.ScreenCaptureLogic()
cap.showViewControllers(False)
cap.captureImageFromView(None, "c:/tmp/test.png")
cap.showViewControllers(True)
```

Capture a single view

```
viewNodeID = "vtkMRMLViewNode1"
import ScreenCapture
cap = ScreenCapture.ScreenCaptureLogic()
view = cap.viewFromNode(slicer.mrmlScene.GetNodeByID(viewNodeID))
cap.captureImageFromView(view, "c:/tmp/test.png")
```

Common values for viewNodeID: vtkMRMLSliceNodeRed, vtkMRMLSliceNodeYellow, vtkMRMLSliceNodeGreen, vtkMRMLViewNode1, vtkMRMLViewNode2. The ScreenCapture module can also create video animations of rotating views, slice sweeps, etc.

Capture a slice view sweep into a series of PNG files

For example, Red slice view, 30 images, from position -125.0 to 75.0, into c:/tmp folder, with name image_00001.png, image_00002.png, ...

```
import ScreenCapture
ScreenCapture.ScreenCaptureLogic().captureSliceSweep(getNode("vtkMRMLSliceNodeRed"), -
↪125.0, 75.0, 30, "c:/tmp", "image_%05d.png")
```

Capture 3D view into PNG file with transparent background

```
# Set background to black (required for transparent background)
view = slicer.app.layoutManager().threeDWidget(0).threeDView()
view.mrmlViewNode().SetBackgroundColor(0,0,0)
view.mrmlViewNode().SetBackgroundColor2(0,0,0)
view.forceRender()
# Capture RGBA image
renderWindow = view.renderWindow()
renderWindow.SetAlphaBitPlanes(1)
wti = vtk.vtkWindowToImageFilter()
wti.SetInputBufferTypeToRGBA()
wti.SetInput(renderWindow)
writer = vtk.vtkPNGWriter()
writer.SetFileName("c:/tmp/screenshot.png")
writer.SetInputConnection(wti.GetOutputPort())
writer.Write()
```

Capture slice view into PNG file with white background

```
sliceViewName = "Red"
filename = "c:/tmp/screenshot.png"
```

(continues on next page)

(continued from previous page)

```
# Set view background to white
view = slicer.app.layoutManager().sliceWidget(sliceViewName).sliceView()
view.setBackgroundColor(qt.QColor.fromRgbF(1,1,1))
view.forceRender()

# Capture a screenshot
import ScreenCapture
cap = ScreenCapture.ScreenCaptureLogic()
cap.captureImageFromView(view, filename)
```

Save a series of images from a slice view

You can use ScreenCapture module to capture series of images. To do it programmatically, save the following into a file such as `/tmp/record.py` and then in the Slicer python console type `execfile("/tmp/record.py")`

```
layoutName = "Green"
imagePathPattern = "/tmp/image-%03d.png"
steps = 10

widget = slicer.app.layoutManager().sliceWidget(layoutName)
view = widget.sliceView()
logic = widget.sliceLogic()
bounds = [0,]*6
logic.GetSliceBounds(bounds)

for step in range(steps):
  offset = bounds[4] + step/(1.*steps) * (bounds[5]-bounds[4])
  logic.SetSliceOffset(offset)
  view.forceRender()
  image = qt.QPixmap.grabWidget(view).toImage()
  image.save(imagePathPattern % step)
```

12.8.16 Segmentations

Load a 3D image or model file as segmentation

```
# Load segmentation from .seg.nrrd file (includes segment names and colors)
slicer.util.loadSegmentation("c:/tmp/tmp/Segmentation.nrrd")

# Create segmentation from a NIFTI + color table file
colorNode = slicer.util.loadColorTable('c:/tmp/tmp/Segmentation-label_ColorTable.ctbl')
slicer.util.loadSegmentation("c:/tmp/tmp/Segmentation.nii", {'colorNodeID': colorNode.
↪GetID()})

# Create segmentation from a STL file
slicer.util.loadSegmentation("c:/tmp/Segment_1.stl")
```

Create a segmentation from a labelmap volume and display in 3D

```
labelmapVolumeNode = getNode("label")
seg = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLSegmentationNode")
slicer.modules.segmentations.logic().ImportLabelmapToSegmentationNode(labelmapVolumeNode,
↪ seg)
seg.CreateClosedSurfaceRepresentation()
slicer.mrmlScene.RemoveNode(labelmapVolumeNode)
```

The last line is optional. It removes the original labelmap volume so that the same information is not shown twice.

Create segmentation from a model node

```
# Create some model that will be added to a segmentation node
sphere = vtk.vtkSphereSource()
sphere.SetCenter(-6, 30, 28)
sphere.SetRadius(10)
modelNode = slicer.modules.models.logic().AddModel(sphere.GetOutputPort())

# Create segmentation
segmentationNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLSegmentationNode")
segmentationNode.CreateDefaultDisplayNodes() # only needed for display

# Import the model into the segmentation node
slicer.modules.segmentations.logic().ImportModelToSegmentationNode(modelNode,␣
↪segmentationNode)
```

Export labelmap node from segmentation node

Export labelmap matching reference geometry of the segmentation:

```
segmentationNode = getNode("Segmentation")
labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
slicer.modules.segmentations.logic().ExportAllSegmentsToLabelmapNode(segmentationNode,␣
↪labelmapVolumeNode, slicer.vtkSegmentation.EXTENT_REFERENCE_GEOMETRY)
```

Export smallest possible labelmap:

```
segmentationNode = getNode("Segmentation")
labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
slicer.modules.segmentations.logic().ExportAllSegmentsToLabelmapNode(segmentationNode,␣
↪labelmapVolumeNode)
```

Export labelmap that matches geometry of a chosen reference volume:

```
segmentationNode = getNode("Segmentation")
labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
slicer.modules.segmentations.logic().
↪ExportVisibleSegmentsToLabelmapNode(segmentationNode, labelmapVolumeNode,␣
↪referenceVolumeNode)
```

Export a selection of segments (identified by their names):

```
segmentNames = ["Prostate", "Urethra"]
segmentIds = vtk.vtkStringArray()
for segmentName in segmentNames:
  segmentId = segmentationNode.GetSegmentation().GetSegmentIdBySegmentName(segmentName)
  segmentIds.InsertNextValue(segmentId)
slicer.vtkSlicerSegmentationsModuleLogic.ExportSegmentsToLabelmapNode(segmentationNode,␣
↪segmentIds, labelmapVolumeNode, referenceVolumeNode)
```

Export to file by pressing Ctrl+Shift+S key:

```
outputPath = "c:/tmp"

def exportLabelmap():
  segmentationNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLSegmentationNode")
  referenceVolumeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")
  labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
  slicer.modules.segmentations.logic().
↪ExportVisibleSegmentsToLabelmapNode(segmentationNode, labelmapVolumeNode,␣
↪referenceVolumeNode)
  filepath = outputPath + "/" + referenceVolumeNode.GetName() + "-label.nrrd"
  slicer.util.saveNode(labelmapVolumeNode, filepath)
  slicer.mrmlScene.RemoveNode(labelmapVolumeNode.GetDisplayNode().GetColorNode())
  slicer.mrmlScene.RemoveNode(labelmapVolumeNode)
  slicer.util.delayDisplay("Segmentation saved to " + filepath)

shortcut = qt.QShortcut(slicer.util.mainWindow())
shortcut.setKey(qt.QKeySequence("Ctrl+Shift+s"))
shortcut.connect( "activated()", exportLabelmap)
```

Import/export labelmap node using custom label value mapping

While in segmentation nodes segments are identified by segment ID, name, or terminology; in labelmap nodes a segment can be identified only by its label value. Slicer can import a labelmap volume into segmentation, visualize/edit the segmentation, then export the segmentation into labelmap volume - preserving the label values in the output. This is achieved by using a color node during labelmap node import and export, which assigns a name for each label value. Segment corresponding to a label value is found by matching the color name to the segment name.

Create color table node

A color table node can be loaded from a *color table file* or created from scratch like this:

```
segment_names_to_labels = [("ribs", 10), ("right lung", 12), ("left lung", 6)]

colorTableNode = slicer.mrmlScene.CreateNodeByClass("vtkMRMLColorTableNode")
colorTableNode.SetTypeToUser()
colorTableNode.HideFromEditorsOff()  # make the color table selectable in the GUI␣
↪outside Colors module
slicer.mrmlScene.AddNode(colorTableNode); colorTableNode.UnRegister(None)
largestLabelValue = max([name_value[1] for name_value in segment_names_to_labels])
colorTableNode.SetNumberOfColors(largestLabelValue + 1)
import random
for segmentName, labelValue in segment_names_to_labels:
    r = random.uniform(0.0, 1.0)
```

(continues on next page)

(continued from previous page)

```
    g = random.uniform(0.0, 1.0)
    b = random.uniform(0.0, 1.0)
    a = 1.0
    success = colorTableNode.SetColor(labelValue, segmentName, r, g, b, a)
```

Export labelmap node from segmentation node using custom label value mapping

```
segmentationNode = getNode('Segmentation')  # source segmentation node
labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")  #␣
↪export to new labelmap volume
referenceVolumeNode = None # it could be set to the master volume
segmentIds = segmentationNode.GetSegmentation().GetSegmentIDs()  # export all segments
colorTableNode = ...  # created from scratch or loaded from file

slicer.modules.segmentations.logic().ExportSegmentsToLabelmapNode(segmentationNode,␣
↪segmentIds, labelmapVolumeNode, referenceVolumeNode, slicer.vtkSegmentation.EXTENT_
↪REFERENCE_GEOMETRY, colorTableNode)
```

Import labelmap node into segmentation node using custom label value mapping

```
labelmapVolumeNode = getNode('Volume-label')
segmentationNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLSegmentationNode")  #␣
↪import into new segmentation node
colorTableNode = ...  # created from scratch or loaded from file

labelmapVolumeNode.GetDisplayNode().SetAndObserveColorNodeID(colorTableNode.GetID())  #␣
↪just in case the custom color table has not been already associated with the labelmap␣
↪volume
slicer.modules.segmentations.logic().ImportLabelmapToSegmentationNode(labelmapVolumeNode,
↪ segmentationNode)
```

Export model nodes from segmentation node

```
segmentationNode = getNode("Segmentation")
shNode = slicer.mrmlScene.GetSubjectHierarchyNode()
exportFolderItemId = shNode.CreateFolderItem(shNode.GetSceneItemID(), "Segments")
slicer.modules.segmentations.logic().ExportAllSegmentsToModels(segmentationNode,␣
↪exportFolderItemId)
```

Create a hollow model from boundary of solid segment

In most cases, the most robust and flexible tool for creating empty shell models (e.g., vessel wall model from contrast agent segmentation) is the "Hollow" effect in Segment Editor module. However, for very thin shells, extrusion of the exported surface mesh representation may be just as robust and require less memory and computation time. In this case it may be a better approach to to export the segment to a mesh and extrude it along surface normal direction:

Example using Dynamic Modeler module (allows real-time update of parameters, using GUI in Dynamic Modeler module):

```
segmentationNode = getNode("Segmentation")
```

(continues on next page)

(continued from previous page)

```
# Export segments to models
shNode = slicer.mrmlScene.GetSubjectHierarchyNode()
exportFolderItemId = shNode.CreateFolderItem(shNode.GetSceneItemID(), "Segments")
slicer.modules.segmentations.logic().ExportAllSegmentsToModels(segmentationNode,␣
↪exportFolderItemId)
segmentModels = vtk.vtkCollection()
shNode.GetDataNodesInBranch(exportFolderItemId, segmentModels)
# Get exported model of first segment
modelNode = segmentModels.GetItemAsObject(0)

# Set up Hollow tool
hollowModeler = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLDynamicModelerNode")
hollowModeler.SetToolName("Hollow")
hollowModeler.SetNodeReferenceID("Hollow.InputModel", modelNode.GetID())
hollowedModelNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLModelNode")  # this node␣
↪will store the hollow model
hollowModeler.SetNodeReferenceID("Hollow.OutputModel", hollowedModelNode.GetID())
hollowModeler.SetAttribute("ShellThickness", "2.5")  # grow outside
hollowModeler.SetContinuousUpdate(True)  # auto-update output model if input parameters␣
↪are changed

# Hide inputs, show output
segmentationNode.GetDisplayNode().SetVisibility(False)
modelNode.GetDisplayNode().SetVisibility(False)
hollowedModelNode.GetDisplayNode().SetOpacity(0.5)
```

Example using VTK filters:

```
# Get closed surface representation of the segment
shellThickness = 3.0  # mm
segmentationNode = getNode("Segmentation")
segmentationNode.CreateClosedSurfaceRepresentation()
polyData = segmentationNode.GetClosedSurfaceInternalRepresentation("Segment_1")

# Create shell
extrude = vtk.vtkLinearExtrusionFilter()
extrude.SetInputData(polyData)
extrude.SetExtrusionTypeToNormalExtrusion()
extrude.SetScaleFactor(shellThickness)

# Compute consistent surface normals
triangle_filter = vtk.vtkTriangleFilter()
triangle_filter.SetInputConnection(extrude.GetOutputPort())
normals = vtk.vtkPolyDataNormals()
normals.SetInputConnection(triangle_filter.GetOutputPort())
normals.FlipNormalsOn()

# Save result into new model node
slicer.modules.models.logic().AddModel(normals.GetOutputPort())
```

Show a segmentation in 3D

Segmentation can only be shown in 3D if closed surface representation (or other 3D-displayable representation) is available. To create closed surface representation:

```
segmentation.CreateClosedSurfaceRepresentation()
```

Modify segmentation display options

```
segmentation = getNode('Segmentation')
segmentID = 'Segment_1'

displayNode = segmentation.GetDisplayNode()
displayNode.SetOpacity3D(0.4)  # Set overall opacity of the segmentation
displayNode.SetSegmentOpacity3D(segmentID, 0.2)  # Set opacity of a single segment

# Segment color is not just a display property, but it is stored in the segment itself␣
↪(and stored in the segmentation file)
segment = segmentation.GetSegmentation().GetSegment(segmentID)
segment.SetColor(1, 0, 0)  # red

# In very special cases (for example, when a segment's color only need to be changed in a␣
↪specific view)
# the segment color can be overridden in the display node.
# This is not recommended for general use.
displayNode.SetSegmentOverrideColor(segmentID, 0, 0, 1)  # blue
```

Get a representation of a segment

Access binary labelmap stored in a segmentation node (without exporting it to a volume node) - if it does not exist, it will return None:

```
image = slicer.vtkOrientedImageData()
segmentationNode.GetBinaryLabelmapRepresentation(segmentID, image)
```

Get closed surface, if it does not exist, it will return None:

```
outputPolyData = vtk.vtkPolyData()
segmentationNode.GetClosedSurfaceRepresentation(segmentID, outputPolyData)
```

Get binary labelmap representation. If it does not exist then it will be created for that single segment. Applies parent transforms by default (if not desired, another argument needs to be added to the end: false):

```
import vtkSegmentationCorePython as vtkSegmentationCore
outputOrientedImageData = vtkSegmentationCore.vtkOrientedImageData()
slicer.vtkSlicerSegmentationsModuleLogic.
↪GetSegmentBinaryLabelmapRepresentation(segmentationNode, segmentID,␣
↪outputOrientedImageData)
```

Same as above, for closed surface representation:

```
outputPolyData = vtk.vtkPolyData()
slicer.vtkSlicerSegmentationsModuleLogic.
↪GetSegmentClosedSurfaceRepresentation(segmentationNode, segmentID, outputPolyData)
```

Convert all segments using default path and conversion parameters

```
segmentationNode.CreateBinaryLabelmapRepresentation()
```

Convert all segments using custom path or conversion parameters

Change reference image geometry parameter based on an existing referenceImageData image:

```
referenceGeometry = slicer.vtkSegmentationConverter.
↪SerializeImageGeometry(referenceImageData)
segmentation.SetConversionParameter(slicer.vtkSegmentationConverter.
↪GetReferenceImageGeometryParameterName(), referenceGeometry)
```

Re-convert using a modified conversion parameter

Changing smoothing factor for closed surface generation:

```
import vtkSegmentationCorePython as vtkSegmentationCore
segmentation = getNode("Segmentation").GetSegmentation()

# Turn of surface smoothing
segmentation.SetConversionParameter("Smoothing factor","0.0")

# Recreate representation using modified parameters (and default conversion path)
segmentation.RemoveRepresentation(vtkSegmentationCore.vtkSegmentationConverter.
↪GetSegmentationClosedSurfaceRepresentationName())
segmentation.CreateRepresentation(vtkSegmentationCore.vtkSegmentationConverter.
↪GetSegmentationClosedSurfaceRepresentationName())
```

Create keyboard shortcut for toggling sphere brush for paint and erase effects

```
def toggleSphereBrush():
  segmentEditorWidget = slicer.modules.segmenteditor.widgetRepresentation().self().editor
  paintEffect = segmentEditorWidget.effectByName("Paint")
  isSphere = paintEffect.integerParameter("BrushSphere")
  # BrushSphere is "common" parameter (shared between paint and erase)
  paintEffect.setCommonParameter("BrushSphere", 0 if isSphere else 1)

shortcut = qt.QShortcut(slicer.util.mainWindow())
shortcut.setKey(qt.QKeySequence("s"))
shortcut.connect("activated()", toggleSphereBrush)
```

Create keyboard shortcut for toggling visibility of a set of segments

This script toggles visibility of "completed" segments if Ctrl-k keyboard shortcut is pressed:

```
slicer.segmentationNode = getNode('Segmentation')
slicer.toggledSegmentState="completed"  # it could be "inprogress", "completed", "flagged
↪"
slicer.visibility = True

def toggleSegmentVisibility():
    slicer.visibility = not slicer.visibility
```

(continues on next page)

(continued from previous page)

```
    segmentation = slicer.segmentationNode.GetSegmentation()
    for segmentIndex in range(segmentation.GetNumberOfSegments()):
        segmentId = segmentation.GetNthSegmentID(segmentIndex)
        segmentationStatus = vtk.mutable("")
        if not segmentation.GetSegment(segmentId).GetTag("Segmentation.Status",␣
↪segmentationStatus):
            continue
        if segmentationStatus != slicer.toggledSegmentState:
            continue
        slicer.segmentationNode.GetDisplayNode().SetSegmentVisibility(segmentId, slicer.
↪visibility)

shortcut = qt.QShortcut(slicer.util.mainWindow())
shortcut.setKey(qt.QKeySequence("Ctrl+k"))
shortcut.connect( "activated()", toggleSegmentVisibility)
```

Customize list of displayed Segment editor effects

Only show Paint and Erase effects:

```
segmentEditorWidget = slicer.modules.segmenteditor.widgetRepresentation().self().editor
segmentEditorWidget.setEffectNameOrder(["Paint", "Erase"])
segmentEditorWidget.unorderedEffectsVisible = False
```

Show list of all available effect names:

```
segmentEditorWidget = slicer.modules.segmenteditor.widgetRepresentation().self().editor
print(segmentEditorWidget.availableEffectNames())
```

Center all views on a segment

This example shows how to center all slice views and 3D views on a segment. The segment's center is not the segment's centroid, but the centroid of the largest island in the effect, because the centroid can be in an empty region if the segment is made up of multiple islands.

```
segmentationNode = getNode("Segmentation")
segmentId = "Segment_2"

position = segmentationNode.GetSegmentCenterRAS(segmentId)
print(position)

# Center slice views and cameras on this position
for sliceNode in slicer.util.getNodesByClass('vtkMRMLSliceNode'):
    sliceNode.JumpSliceByCentering(*position)
for camera in slicer.util.getNodesByClass('vtkMRMLCameraNode'):
    camera.SetFocalPoint(position)
```

Read and write a segment as a numpy array

This example shows how to read and write voxels of binary labelmap representation of a segment as a numpy array.

```
volumeNode = getNode('MRHead')
segmentationNode = getNode('Segmentation')
segmentId = segmentationNode.GetSegmentation().GetSegmentIdBySegmentName('Segment_1')

# Get segment as numpy array
segmentArray = slicer.util.arrayFromSegmentBinaryLabelmap(segmentationNode, segmentId,␣
↪volumeNode)

# Modify the segmentation
segmentArray[:] = 0  # clear the segmentation
segmentArray[ slicer.util.arrayFromVolume(volumeNode) > 80 ] = 1  # create segment by␣
↪simple thresholding of an image
segmentArray[20:80, 40:90, 30:70] = 1  # fill a rectangular region using numpy indexing
slicer.util.updateSegmentBinaryLabelmapFromArray(segmentArray, segmentationNode,␣
↪segmentId, volumeNode)
```

Segment arrays can also be used in numpy operations to read/write the corresponding region of a volume:

```
# Get voxels of a volume within the segmentation and compute some statistics
volumeArray = slicer.util.arrayFromVolume(volumeNode)
volumeVoxelsInSegmentArray = volumeArray[ segmentArray > 0 ]
print(f"Lowest voxel value in segment: {volumeVoxelsInSegmentArray.min()}")
print(f"Highest voxel value in segment: {volumeVoxelsInSegmentArray.max()}")

# Modify the volume
# For example, increase the contrast inside the selected segment by a factor of 4x:
volumeArray[ segmentArray > 0 ] = volumeArray[ segmentArray > 0 ] * 4
# Indicate that we have completed modifications on the volume array
slicer.util.arrayFromVolumeModified(volumeNode)
```

Get centroid of a segment in world (RAS) coordinates

This example shows how to get centroid of a segment in world coordinates and show that position in all slice views.

```
segmentationNode = getNode("Segmentation")
segmentId = "Segment_1"

# Get array voxel coordinates
import numpy as np
seg = slicer.util.arrayFromSegmentBinaryLabelmap(segmentationNode, segmentId)
# numpy array has voxel coordinates in reverse order (KJI instead of IJK)
# and the array is cropped to minimum size in the segmentation
mean_KjiCropped = [coords.mean() for coords in np.nonzero(seg)]

# Get segmentation voxel coordinates
segImage = slicer.vtkOrientedImageData()
segmentationNode.GetBinaryLabelmapRepresentation(segmentId, segImage)
segImageExtent = segImage.GetExtent()
# origin of the array in voxel coordinates is determined by the start extent
mean_Ijk = [mean_KjiCropped[2], mean_KjiCropped[1], mean_KjiCropped[0]] + np.
↪array([segImageExtent[0], segImageExtent[2], segImageExtent[4]])

# Get segmentation physical coordinates
```

(continues on next page)

(continued from previous page)

```
ijkToWorld = vtk.vtkMatrix4x4()
segImage.GetImageToWorldMatrix(ijkToWorld)
mean_World = [0, 0, 0, 1]
ijkToWorld.MultiplyPoint(np.append(mean_Ijk,1.0), mean_World)
mean_World = mean_World[0:3]

# If segmentation node is transformed, apply that transform to get RAS coordinates
transformWorldToRas = vtk.vtkGeneralTransform()
slicer.vtkMRMLTransformNode.GetTransformBetweenNodes(segmentationNode.
↪GetParentTransformNode(), None, transformWorldToRas)
mean_Ras = transformWorldToRas.TransformPoint(mean_World)

# Show mean position value and jump to it in all slice viewers
print(mean_Ras)
slicer.modules.markups.logic().JumpSlicesToLocation(mean_Ras[0], mean_Ras[1], mean_
↪Ras[2], True)
```

Get histogram of a segmented region

```
# Generate example input data (volumeNode, segmentationNode, segmentId)
################################################

# Load source volume
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
volumeNode = sampleDataLogic.downloadMRBrainTumor1()

# Create segmentation
segmentationNode = slicer.vtkMRMLSegmentationNode()
slicer.mrmlScene.AddNode(segmentationNode)
segmentationNode.CreateDefaultDisplayNodes() # only needed for display
segmentationNode.SetReferenceImageGeometryParameterFromVolumeNode(volumeNode)

# Create segment
tumorSeed = vtk.vtkSphereSource()
tumorSeed.SetCenter(-6, 30, 28)
tumorSeed.SetRadius(25)
tumorSeed.Update()
segmentId = segmentationNode.AddSegmentFromClosedSurfaceRepresentation(tumorSeed.
↪GetOutput(), "Segment A", [1.0,0.0,0.0])

# Compute histogram
################################################

# Get voxel values of volume in the segmented region
import numpy as np
volumeArray = slicer.util.arrayFromVolume(volumeNode)
segmentArray = slicer.util.arrayFromSegmentBinaryLabelmap(segmentationNode, segmentId,␣
↪volumeNode)
segmentVoxels = volumeArray[segmentArray != 0]
```

(continues on next page)

(continued from previous page)

```
# Compute histogram
import numpy as np
histogram = np.histogram(segmentVoxels, bins=50)

# Plot histogram
################################################

slicer.util.plot(histogram, xColumnIndex = 1)
```

Get segments visible at a selected position

Show in the console names of segments visible at a markups control point position:

```
segmentationNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLSegmentationNode")
pointListNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLMarkupsFiducialNode")
sliceViewLabel = "Red"  # any slice view where segmentation node is visible works

def printSegmentNames(unused1=None, unused2=None):

  sliceViewNode = slicer.mrmlScene.GetNodeByID(f"vtkMRMLSliceNode{sliceViewLabel}")
  appLogic = slicer.app.applicationLogic()
  segmentationsDisplayableManager = appLogic.
↪GetViewDisplayableManagerByClassName(sliceViewNode,
↪"vtkMRMLSegmentationsDisplayableManager2D")
  ras = [0,0,0]
  pointListNode.GetNthControlPointPositionWorld(0, ras)
  segmentIds = vtk.vtkStringArray()
  segmentationsDisplayableManager.GetVisibleSegmentsForPosition(ras, segmentationNode.
↪GetDisplayNode(), segmentIds)
  for idIndex in range(segmentIds.GetNumberOfValues()):
    segment = segmentationNode.GetSegmentation().GetSegment(segmentIds.GetValue(idIndex))
    print("Segment found at position {0}: {1}".format(ras, segment.GetName()))

# Observe markup node changes
pointListNode.AddObserver(slicer.vtkMRMLMarkupsNode.PointModifiedEvent,␣
↪printSegmentNames)
printSegmentNames()
```

Set default segmentation options

Allow segments to overlap each other by default:

```
defaultSegmentEditorNode = slicer.vtkMRMLSegmentEditorNode()
defaultSegmentEditorNode.SetOverwriteMode(slicer.vtkMRMLSegmentEditorNode.OverwriteNone)
slicer.mrmlScene.AddDefaultNode(defaultSegmentEditorNode)
```

To always make this the default, add the lines above to your *.slicerrc.py file*.

How to run segment editor effects from a script

Editor effects are complex because they need to handle changing source volumes, undo/redo, masking operations, etc. Therefore, it is recommended to use the effect by instantiating a qMRMLSegmentEditorWidget or use/extract processing logic of the effect and use that from a script.

Use Segment editor effects from script (qMRMLSegmentEditorWidget)

Examples:

- brain tumor segmentation using grow from seeds effect
- AI-assisted brain tumor segmentation
- skin surface extraction using thresholding and smoothing
- mask a volume with segments and compute histogram for each region
- create fat/muscle/bone segment by thresholding and report volume of each segment
- segment cranial cavity automatically in dry bone skull CT
- remove patient table from CT image
- fill holes inside bones

Description of effect parameters are available *here*.

Use logic of effect from a script

This example shows how to perform operations on segmentations using VTK filters *extracted* from an effect:

- brain tumor segmentation using grow from seeds effect

Process segment using a VTK filter

This example shows how to apply a VTK filter to a segment that dilates the image by a specified margin.

```
segmentationNode = getNode("Segmentation")
segmentId = "Segment_1"
kernelSize = [3,1,5]

# Export segment as vtkImageData (via temporary labelmap volume node)
segmentIds = vtk.vtkStringArray()
segmentIds.InsertNextValue(segmentId)
labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
slicer.modules.segmentations.logic().ExportSegmentsToLabelmapNode(segmentationNode,␣
↪segmentIds, labelmapVolumeNode)

# Process segmentation
segmentImageData = labelmapVolumeNode.GetImageData()
erodeDilate = vtk.vtkImageDilateErode3D()
erodeDilate.SetInputData(segmentImageData)
erodeDilate.SetDilateValue(1)
erodeDilate.SetErodeValue(0)
erodeDilate.SetKernelSize(*kernelSize)
erodeDilate.Update()
segmentImageData.DeepCopy(erodeDilate.GetOutput())
```

(continues on next page)

(continued from previous page)

```
# Import segment from vtkImageData
slicer.modules.segmentations.logic().ImportLabelmapToSegmentationNode(labelmapVolumeNode,
↪ segmentationNode, segmentIds)

# Cleanup temporary nodes
slicer.mrmlScene.RemoveNode(labelmapVolumeNode.GetDisplayNode().GetColorNode())
slicer.mrmlScene.RemoveNode(labelmapVolumeNode)
```

Use segmentation files in Python - outside Slicer

You can use slicerio Python package (in any Python environment, not just within Slicer) to get information from segmentation (.seg.nrrd) files.

For example, a common need when training AI tools is to assemble data sets from various sources, which use different label values for the same segments. If data sets are in .seg.nrrd format, then segment names or standard terminology can be used to identify segments and then assign label values consistently.

Extract selected segments by standard terminology

Segments cannot be reliably identified using "name" (simple string label). Instead, it is recommended to use a standard terminology. This code snippet extracts selected segments from a segmentation and writes the result into a nrrd file.

```
# pip install slicerio

import slicerio

input_filename = "path/to/Segmentation.seg.nrrd"
output_filename = "path/to/SegmentationExtracted.seg.nrrd"
segments_to_labels = [
   ({"category": ["SCT", "123037004", "Anatomical Structure"], "type": ["SCT", "113197003
↪", "Ribs"]}, 1),
   ({"category": ["SCT", "123037004", "Anatomical Structure"], "type": ["SCT", "39607008
↪", "Lung"], "typeModifier": ["SCT", "24028007", "Right"]}, 3)
   ]

segmentation = slicerio.read_segmentation(input_filename)
extracted_segmentation = slicerio.extract_segments(segmentation, segments_to_labels)
slicerio.write_segmentation(output_filename, extracted_segmentation)
```

Extract selected segments by segment name

This code snippet extracts selected segments from a segmentation by segment name and writes it into a nrrd file.

```
# pip install slicerio

import slicerio
import nrrd

input_filename = "path/to/Segmentation.seg.nrrd"
output_filename = "path/to/SegmentationExtracted.seg.nrrd"
segment_names_to_labels = [("ribs", 10), ("right lung", 12), ("left lung", 6)]
```

(continues on next page)

(continued from previous page)

```
# Read voxels and metadata from a .seg.nrrd file
voxels, header = nrrd.read(input_filename)
# Get selected segments in a 3D numpy array and updated segment metadata
extracted_voxels, extracted_header = slicerio.extract_segments(voxels, header,␣
↪segmentation_info, segment_names_to_labels)
# Write extracted segments and metadata to .seg.nrrd file
nrrd.write(output_filename, extracted_voxels, extracted_header)
```

Clone a segment

A copy of the segment can be created by using `CopySegmentFromSegmentation` method:

```
segmentationNode = getNode("Segmentation")
sourceSegmentName = "Segment_1"

segmentation = segmentationNode.GetSegmentation()
sourceSegmentId = segmentation.GetSegmentIdBySegmentName(sourceSegmentName)
segmentation.CopySegmentFromSegmentation(segmentation, sourceSegmentId)
```

Resample segmentation to higher resolution

This code snippet can be used to resample internal binary labelmap representation of a segmentation to allow segmenting finer details

```
# Set inputs
volumeNode = getNode("MRHead")
segmentationNode = getNode("Segmentation")

# The higher the oversampling factor is the finer resolution the segmentation will be,
# at the cost of more memory usage and longer computation times.
# Note that oversampling by a factor of 2 increases memory usage by a factor of 2 * 2 *␣
↪2 = 8.
oversamplingFactor = 2.0

# Make spacing value uniform for all axes.
# It is useful for removing staircase artifacts in 3D reconstructions but may increase
# memory usage and computation time.
isotropicSpacing = True

# Update geometry of internal binary labelmap representation in segmentation node
segmentationGeometryLogic = slicer.vtkSlicerSegmentationGeometryLogic()
segmentationGeometryLogic.SetInputSegmentationNode(segmentationNode)
segmentationGeometryLogic.SetSourceGeometryNode(volumeNode)
segmentationGeometryLogic.SetOversamplingFactor(oversamplingFactor)
segmentationGeometryLogic.SetIsotropicSpacing(isotropicSpacing)
segmentationGeometryLogic.CalculateOutputGeometry()
segmentationGeometryLogic.SetReferenceImageGeometryInSegmentationNode()
segmentationGeometryLogic.ResampleLabelmapsInSegmentationNode()
```

Quantifying segments

Get volume of each segment

```
segmentationNode = getNode("Segmentation")

# Compute segment statistics
import SegmentStatistics
segStatLogic = SegmentStatistics.SegmentStatisticsLogic()
segStatLogic.getParameterNode().SetParameter("Segmentation", segmentationNode.GetID())
segStatLogic.computeStatistics()
stats = segStatLogic.getStatistics()

# Display volume of each segment
for segmentId in stats["SegmentIDs"]:
  volume_cm3 = stats[segmentId,"LabelmapSegmentStatisticsPlugin.volume_cm3"]
  segmentName = segmentationNode.GetSegmentation().GetSegment(segmentId).GetName()
  print(f"{segmentName} volume = {volume_cm3} cm3")
```

Get centroid of each segment

Place a markups control point at the centroid of each segment.

```
segmentationNode = getNode("Segmentation")

# Compute centroids
import SegmentStatistics
segStatLogic = SegmentStatistics.SegmentStatisticsLogic()
segStatLogic.getParameterNode().SetParameter("Segmentation", segmentationNode.GetID())
segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.centroid_
↪ras.enabled", str(True))
segStatLogic.computeStatistics()
stats = segStatLogic.getStatistics()

# Place a markup point in each centroid
pointListNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsFiducialNode")
pointListNode.CreateDefaultDisplayNodes()
for segmentId in stats["SegmentIDs"]:
  centroid_ras = stats[segmentId,"LabelmapSegmentStatisticsPlugin.centroid_ras"]
  segmentName = segmentationNode.GetSegmentation().GetSegment(segmentId).GetName()
  pointListNode.AddControlPoint(centroid_ras, segmentName)
```

Get size, position, and orientation of each segment

Get oriented bounding box and display them using markups ROI node.

Markups ROI

```
segmentationNode = getNode("Segmentation")

# Compute bounding boxes
import SegmentStatistics
segStatLogic = SegmentStatistics.SegmentStatisticsLogic()
```

(continues on next page)

(continued from previous page)

```
segStatLogic.getParameterNode().SetParameter("Segmentation", segmentationNode.GetID())
segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_origin_
↪ras.enabled",str(True))
segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪diameter_mm.enabled",str(True))
segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_x.enabled",str(True))
segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_y.enabled",str(True))
segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_z.enabled",str(True))
segStatLogic.computeStatistics()
stats = segStatLogic.getStatistics()

# Draw ROI for each oriented bounding box
import numpy as np
for segmentId in stats["SegmentIDs"]:
  # Get bounding box
  obb_origin_ras = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_origin_
↪ras"])
  obb_diameter_mm = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪diameter_mm"])
  obb_direction_ras_x = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_x"])
  obb_direction_ras_y = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_y"])
  obb_direction_ras_z = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_z"])
  # Create ROI
  segment = segmentationNode.GetSegmentation().GetSegment(segmentId)
  roi=slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsROINode")
  roi.SetName(segment.GetName() + " OBB")
  roi.GetDisplayNode().SetHandlesInteractive(False)  # do not let the user resize the box
  roi.SetSize(obb_diameter_mm)
  # Position and orient ROI using a transform
  obb_center_ras = obb_origin_ras+0.5*(obb_diameter_mm[0] * obb_direction_ras_x + obb_
↪diameter_mm[1] * obb_direction_ras_y + obb_diameter_mm[2] * obb_direction_ras_z)
  boundingBoxToRasTransform = np.row_stack((np.column_stack((obb_direction_ras_x, obb_
↪direction_ras_y, obb_direction_ras_z, obb_center_ras)), (0, 0, 0, 1)))
  boundingBoxToRasTransformMatrix = slicer.util.
↪vtkMatrixFromArray(boundingBoxToRasTransform)
  roi.SetAndObserveObjectToNodeMatrix(boundingBoxToRasTransformMatrix)
```

Note

Complete list of available segment statistics parameters can be obtained by running `segStatLogic.getParameterNode().GetParameterNames()`.

12.8.17 Registration

Register two images with BRAINSFit

Register two brain MRI images of the same patient acquired at different timepoints using BRAINSFit ("General registration (BRAINS)" module).

```
# Get sample input data
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
fixedVolumeNode = sampleDataLogic.downloadMRBrainTumor1()
movingVolumeNode = sampleDataLogic.downloadMRBrainTumor2()
# Create new nodes for output
transformedMovingVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode
↪")
transformNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLTransformNode")

# Run registration
parameters = {}
parameters["fixedVolume"] = fixedVolumeNode.GetID()
parameters["movingVolume"] = movingVolumeNode.GetID()
parameters["outputVolume"] = transformedMovingVolumeNode.GetID()
parameters["linearTransform"] = transformNode.GetID()
parameters["useRigid"] = True  # options include: "useRigid", "useAffine", "useBSpline"
parameters["initializeTransformMode"] = "useGeometryAlign"
parameters["samplingPercentage"] = 0.02
cliBrainsFitRigidNode = slicer.cli.run(slicer.modules.brainsfit, None, parameters, wait_
↪for_completion=True)

# Display fused result. Computed transformNode is automatically applied to the␣
↪movingVolumeNode.
# Ctrl + left-click-and-drag up/down to change opacity in slice views.
slicer.util.setSliceViewerLayers(background=fixedVolumeNode, foreground=movingVolumeNode,
↪ foregroundOpacity=0.5)
```

All parameters for BRAINSFit can be found here. The module can register any kind of images, not just brains, but for general-purpose registration extensions consider using more robust registration tools, such as SlicerElastix and SlicerANTs extensions.

12.8.18 Sequences

Access voxels of a 4D volume as numpy array

```
# Get sequence node
import SampleData
sequenceNode = SampleData.SampleDataLogic().downloadSample("CTPCardioSeq")
# Alternatively, get the first sequence node in the scene:
# sequenceNode = slicer.util.getNodesByClass("vtkMRMLSequenceNode")
```

(continues on next page)

(continued from previous page)

```
# Get voxels of itemIndex'th volume as numpy array
itemIndex = 5
voxelArray = slicer.util.arrayFromVolume(sequenceNode.GetNthDataNode(itemIndex))
```

Access voxels of a 4D volume as a single numpy array

Get all voxels of a 4D volume (3D volume sequence) as a numpy array called `voxelArray`. Dimensions of the array: `k`, `j`, `i`, `t` (first three are voxel coordinates, fourth is the volume index).

```
sequenceNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLSequenceNode")

# Preallocate a 4D numpy array that will hold the entire sequence
import numpy as np
dims = slicer.util.arrayFromVolume(sequenceNode.GetNthDataNode(0)).shape
voxelArray = np.zeros([dims[0], dims[1], dims[2], sequenceNode.GetNumberOfDataNodes()])
# Fill in the 4D array from the sequence node
for volumeIndex in range(sequenceNode.GetNumberOfDataNodes()):
    voxelArray[:, :, :, volumeIndex] = slicer.util.arrayFromVolume(sequenceNode.
↪GetNthDataNode(volumeIndex))
```

Get index value

```
print("Index value of {0}th item: {1} = {2} {3}".format(
  itemIndex,
  sequenceNode.GetIndexName(),
  sequenceNode.GetNthIndexValue(itemIndex),
  sequenceNode.GetIndexUnit()))
```

Browse a sequence and access currently displayed nodes

```
# Get a sequence node
import SampleData
sequenceNode = SampleData.SampleDataLogic().downloadSample("CTPCardioSeq")

# Find corresponding sequence browser node
browserNode = slicer.modules.sequences.logic().
↪GetFirstBrowserNodeForSequenceNode(sequenceNode)

# Print sequence information
print("Number of items in the sequence: {0}".format(browserNode.GetNumberOfItems()))
print("Index name: {0}".format(browserNode.GetMasterSequenceNode().GetIndexName()))

# Jump to a selected sequence item
browserNode.SetSelectedItemNumber(5)

# Get currently displayed volume node voxels as numpy array
volumeNode = browserNode.GetProxyNode(sequenceNode)
voxelArray = slicer.util.arrayFromVolume(volumeNode)
```

Concatenate all sequences in the scene into a new sequence

```
# Get all sequence nodes in the scene
sequenceNodes = slicer.util.getNodesByClass("vtkMRMLSequenceNode")
mergedSequenceNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLSequenceNode", "Merged␣
↪sequence")

# Merge all sequence nodes into a new sequence node
mergedIndexValue = 0
for sequenceNode in sequenceNodes:
  for itemIndex in range(sequenceNode.GetNumberOfDataNodes()):
    dataNode = sequenceNode.GetNthDataNode(itemIndex)
    mergedSequenceNode.SetDataNodeAtValue(dataNode, str(mergedIndexValue))
    mergedIndexValue += 1
  # Delete the sequence node we copied the data from, to prevent sharing of the same
  # node by multiple sequences
  slicer.mrmlScene.RemoveNode(sequenceNode)

# Create a sequence browser node for the new merged sequence
mergedSequenceBrowserNode = slicer.mrmlScene.AddNewNodeByClass(
↪"vtkMRMLSequenceBrowserNode", "Merged")
mergedSequenceBrowserNode.AddSynchronizedSequenceNode(mergedSequenceNode)
slicer.modules.sequences.toolBar().setActiveBrowserNode(mergedSequenceBrowserNode)
# Show proxy node in slice viewers
mergedProxyNode = mergedSequenceBrowserNode.GetProxyNode(mergedSequenceNode)
slicer.util.setSliceViewerLayers(background=mergedProxyNode)
```

Plot segments average intensity over time

This code snippet can be used to plot average intensity in specific regions (designated using segments in a segmentation node) of a volume sequence over time.

```
# inputs
volumeSequenceProxyNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")
segmentationNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLSegmentationNode")

# get volume sequence as numpy array
volumeSequenceBrowserNode = slicer.modules.sequences.logic().
↪GetFirstBrowserNodeForProxyNode(volumeSequenceProxyNode)
volumeSequenceNode = volumeSequenceBrowserNode.GetSequenceNode(volumeSequenceProxyNode)

# get voxels of visible segments as numpy arrays
segmentNames = []
segmentArrays = []
visibleSegmentIds = vtk.vtkStringArray()
segmentationNode.GetDisplayNode().GetVisibleSegmentIDs(visibleSegmentIds)
for segmentIdIndex in range(visibleSegmentIds.GetNumberOfValues()):
    segmentId = visibleSegmentIds.GetValue(segmentIdIndex)
    segmentArrays.append(slicer.util.arrayFromSegmentBinaryLabelmap(segmentationNode,␣
↪segmentId, volumeSequenceProxyNode))
    segmentNames.append(segmentationNode.GetSegmentation().GetSegment(segmentId).
↪GetName())
```

(continues on next page)

(continued from previous page)

```
# Create table that will contain time values and mean intensity value for each segment␣
↪for each time point
import numpy as np
intensityTable = np.zeros([volumeSequenceNode.GetNumberOfDataNodes(),␣
↪len(segmentArrays)+1])
intensityTableTimeColumn = 0
intensityTableColumnNames = [volumeSequenceNode.GetIndexName()] + segmentNames
for volumeIndex in range(volumeSequenceNode.GetNumberOfDataNodes()):
    intensityTable[volumeIndex, intensityTableTimeColumn] = volumeSequenceNode.
↪GetNthIndexValue(volumeIndex)
    for segmentIndex, segmentArray in enumerate(segmentArrays):
        voxelArray = slicer.util.arrayFromVolume(volumeSequenceNode.
↪GetNthDataNode(volumeIndex))
        intensityTable[volumeIndex, segmentIndex+1] = voxelArray[segmentArray>0].mean()

# Plot results
plotNodes = {}
slicer.util.plot(intensityTable, intensityTableTimeColumn, intensityTableColumnNames,
↪"Intensity", nodes=plotNodes)
# Set color and name of plots to match segment names and colors
for segmentIdIndex in range(visibleSegmentIds.GetNumberOfValues()):
    segment = segmentationNode.GetSegmentation().GetSegment(visibleSegmentIds.
↪GetValue(segmentIdIndex))
    seriesNode = plotNodes['series'][segmentIdIndex]
    seriesNode.SetColor(segment.GetColor())
    seriesNode.SetName(segment.GetName())
```

Export nodes warped by transform sequence

Warp a segmentation with a sequence of transforms and write each transformed segmentation to a ply file. It can be used on sequence registration results created as shown in this tutorial video.

```
# Inputs
transformSequenceNode = slicer.util.getNode("OutputTransforms")
segmentationNode = slicer.util.getNode("Segmentation")
segmentIndex = 0
outputFilePrefix = r"c:/tmp/20220312/seg"

# Ensure the segmentation contains closed surface representation
segmentationNode.CreateClosedSurfaceRepresentation()
# Create temporary node that will be warped
segmentModelNode = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLModelNode')

for itemIndex in range(transformSequenceNode.GetNumberOfDataNodes()):
  # Get a copy of the segment that will be transformed
  segment = segmentationNode.GetSegmentation().GetNthSegment(segmentIndex)
  slicer.modules.segmentations.logic().ExportSegmentToRepresentationNode(segment,␣
↪segmentModelNode)
  # Apply the transform
  transform = transformSequenceNode.GetNthDataNode(itemIndex).GetTransformToParent()
  segmentModelNode.ApplyTransform(transform)
  # Write to file
```

(continues on next page)

(continued from previous page)

```
  outputFileName = f"{outputFilePrefix}_{itemIndex:03}.ply"
  print(outputFileName)
  slicer.util.saveNode(segmentModelNode, outputFileName)

# Delete temporary node
slicer.mrmlScene.RemoveNode(segmentModelNode)
```

Create a 4D volume in Python - outside Slicer

You can write a seq.nrrd file (that Slicer can load as a volume sequence) from an img numpy array of with dimensions `t`, `i`, `j`, `k` (volume index, followed by voxel coordinates). `space origin` specifies the image origin. `space directions` specify the image axis directions and spacing (spacing is the Euclidean norm of the axis vector).

Prerequisite: install pynrrd.

```
import nrrd
header = {
    'type': 'int',
    'dimension': 4,
    'space': 'right-anterior-superior',
    'space directions': [[float('nan'), float('nan'), float('nan')], [1.953125, 0., 0.],␣
↪[0., 1.953125, 0.], [0., 0., 1.953125]],
    'kinds': ['list', 'domain', 'domain', 'domain'],
    'labels': ['frame', '', '', ''],
    'endian': 'little',
    'encoding': 'raw',
    'space origin': [-137.16099548,  -36.80649948, -309.71899414],
    'measurement frame': [[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]],
    'axis 0 index type': 'numeric',
    'axis 0 index values': '0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22␣
↪23 24 25'
}
nrrd.write("c:/tmp/test.seq.nrrd", img, header)
```

12.8.19 Subject hierarchy

Get the pseudo-singleton subject hierarchy node

It manages the whole hierarchy and provides functions to access and manipulate

```
shNode = slicer.mrmlScene.GetSubjectHierarchyNode()
```

Create subject hierarchy item

```
# If it is for a data node, it is automatically created, but the create function can be␣
↪used to set parent:
shNode.CreateItem(parentItemID, dataNode)
# If it is a hierarchy item without a data node, then the create function must be used:
shNode.CreateSubjectItem(parentItemID, name)
shNode.CreateFolderItem(parentItemID, name)
shNode.CreateHierarchyItem(parentItemID, name, level) # Advanced method to set level␣
↪attribute manually (usually subject, study, or folder, but it can be a virtual branch␣
↪for example)
```

Get subject hierarchy item

Items in subject hierarchy are uniquely identified by integer IDs

```
# Get scene item ID first because it is the root item:
sceneItemID = shNode.GetSceneItemID()
# Get direct child by name
subjectItemID = shNode.GetItemChildWithName(sceneItemID, "Subject_1")
# Get item for data node
itemID = shNode.GetItemByDataNode(dataNode)
# Get item by UID (such as DICOM)
itemID = shNode.GetItemByUID(slicer.vtkMRMLSubjectHierarchyConstants.GetDICOMUIDName(),␣
↪seriesInstanceUid)
itemID = shNode.GetItemByUIDList(slicer.vtkMRMLSubjectHierarchyConstants.
↪GetDICOMInstanceUIDName(), instanceUID)
# Invalid item ID for checking validity of a given ID (most functions return the invalid␣
↪ID when item is not found)
invalidItemID = slicer.vtkMRMLSubjectHierarchyNode.GetInvalidItemID()
```

Traverse children of a subject hierarchy item

```
children = vtk.vtkIdList()
shNode.GetItemChildren(parent, children) # Add a third argument with value True for␣
↪recursive query
for i in range(children.GetNumberOfIds()):
  child = children.GetId(i)
  ...
```

Manipulate subject hierarchy item

Instead of node operations on the individual subject hierarchy nodes, item operations are performed on the one subject hierarchy node.

```
# Set item name
shNode.SetItemName(itemID, "NewName")
# Set item parent (reparent)
shNode.SetItemParent(itemID, newParentItemID)
# Set visibility of data node associated to an item
shNode.SetItemDisplayVisibility(itemID, 1)
# Set visibility of whole branch
# Note: Folder-type items (fodler, subject, study, etc.) create their own display nodes␣
↪when show/hiding from UI.
#       The displayable managers use SH information to determine visibility of an item,␣
↪so no need to show/hide individual leaf nodes any more.
#       Once the folder display node is created, it can be shown hidden simply using␣
↪shNode.SetItemDisplayVisibility
# From python, this is how to trigger creating a folder display node
pluginHandler = slicer.qSlicerSubjectHierarchyPluginHandler().instance()
folderPlugin = pluginHandler.pluginByName("Folder")
folderPlugin.setDisplayVisibility(folderItemID, 1)
```

Filter items in TreeView or ComboBox

Displayed items can be filtered using *setAttributeFilter* method. An example of the usage can be found in the unit test. Modified version here:

```
print(shTreeView.displayedItemCount()) # 5
shTreeView.setAttributeFilter("DICOM.Modality") # Nodes must have this attribute
print(shTreeView.displayedItemCount()) # 3
shTreeView.setAttributeFilter("DICOM.Modality","CT") # Have attribute and equal ``CT``
print(shTreeView.displayedItemCount()) # 1
shTreeView.removeAttributeFilter()
print(shTreeView.displayedItemCount()) # 5
```

Listen to subject hierarchy item events

The subject hierarchy node sends the node item id as calldata. Item IDs are vtkIdType, which are NOT vtkObjects. You need to use vtk.calldata_type(vtk.VTK_LONG) (otherwise the application crashes).

```
class MyListenerClass(VTKObservationMixin):
  def __init__(self):
    VTKObservationMixin.__init__(self)

    shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
    self.addObserver(shNode, shNode.SubjectHierarchyItemModifiedEvent, self.
↪shItemModifiedEvent)

  @vtk.calldata_type(vtk.VTK_LONG)
  def shItemModifiedEvent(self, caller, eventId, callData):
    print("SH Node modified")
    print("SH item ID: {0}".format(callData))
```

Subject hierarchy plugin offering view context menu action

If an object that supports view context menus (e.g. markups) is right-clicked in a slice or 3D view, it can offer custom actions.

Due to internal limitations, in order to use view menus in scripted plugins, it needs to be registered differently, so that the Python API can be fully built by the time this function is called. The following changes are necessary compared to regular initialization:

1. Remove the following line from constructor

```
AbstractScriptedSubjectHierarchyPlugin.__init__(self, scriptedPlugin)
```

2. In addition to the initialization where the scripted plugin is instantialized and the source set, the plugin also needs to be registered manually:

```
pluginHandler = slicer.qSlicerSubjectHierarchyPluginHandler.instance()
pluginHandler.registerPlugin(scriptedPlugin)
```

This is a complete example. It must be saved as `ViewContextMenu.py` and placed in a folder that is added to "Additional module paths" in Application Settings / Modules section.

```
import vtk, qt, ctk, slicer
from slicer.ScriptedLoadableModule import *
```

(continues on next page)

(continued from previous page)

```
from slicer.util import VTKObservationMixin

from SubjectHierarchyPlugins import AbstractScriptedSubjectHierarchyPlugin

class ViewContextMenu(ScriptedLoadableModule):

  def __init__(self, parent):
    ScriptedLoadableModule.__init__(self, parent)
    self.parent.title = "Context menu example"
    self.parent.categories = ["Examples"]
    self.parent.contributors = ["Steve Pieper (Isomics, Inc.)"]
    slicer.app.connect("startupCompleted()", self.onStartupCompleted)

  def onStartupCompleted(self):
    """register subject hierarchy plugin once app is initialized"""
    import SubjectHierarchyPlugins
    from ViewContextMenu import ViewContextMenuSubjectHierarchyPlugin
    scriptedPlugin = slicer.qSlicerSubjectHierarchyScriptedPlugin(None)
    scriptedPlugin.setPythonSource(ViewContextMenuSubjectHierarchyPlugin.filePath)
    pluginHandler = slicer.qSlicerSubjectHierarchyPluginHandler.instance()
    pluginHandler.registerPlugin(scriptedPlugin)
    print("ViewContextMenuSubjectHierarchyPlugin loaded")

class ViewContextMenuSubjectHierarchyPlugin(AbstractScriptedSubjectHierarchyPlugin):

  # Necessary static member to be able to set python source to scripted subject␣
↪hierarchy plugin
  filePath = __file__

  def __init__(self, scriptedPlugin):
    self.viewAction = qt.QAction("CUSTOM VIEW...", scriptedPlugin)
    self.viewAction.objectName = "CustomViewAction"
    # Set the action's position in the menu: by using `SectionNode+5` we place the action␣
↪in a new section, after "node actions" section.
    slicer.qSlicerSubjectHierarchyAbstractPlugin.setActionPosition(self.viewAction,␣
↪slicer.qSlicerSubjectHierarchyAbstractPlugin.SectionNode+5)
    self.viewAction.connect("triggered()", self.onViewAction)

  def viewContextMenuActions(self):
    return [self.viewAction]

  def showViewContextMenuActionsForItem(self, itemID, eventData=None):
    # We can decide here if we want to show this action based on the itemID or eventData␣
↪(ViewNodeID, ...).
    print(f"itemID: {itemID}")
    print(f"eventData: {eventData}")
    self.viewAction.visible = True

  def onViewAction(self):
    print("Custom view action is called")
    slicer.util.messageBox("This works!")
```

Use allowlist to customize view menu

When right-clicking certain types of nodes in the 2D/3D views, a subject hierarchy menu pops up. If menu actions need to be removed, an allowlist can be used to specify the ones that should show up.

```
pluginHandler = slicer.qSlicerSubjectHierarchyPluginHandler.instance()
pluginLogic = pluginHandler.pluginLogic()

# Display list of all available view context menu action names.
# This will print something like this: 'EditPropertiesAction',
↪'MouseModeViewTransformAction', 'MouseModeAdjustWindowLevelAction', 'MouseModePlaceAction',
↪ ...).
print(pluginLogic.registeredViewContextMenuActionNames)

# Hide all the other menu items and show only "Rename point":
pluginLogic.allowedViewContextMenuActionNames = ["RenamePointAction"]
```

Save files to directory structure matching subject hierarchy folders

This code snippet saves all the storable files (volumes, transforms, markups, etc.) into a folder structure that mirrors the structure of the subject hierarchy tree (file folders have the same name as subject hierarchy folders).

```
def exportNodes(shFolderItemId, outputFolder):
    # Get items in the folder
    childIds = vtk.vtkIdList()
    shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
    shNode.GetItemChildren(shFolderItemId, childIds)
    if childIds.GetNumberOfIds() == 0:
        return
    # Create output folder
    import os
    os.makedirs(outputFolder, exist_ok=True)
    # Write each child item to file
    for itemIdIndex in range(childIds.GetNumberOfIds()):
        shItemId = childIds.GetId(itemIdIndex)
        # Write node to file (if storable)
        dataNode = shNode.GetItemDataNode(shItemId)
        if dataNode and dataNode.IsA("vtkMRMLStorableNode") and dataNode.
↪GetStorageNode():
            storageNode = dataNode.GetStorageNode()
            filename = os.path.basename(storageNode.GetFileName())
            filepath = outputFolder + "/" + filename
            slicer.util.exportNode(dataNode, filepath)
        # Write all children of this child item
        grandChildIds = vtk.vtkIdList()
        shNode.GetItemChildren(shItemId, grandChildIds)
        if grandChildIds.GetNumberOfIds() > 0:
            exportNodes(shItemId, outputFolder+"/"+shNode.GetItemName(shItemId))

shNode = slicer.vtkMRMLSubjectHierarchyNode.GetSubjectHierarchyNode(slicer.mrmlScene)
outputFolder = "c:/tmp/test20211123"
slicer.app.ioManager().addDefaultStorageNodes()
exportNodes(shNode.GetSceneItemID(), outputFolder)
```

12.8.20 Tractography

Export a tract (FiberBundle) to Blender, including color

> **Note**
>
> An interactive version of this script is now included in the SlicerDMRI extension (module code). After installing SlicerDMRI, go to *Modules -> Diffusion -> Import and Export -> Export tractography to PLY (mesh)*.

The example below shows how to export a tractography "FiberBundleNode" to a PLY file:

```
lineDisplayNode = getNode("*LineDisplay*")
plyFilePath = "/tmp/fibers.ply"
outputCoordinateSystem = "RAS"  # can be "RAS" (still used in neuroimaging) or "LPS"␣
↪(most commonly used coordinate system in medical image computing)

tuber = vtk.vtkTubeFilter()
tuber.SetInputData(lineDisplayNode.GetOutputPolyData())
tuber.Update()
tubes = tuber.GetOutputDataObject(0)
scalars = tubes.GetPointData().GetArray(0)
scalars.SetName("scalars")

triangles = vtk.vtkTriangleFilter()
triangles.SetInputData(tubes)
triangles.Update()

colorNode = lineDisplayNode.GetColorNode()
lookupTable = vtk.vtkLookupTable()
lookupTable.DeepCopy(colorNode.GetLookupTable())
lookupTable.SetTableRange(0,1)

plyWriter = vtk.vtkPLYWriter()

if outputCoordinateSystem == "RAS":
    plyWriter.SetInputData(triangles.GetOutput())
elif outputCoordinateSystem == "LPS":
    transformRasToLps = vtk.vtkTransformPolyDataFilter()
    rasToLps = vtk.vtkTransform()
    rasToLps.Scale(-1, -1, 1)
    transformRasToLps.SetTransform(rasToLps)
    transformRasToLps.SetInputData(triangles.GetOutput())
    plyWriter.SetInputConnection(transformRasToLps.GetOutputPort())
else:
    raise RuntimeError("Invalid output coordinate system")

plyWriter.SetLookupTable(lookupTable)
plyWriter.SetArrayName("scalars")

plyWriter.SetFileName(plyFilePath)
plyWriter.Write()
```

Iterate over tract (FiberBundle) streamline points

This example shows how to access the points in each line of a FiberBundle as a numpy array (view).

```
from vtk.util.numpy_support import vtk_to_numpy

fb = getNode("FiberBundle_F") # <- fill in node ID here

# get point data as 1d array
points = slicer.util.arrayFromModelPoints(fb)

# get line cell ids as 1d array
line_ids = vtk_to_numpy(fb.GetPolyData().GetLines().GetData())

# VTK cell ids are stored as
#   [ N0 c0_id0 ... c0_id0
#     N1 c1_id0 ... c1_idN1 ]
# so we need to
# - read point count for each line (cell)
# - grab the ids in that range from `line_ids` array defined above
# - index the `points` array by those ids
cur_idx = 1
for _ in range(pd.GetLines().GetNumberOfCells()):
  # - read point count for this line (cell)
  count = lines[cur_idx - 1]
  # - grab the ids in that range from `lines`
  index_array = line_ids[ cur_idx : cur_idx + count]
  # update to the next range
  cur_idx += count + 1
  # - index the point array by those ids
  line_points = points[index_array]
  # do work here
```

12.8.21 Transforms

Get a notification if a transform is modified

```
def onTransformNodeModified(transformNode, unusedArg2=None, unusedArg3=None):
   transformMatrix = vtk.vtkMatrix4x4()
   transformNode.GetMatrixTransformToWorld(transformMatrix)
   print("Position: [{0}, {1}, {2}]".format(transformMatrix.GetElement(0,3),␣
↪transformMatrix.GetElement(1,3), transformMatrix.GetElement(2,3)))

transformNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLTransformNode")
transformNode.AddObserver(slicer.vtkMRMLTransformNode.TransformModifiedEvent,␣
↪onTransformNodeModified)
```

Rotate a node around a specified point

Set up the scene:

- Add a markup point list node (centerOfRotationMarkupsNode) with a single point to specify center of rotation.
- Add a rotation transform (rotationTransformNode) that will be edited in Transforms module to specify rotation angles.

- Add a transform (finalTransformNode) and apply it (not harden) to those nodes (images, models, etc.) that you want to rotate around the center of rotation point.

Then run the script below, go to Transforms module, select rotationTransformNode, and move rotation sliders.

```
# This markups point list node specifies the center of rotation
centerOfRotationMarkupsNode = getNode("F")
# This transform can be  edited in Transforms module
rotationTransformNode = getNode("LinearTransform_3")
# This transform has to be applied to the image, model, etc.
finalTransformNode = getNode("LinearTransform_4")

def updateFinalTransform(unusedArg1=None, unusedArg2=None, unusedArg3=None):
  rotationMatrix = vtk.vtkMatrix4x4()
  rotationTransformNode.GetMatrixTransformToParent(rotationMatrix)
  rotationCenterPointCoord = [0.0, 0.0, 0.0]
  centerOfRotationMarkupsNode.GetNthControlPointPositionWorld(0,␣
↪rotationCenterPointCoord)
  finalTransform = vtk.vtkTransform()
  finalTransform.Translate(rotationCenterPointCoord)
  finalTransform.Concatenate(rotationMatrix)
  finalTransform.Translate(-rotationCenterPointCoord[0], -rotationCenterPointCoord[1], -
↪rotationCenterPointCoord[2])
  finalTransformNode.SetAndObserveMatrixTransformToParent(finalTransform.GetMatrix())

# Manual initial update
updateFinalTransform()

# Automatic update when point is moved or transform is modified
rotationTransformNodeObserver = rotationTransformNode.AddObserver(slicer.
↪vtkMRMLTransformNode.TransformModifiedEvent, updateFinalTransform)
centerOfRotationMarkupsNodeObserver = centerOfRotationMarkupsNode.AddObserver(slicer.
↪vtkMRMLMarkupsNode.PointModifiedEvent, updateFinalTransform)

# Execute these lines to stop automatic updates:
# rotationTransformNode.RemoveObserver(rotationTransformNodeObserver)
# centerOfRotationMarkupsNode.RemoveObserver(centerOfRotationMarkupsNodeObserver)
```

Rotate a node around a specified line

Set up the scene:

- Add a markup line node (rotationAxisMarkupsNode) with 2 points to specify rotation axis.
- Add a rotation transform (rotationTransformNode) that will be edited in Transforms module to specify rotation angle.
- Add a transform (finalTransformNode) and apply it (not harden) to those nodes (images, models, etc.) that you want to rotate around the line.

Then run the script below, go to Transforms module, select rotationTransformNode, and move Edit / Rotation / IS slider.

```
# This markups point list node specifies the center of rotation
rotationAxisMarkupsNode = getNode("L")
# This transform can be edited in Transforms module (Edit / Rotation / IS slider)
rotationTransformNode = getNode("LinearTransform_3")
```

(continues on next page)

(continued from previous page)

```
# This transform has to be applied to the image, model, etc.
finalTransformNode = getNode("LinearTransform_4")

def updateFinalTransform(unusedArg1=None, unusedArg2=None, unusedArg3=None):
  import numpy as np
  rotationAxisPoint1_World = np.zeros(3)
  rotationAxisMarkupsNode.GetNthControlPointPositionWorld(0, rotationAxisPoint1_World)
  rotationAxisPoint2_World = np.zeros(3)
  rotationAxisMarkupsNode.GetNthControlPointPositionWorld(1, rotationAxisPoint2_World)
  axisDirectionZ_World = rotationAxisPoint2_World-rotationAxisPoint1_World
  axisDirectionZ_World = axisDirectionZ_World/np.linalg.norm(axisDirectionZ_World)
  # Get transformation between world coordinate system and rotation axis aligned␣
↪coordinate system
  worldToRotationAxisTransform = vtk.vtkMatrix4x4()
  p=vtk.vtkPlaneSource()
  p.SetNormal(axisDirectionZ_World)
  axisOrigin = np.array(p.GetOrigin())
  axisDirectionX_World = np.array(p.GetPoint1())-axisOrigin
  axisDirectionY_World = np.array(p.GetPoint2())-axisOrigin
  rotationAxisToWorldTransform = np.row_stack((np.column_stack((axisDirectionX_World,␣
↪axisDirectionY_World, axisDirectionZ_World, rotationAxisPoint1_World)), (0, 0, 0, 1)))
  rotationAxisToWorldTransformMatrix = slicer.util.
↪vtkMatrixFromArray(rotationAxisToWorldTransform)
  worldToRotationAxisTransformMatrix = slicer.util.vtkMatrixFromArray(np.linalg.
↪inv(rotationAxisToWorldTransform))
  # Compute transformation chain
  rotationMatrix = vtk.vtkMatrix4x4()
  rotationTransformNode.GetMatrixTransformToParent(rotationMatrix)
  finalTransform = vtk.vtkTransform()
  finalTransform.Concatenate(rotationAxisToWorldTransformMatrix)
  finalTransform.Concatenate(rotationMatrix)
  finalTransform.Concatenate(worldToRotationAxisTransformMatrix)
  finalTransformNode.SetAndObserveMatrixTransformToParent(finalTransform.GetMatrix())

# Manual initial update
updateFinalTransform()

# Automatic update when point is moved or transform is modified
rotationTransformNodeObserver = rotationTransformNode.AddObserver(slicer.
↪vtkMRMLTransformNode.TransformModifiedEvent, updateFinalTransform)
rotationAxisMarkupsNodeObserver = rotationAxisMarkupsNode.AddObserver(slicer.
↪vtkMRMLMarkupsNode.PointModifiedEvent, updateFinalTransform)

# Execute these lines to stop automatic updates:
# rotationTransformNode.RemoveObserver(rotationTransformNodeObserver)
# rotationAxisMarkupsNode.RemoveObserver(rotationAxisMarkupsNodeObserver)
```

Convert between ITK and Slicer linear transforms

ITK transform files store the resampling transform ("transform from parent") in LPS coordinate system. Slicer displays the modeling transform ("transform to parent") in RAS coordinate system. The following code snippets show how to compute the matrix that Slicer displays from an ITK transform file.

```
# Copy the content between the following triple-quotes to a file called 'LinearTransform.
↪tfm', and load into Slicer

"""
#Insight Transform File V1.0
#Transform 0
Transform: AffineTransform_double_3_3
Parameters: -0.213789703753722 -0.7715409537503184 0.5991815411500568 -0.
↪6877744504386172 0.5544585467133546 0.46855311897413043 -0.6937292468195099 -0.
↪3119299226790803 -0.6491836839020759 -6.080443282989144 10.318378593490745 159.
↪38003926210894
FixedParameters: -4.363375593759671 -23.302979114450725 -56.27550193307294
"""

import numpy as np
import re

def read_itk_affine_transform(filename):
    with open(filename, 'r') as f:
        lines = f.readlines()

    # Extract Parameters
    params_line = next(line for line in lines if line.startswith('Parameters:'))
    params = list(map(float, params_line.removeprefix('Parameters:').strip().split()))
    rotation = np.array(params[:9]).reshape((3, 3))
    translation = np.array(params[9:12])

    # Extract FixedParameters (center of rotation)
    fixed_params_line = next(line for line in lines if line.startswith('FixedParameters:
↪'))
    rotation_center = np.array(list(map(float, fixed_params_line.removeprefix(
↪'FixedParameters:').strip().split())))

    # Get homogeneous transformation matrix translation component from transform␣
↪parameters and fixed parameters.
    # The homogeneous transformation matrix rotation part is the same as rotation stored␣
↪in parameters.
    # However, the translation part is not directly stored in these values, but it is␣
↪computed from the fixed and moving parameters (and the value is called "offset").
    # See computation implementation in MatrixOffsetTransformBase::ComputeOffset
    # https://github.com/InsightSoftwareConsortium/ITK/blob/
↪09305ce7c3509139ab0ca12047f8e6793b01db74/Modules/Core/Transform/include/
↪itkMatrixOffsetTransformBase.hxx#L611-L629
    offset = translation + rotation_center - rotation @ rotation_center

    # Construct homogeneous transformation matrix
    transform = np.eye(4)
    transform[:3, :3] = rotation
    transform[:3, 3] = offset

    return transform

def itk_to_slicer_transform(itk_transform):
```

(continues on next page)

(continued from previous page)

```
    # ITK transform: from parent, using LPS coordinate system
    # Transform displayed in Slicer: to parent, using RAS coordinate system
    transform_from_parent_LPS = itk_transform
    ras2lps = np.diag([-1, -1, 1, 1])
    transform_from_parent_RAS = ras2lps @ transform_from_parent_LPS @ ras2lps
    transform_to_parent_RAS = np.linalg.inv(transform_from_parent_RAS)
    return transform_to_parent_RAS

filename = "path/to/LinearTransform.tfm"
itk_tfm = read_itk_affine_transform(filename)
slicer_tfm = itk_to_slicer_transform(itk_tfm)
with np.printoptions(precision=6, suppress=True):
    print(slicer_tfm)

# Running the code above in Python should print the following output.
# This output should match the display the loaded .tfm file in the Transforms module:
# [[  0.929794  -0.269457  -0.25075   -52.64097 ]
#  [  0.038348   0.748446  -0.662086   46.126957]
#  [  0.366077   0.605988   0.706234    0.481854]
#  [  0.         0.         0.          1.      ]]
```

Apply a transform to a transformable node

Python:

```
transformToParentMatrix = vtk.vtkMatrix4x4()
...
transformNode.SetMatrixTransformToParent(matrix)
transformableNode.SetAndObserveTransformNodeID(transformNode.GetID())
```

C++:

```
vtkNew<vtkMRMLTransformNode> transformNode;
scene->AddNode(transformNode.GetPointer());
...
vtkNew<vtkMatrix4x4> matrix;
...
transform->SetMatrixTransformToParent( matrix.GetPointer() );
...
vtkMRMLVolumeNode* transformableNode = ...; // or vtkMRMLModelNode*...
transformableNode->SetAndObserveTransformNodeID( transformNode->GetID() );
```

Set a transformation matrix from a numpy array

```
# Create a 4x4 transformation matrix as numpy array
transformNode = ...
transformMatrixNP = np.array(
  [[0.92979,-0.26946,-0.25075,52.64097],
  [0.03835, 0.74845, -0.66209, -46.12696],
  [0.36608, 0.60599, 0.70623, -0.48185],
  [0, 0, 0, 1]])
```

(continues on next page)

(continued from previous page)

```
# Update matrix in transform node
transformNode.SetAndObserveMatrixTransformToParent(slicer.util.
↪vtkMatrixFromArray(transformMatrixNP))
```

Example of moving a volume along a trajectory using a transform

```
# Load sample volume
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
mrHead = sampleDataLogic.downloadMRHead()

# Create transform and apply to sample volume
transformNode = slicer.vtkMRMLTransformNode()
slicer.mrmlScene.AddNode(transformNode)
mrHead.SetAndObserveTransformNodeID(transformNode.GetID())

# How to move a volume along a trajectory using a transform:
import time
import math
transformMatrix = vtk.vtkMatrix4x4()
for xPos in range(-30,30):
  transformMatrix.SetElement(0,3, xPos)
  transformMatrix.SetElement(1,3, math.sin(xPos)*10)
  transformNode.SetMatrixTransformToParent(transformMatrix)
  slicer.app.processEvents()
  time.sleep(0.02)
# Note: for longer animations use qt.QTimer.singleShot(100, callbackFunction)
# instead of a for loop.
```

Combine multiple transforms

Because a transform node is also a transformable node, it is possible to concatenate transforms with each other.

Python:

```
transformNode2.SetAndObserveTransformNodeID(transformNode1.GetID())
transformableNode.SetAndObserveTransformNodeID(transformNode2.GetID())
```

C++:

```
vtkMRMLTransformNode* transformNode1 = ...;
vtkMRMLTransformNode* transformNode2 = ...;
transformNode2->SetAndObserveTransformNodeID(transformNode1->GetID());
transformable->SetAndObserveTransformNodeID(transformNode2->GetID());
```

Convert the transform to a grid transform

Any transform can be converted to a grid transform (also known as displacement field transform):

```
transformNode=slicer.util.getNode('LinearTransform_3')
referenceVolumeNode=slicer.util.getNode('MRHead')
slicer.modules.transforms.logic().ConvertToGridTransform(transformNode,␣
↪referenceVolumeNode)
```

Note

- Conversion to grid transform is useful because some software cannot use inverse transforms or can only use grid transforms.
- Displacement field transforms are saved to file differently than displacement field volumes: displacement vectors in transforms are converted to LPS coordinate system on saving, displacement vectors in volumes are saved to file unchanged.

Export the displacement magnitude of the transform as a volume

```
transformNode=slicer.util.getNode('LinearTransform_3')
referenceVolumeNode=slicer.util.getNode('MRHead')
slicer.modules.transforms.logic().CreateDisplacementVolumeFromTransform(transformNode,␣
→referenceVolumeNode, False)
```

Visualize the displacement magnitude as a color volume

```
transformNode=slicer.util.getNode('LinearTransform_3')
referenceVolumeNode=slicer.util.getNode('MRHead')
slicer.modules.transforms.logic().CreateDisplacementVolumeFromTransform(transformNode,␣
→referenceVolumeNode, True)
```

12.8.22 Volumes

Load volume from file

```
loadedVolumeNode = slicer.util.loadVolume("c:/Users/abc/Documents/MRHead.nrrd")
```

Additional options may be specified in `properties` argument. For example, load an image stack by disabling `singleFile` option:

```
loadedVolumeNode = slicer.util.loadVolume("c:/Users/abc/Documents/SomeImage/file001.png",
→ {"singleFile": False})
```

Note

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
- `colorNodeID` (string): ID of the color node used to display the volume. Default is `vtkMRMLColorTableNodeGrey` for scalar volume and `vtkMRMLColorTableNodeFileGenericColors.txt` for labelmap volume.

Save volume to file

Get the first volume node in the scene and save as .nrrd file. To save in any other supported file format, change the output file name.

```
volumeNode = slicer.mrmlScene.GetFirstNodeByClass('vtkMRMLScalarVolumeNode')
slicer.util.exportNode(volumeNode, "c:/tmp/test.nrrd")
```

If you are saving to a format with optional compression, like nrrd, compression is on by default. Saving is much faster with compression turned off but the files may be much larger (about 3x for medical images).

```
slicer.util.exportNode(volumeNode, imagePath, {"useCompression": 0})
```

By default, parent transforms are ignored. To export the node in the world coordinate system (all transforms hardened), set `world=True`.

```
slicer.util.exportNode(volumeNode, imagePath, {"useCompression": 0}, world=True)
```

`saveNode` method can be used instead of `exportNode` to update the current storage options (filename, compression options, etc.) in the scene.

Load volume from .vti file

Slicer does not provide reader for VTK XML image data file format (as they are not commonly used for storing medical images and they cannot store image axis directions) but such files can be read by using this script:

```
reader=vtk.vtkXMLImageDataReader()
reader.SetFileName("/path/to/file.vti")
reader.Update()
imageData = reader.GetOutput()
spacing = imageData.GetSpacing()
origin = imageData.GetOrigin()
imageData.SetOrigin(0,0,0)
imageData.SetSpacing(1,1,1)
volumeNode=slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode")
volumeNode.SetAndObserveImageData(imageData)
volumeNode.SetSpacing(spacing)
volumeNode.SetOrigin(origin)
slicer.util.setSliceViewerLayers(background=volumeNode, fit=True)
```

Load volume from a remote server

Download a volume from a remote server by an URL and load it into the scene using the code snippets below.

Note

Downloaded data is temporarily stored in the application's cache folder and if the checksum of the already downloaded data matches the specified checksum (:) then the file is retrieved from the cache instead of being downloaded again. To compute digest with algo *SHA256*, you can run `slicer.util.computeChecksum("SHA256", "path/to/file")()`.

Simple download

```
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
loadedNodes = sampleDataLogic.downloadFromURL(
  nodeNames="MRHead",
  fileNames="MR-head25.nrrd",
  uris="https://github.com/Slicer/SlicerTestingData/releases/download/SHA256/
↪cc211f0dfd9a05ca3841ce1141b292898b2dd2d3f08286affadf823a7e58df93",
  checksums="SHA256:cc211f0dfd9a05ca3841ce1141b292898b2dd2d3f08286affadf823a7e58df93")[0]
```

Download with interruptible progress reporting

```
import SampleData

def reportProgress(msg, level=None):
  # Print progress in the console
  print("Loading... {0}%".format(sampleDataLogic.downloadPercent))
  # Abort download if cancel is clicked in progress bar
  if slicer.progressWindow.wasCanceled:
    raise Exception("download aborted")
  # Update progress window
  slicer.progressWindow.show()
  slicer.progressWindow.activateWindow()
  slicer.progressWindow.setValue(int(sampleDataLogic.downloadPercent))
  slicer.progressWindow.setLabelText("Downloading...")
  # Process events to allow screen to refresh
  slicer.app.processEvents()

try:
  volumeNode = None
  slicer.progressWindow = slicer.util.createProgressDialog()
  sampleDataLogic = SampleData.SampleDataLogic()
  sampleDataLogic.logMessage = reportProgress
  loadedNodes = sampleDataLogic.downloadFromURL(
    nodeNames="MRHead",
    fileNames="MR-head25.nrrd",
    uris="https://github.com/Slicer/SlicerTestingData/releases/download/SHA256/
↪cc211f0dfd9a05ca3841ce1141b292898b2dd2d3f08286affadf823a7e58df93",
    checksums="SHA256:cc211f0dfd9a05ca3841ce1141b292898b2dd2d3f08286affadf823a7e58df93")
  volumeNode = loadedNodes[0]
finally:
  slicer.progressWindow.close()
```

Show volume rendering automatically when a volume is loaded

To show volume rendering of a volume automatically when it is loaded, add the lines below to your *.slicerrc.py file*.

```
@vtk.calldata_type(vtk.VTK_OBJECT)
def onNodeAdded(caller, event, calldata):
  node = calldata
  if isinstance(node, slicer.vtkMRMLVolumeNode):
    # Call showVolumeRendering using a timer instead of calling it directly
    # to allow the volume loading to fully complete.
    qt.QTimer.singleShot(0, lambda: showVolumeRendering(node))

def showVolumeRendering(volumeNode):
  print("Show volume rendering of node " + volumeNode.GetName())
  volRenLogic = slicer.modules.volumerendering.logic()
  displayNode = volRenLogic.CreateDefaultVolumeRenderingNodes(volumeNode)
  displayNode.SetVisibility(True)
  scalarRange = volumeNode.GetImageData().GetScalarRange()
  if scalarRange[1]-scalarRange[0] < 1500:
    # Small dynamic range, probably MRI
    displayNode.GetVolumePropertyNode().Copy(volRenLogic.GetPresetByName("MR-Default"))
  else:
    # Larger dynamic range, probably CT
    displayNode.GetVolumePropertyNode().Copy(volRenLogic.GetPresetByName("CT-Chest-
↪Contrast-Enhanced"))

slicer.mrmlScene.AddObserver(slicer.vtkMRMLScene.NodeAddedEvent, onNodeAdded)
```

Show volume rendering using maximum intensity projection

```
def showVolumeRenderingMIP(volumeNode, useSliceViewColors=True):
  """Render volume using maximum intensity projection
  :param useSliceViewColors: use the same colors as in slice views.
  """
  # Get/create volume rendering display node
  volRenLogic = slicer.modules.volumerendering.logic()
  displayNode = volRenLogic.GetFirstVolumeRenderingDisplayNode(volumeNode)
  if not displayNode:
    displayNode = volRenLogic.CreateDefaultVolumeRenderingNodes(volumeNode)
  # Choose MIP volume rendering preset
  if useSliceViewColors:
    volRenLogic.CopyDisplayToVolumeRenderingDisplayNode(displayNode)
  else:
    scalarRange = volumeNode.GetImageData().GetScalarRange()
    if scalarRange[1]-scalarRange[0] < 1500:
      # Small dynamic range, probably MRI
      displayNode.GetVolumePropertyNode().Copy(volRenLogic.GetPresetByName("MR-MIP"))
    else:
      # Larger dynamic range, probably CT
      displayNode.GetVolumePropertyNode().Copy(volRenLogic.GetPresetByName("CT-MIP"))
  # Switch views to MIP mode
  for viewNode in slicer.util.getNodesByClass("vtkMRMLViewNode"):
    viewNode.SetRaycastTechnique(slicer.vtkMRMLViewNode.MaximumIntensityProjection)
```

(continues on next page)

(continued from previous page)

```
  # Show volume rendering
  displayNode.SetVisibility(True)

volumeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")
showVolumeRenderingMIP(volumeNode)
```

Show volume rendering making soft tissues transparent

```
def showTransparentRendering(volumeNode, maxOpacity=0.2, gradientThreshold=30.0):
  """Make constant regions transparent and the entire volume somewhat transparent
  :param maxOpacity: lower value makes the volume more transparent overall
    (value is between 0.0 and 1.0)
  :param gradientThreshold: regions that has gradient value below this threshold will be␣
↪made transparent
    (minimum value is 0.0, higher values make more tissues transparent, starting with␣
↪soft tissues)
  """
  # Get/create volume rendering display node
  volRenLogic = slicer.modules.volumerendering.logic()
  displayNode = volRenLogic.GetFirstVolumeRenderingDisplayNode(volumeNode)
  if not displayNode:
    displayNode = volRenLogic.CreateDefaultVolumeRenderingNodes(volumeNode)
  # Set up gradient vs opacity transfer function
  gradientOpacityTransferFunction = displayNode.GetVolumePropertyNode().
↪GetVolumeProperty().GetGradientOpacity()
  gradientOpacityTransferFunction.RemoveAllPoints()
  gradientOpacityTransferFunction.AddPoint(0, 0.0)
  gradientOpacityTransferFunction.AddPoint(gradientThreshold-1, 0.0)
  gradientOpacityTransferFunction.AddPoint(gradientThreshold+1, maxOpacity)
  # Show volume rendering
  displayNode.SetVisibility(True)

volumeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")
showTransparentRendering(volumeNode, 0.2, 30.0)
```

Automatically load volumes that are copied into a folder

This example shows how to implement a simple background task by using a timer. The background task is to check for any new volume files in folder and if there is any then automatically load it.

There are more efficient methods for file system monitoring or exchanging image data in real-time (for example, using OpenIGTLink), the example below is just for demonstration purposes.

```
incomingVolumeFolder = "c:/tmp/incoming"
incomingVolumesProcessed = []

def checkForNewVolumes():
  # Check if there is a new file in the
  from os import listdir
  from os.path import isfile, join
  for f in listdir(incomingVolumeFolder):
    if f in incomingVolumesProcessed:
```

(continues on next page)

(continued from previous page)

```
      # This is an incoming file, it was already there
      continue
    filePath = join(incomingVolumeFolder, f)
    if not isfile(filePath):
      # ignore directories
      continue
    logging.info("Loading new file: " + f)
    incomingVolumesProcessed.append(f)
    slicer.util.loadVolume(filePath)
  # Check again in 3000ms
  qt.QTimer.singleShot(3000, checkForNewVolumes)

# Start monitoring
checkForNewVolumes()
```

Extract slice from volume

This code snippet can extract arbitrarily oriented slice from a volume. The slice position, orientation, and size is specified by a markup plane node.

```
# Inputs
volumeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLScalarVolumeNode")  # input 3D␣
↪volume
planeNode = slicer.mrmlScene.GetFirstNodeByClass("vtkMRMLMarkupsPlaneNode")  # input␣
↪markup plane
spacing = [0.5, 0.5, 1.0]  # spacing of the extracted image slice

# Create slice image node
sliceImageNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode")
sliceImageNode.CreateDefaultDisplayNodes()
imageData = vtk.vtkImageData()
sliceImageNode.SetAndObserveImageData(imageData)
# Set image size
sizeWorld = planeNode.GetSizeWorld()
imageData.SetExtent(0, int(sizeWorld[0]/spacing[0]-1), 0, int(sizeWorld[1]/spacing[1]-1),
↪ 0, 0)
imageData.AllocateScalars(vtk.VTK_UNSIGNED_CHAR, 1)
# Set spacing
sliceImageNode.SetSpacing(spacing)
# Set directions
import numpy as np
ijkToRASDirections = np.eye(3)
planeNode.GetAxesWorld(ijkToRASDirections[0:3, 0], ijkToRASDirections[0:3, 1],␣
↪ijkToRASDirections[0:3, 2])
sliceImageNode.SetIJKToRASDirections(ijkToRASDirections)
# Set origin
origin = np.array(planeNode.GetCenter()) - ijkToRASDirections[0:3, 0] * sizeWorld[0] / 2␣
↪- ijkToRASDirections[0:3, 1] * sizeWorld[1] / 2
sliceImageNode.SetOrigin(origin)

# Resample the volume to the slice image node
referenceVolumeNode = sliceImageNode
```

(continues on next page)

(continued from previous page)

```
parameters = {
    "inputVolume": volumeNode.GetID(),
    "outputVolume": sliceImageNode.GetID(),
    "referenceVolume": sliceImageNode.GetID(),
    "interpolationType": "linear"
}
parameterNode = slicer.cli.run(slicer.modules.resamplescalarvectordwivolume, None,␣
↪parameters, wait_for_completion=True, update_display=False)
slicer.mrmlScene.RemoveNode(parameterNode)
```

Extract randomly oriented slabs of given shape from a volume

Returns a numpy array of sliceCount random tiles.

```
def randomSlices(volume, sliceCount, sliceShape):
  layoutManager = slicer.app.layoutManager()
  redWidget = layoutManager.sliceWidget("Red")
  sliceNode = redWidget.mrmlSliceNode()
  sliceNode.SetDimensions(*sliceShape, 1)
  sliceNode.SetFieldOfView(*sliceShape, 1)
  bounds = [0]*6
  volume.GetRASBounds(bounds)
  imageReslice = redWidget.sliceLogic().GetBackgroundLayer().GetReslice()

  sliceSize = sliceShape[0] * sliceShape[1]
  X = numpy.zeros([sliceCount, sliceSize])

  for sliceIndex in range(sliceCount):
    position = numpy.random.rand(3) * 2 - 1
    position = [bounds[0] + bounds[1]-bounds[0] * position[0],
      bounds[2] + bounds[3]-bounds[2] * position[1],
      bounds[4] + bounds[5]-bounds[4] * position[2]]
    normal = numpy.random.rand(3) * 2 - 1
    normal = normal / numpy.linalg.norm(normal)
    transverse = numpy.cross(normal, [0,0,1])
    orientation = 0
    sliceNode.SetSliceToRASByNTP( normal[0], normal[1], normal[2],
      transverse[0], transverse[1], transverse[2],
      position[0], position[1], position[2],
      orientation)
    if sliceIndex % 100 == 0:
      slicer.app.processEvents()
    imageReslice.Update()
    imageData = imageReslice.GetOutputDataObject(0)
    array = vtk.util.numpy_support.vtk_to_numpy(imageData.GetPointData().GetScalars())
    X[sliceIndex] = array
  return X
```

Clone a volume

This example shows how to clone the MRHead sample volume, including its pixel data and display settings.

```
sourceVolumeNode = slicer.util.getNode("MRHead")
volumesLogic = slicer.modules.volumes.logic()
clonedVolumeNode = volumesLogic.CloneVolume(slicer.mrmlScene, sourceVolumeNode, "Cloned␣
↪volume")
```

Create a new volume

This example shows how to create a new empty volume. The "Image Maker" extension contains a module that allows creating a volume from scratch without programming.

```
nodeName = "MyNewVolume"
imageSize = [512, 512, 512]
voxelType=vtk.VTK_UNSIGNED_CHAR
imageOrigin = [0.0, 0.0, 0.0]
imageSpacing = [1.0, 1.0, 1.0]
imageDirections = [[1,0,0], [0,1,0], [0,0,1]]
fillVoxelValue = 0

# Create an empty image volume, filled with fillVoxelValue
imageData = vtk.vtkImageData()
imageData.SetDimensions(imageSize)
imageData.AllocateScalars(voxelType, 1)
imageData.GetPointData().GetScalars().Fill(fillVoxelValue)
# Create volume node
volumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode", nodeName)
volumeNode.SetOrigin(imageOrigin)
volumeNode.SetSpacing(imageSpacing)
volumeNode.SetIJKToRASDirections(imageDirections)
volumeNode.SetAndObserveImageData(imageData)
volumeNode.CreateDefaultDisplayNodes()
volumeNode.CreateDefaultStorageNode()
```

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()
```

Note

Origin and spacing must be set on the volume node instead of the image data.

Create a new volume from ROI

This example shows how to create a new empty volume with a specified voxel size, with axis directions and extents set from a markups ROI node.

```
def createVolumeFromRoi(exportRoi, spacingMm, fillValue=0, numberOfComponents=1):
    import math
    roiDiameter = exportRoi.GetSize()
    roiOrigin_Roi = [-roiDiameter[0]/2, -roiDiameter[1]/2, -roiDiameter[2]/2, 1]
    roiToRas = exportRoi.GetObjectToWorldMatrix()
    exportVolumeSize = [int(math.ceil(diameterComponent/spacingMm)) for␣
↪diameterComponent in roiDiameter]
    # Create image data
    exportImageData = vtk.vtkImageData()
    exportImageData.SetExtent(0, exportVolumeSize[0]-1, 0, exportVolumeSize[1]-1, 0,␣
↪exportVolumeSize[2]-1)
    exportImageData.AllocateScalars(vtk.VTK_DOUBLE, numberOfComponents)
    exportImageData.GetPointData().GetScalars().Fill(fillValue)
    # Create volume node
    exportVolume = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode" if␣
↪numberOfComponents==1 else "vtkMRMLVectorVolumeNode")
    exportVolume.SetAndObserveImageData(exportImageData)
    exportVolume.SetIJKToRASDirections(roiToRas.GetElement(0,0), roiToRas.GetElement(0,
↪1), roiToRas.GetElement(0,2), roiToRas.GetElement(1,0), roiToRas.GetElement(1,1),␣
↪roiToRas.GetElement(1,2), roiToRas.GetElement(2,0), roiToRas.GetElement(2,1), roiToRas.
↪GetElement(2,2))
    exportVolume.SetSpacing(spacingMm, spacingMm, spacingMm)
    roiOrigin_Ras = roiToRas.MultiplyPoint(roiOrigin_Roi)
    exportVolume.SetOrigin(roiOrigin_Ras[0:3])
    return exportVolume

# Create volume node from ROI node "R"
roiNode = getNode('R')
volumeNode = createVolumeFromRoi(roiNode, 0.5, 120)
# Show in slice views and set its window/level
slicer.util.setSliceViewerLayers(background=volumeNode)
volumeNode.GetScalarVolumeDisplayNode().AutoWindowLevelOff()
volumeNode.GetScalarVolumeDisplayNode().SetWindowLevel(110,130)
```

Get value of a volume at specific voxel coordinates

This example shows how to get voxel value of “volumeNode” at “ijk” volume voxel coordinates.

```
volumeNode = slicer.util.getNode("MRHead")
ijk = [20,40,30]  # volume voxel coordinates

voxels = slicer.util.arrayFromVolume(volumeNode)  # get voxels as a numpy array
voxelValue = voxels[ijk[2], ijk[1], ijk[0]]  # note that numpy array index order is kji␣
↪(not ijk)
```

Modify voxels in a volume

Typically the fastest and simplest way of modifying voxels is by using numpy operators. Voxels can be retrieved in a numpy array using the `array` method and modified using standard numpy methods. For example, threshold a volume:

```
nodeName = "MRHead"
thresholdValue = 100
voxelArray = array(nodeName) # get voxels as numpy array
voxelArray[voxelArray < thresholdValue] = 0 # modify voxel values
getNode(nodeName).Modified() # at the end of all processing, notify Slicer that the␣
↪image modification is completed
```

This example shows how to change voxels values of the MRHead sample volume. The values will be computed by function `f(r,a,s,) = (r-10)*(r-10)+(a+15)*(a+15)+s*s`.

```
volumeNode=slicer.util.getNode("MRHead")
ijkToRas = vtk.vtkMatrix4x4()
volumeNode.GetIJKToRASMatrix(ijkToRas)
imageData=volumeNode.GetImageData()
extent = imageData.GetExtent()
for k in range(extent[4], extent[5]+1):
  for j in range(extent[2], extent[3]+1):
    for i in range(extent[0], extent[1]+1):
      position_Ijk=[i, j, k, 1]
      position_Ras=ijkToRas.MultiplyPoint(position_Ijk)
      r=position_Ras[0]
      a=position_Ras[1]
      s=position_Ras[2]
      functionValue=(r-10)*(r-10)+(a+15)*(a+15)+s*s
      imageData.SetScalarComponentFromDouble(i,j,k,0,functionValue)
imageData.Modified()
```

Get volume voxel coordinates from markup control point RAS coordinates

This example shows how to get voxel coordinate of a volume corresponding to a markup control point position.

```
# Inputs
volumeNode = getNode("MRHead")
pointListNode = getNode("F")
markupsIndex = 0

# Get point coordinate in RAS
point_Ras = [0, 0, 0]
pointListNode.GetNthControlPointPositionWorld(markupsIndex, point_Ras)

# If volume node is transformed, apply that transform to get volume's RAS coordinates
transformRasToVolumeRas = vtk.vtkGeneralTransform()
slicer.vtkMRMLTransformNode.GetTransformBetweenNodes(None, volumeNode.
↪GetParentTransformNode(), transformRasToVolumeRas)
point_VolumeRas = transformRasToVolumeRas.TransformPoint(point_Ras)

# Get voxel coordinates from physical coordinates
volumeRasToIjk = vtk.vtkMatrix4x4()
volumeNode.GetRASToIJKMatrix(volumeRasToIjk)
```

(continues on next page)

(continued from previous page)

```
point_Ijk = [0, 0, 0, 1]
volumeRasToIjk.MultiplyPoint(np.append(point_VolumeRas,1.0), point_Ijk)
point_Ijk = [ int(round(c)) for c in point_Ijk[0:3] ]

# Print output
print(point_Ijk)
```

Get markup control point RAS coordinates from volume voxel coordinates

This example shows how to get position of maximum intensity voxel of a volume (determined by numpy, in IJK coordinates) in RAS coordinates so that it can be marked with a markup control point.

```
# Inputs
volumeNode = getNode("MRHead")
pointListNode = getNode("F")

# Get voxel position in IJK coordinate system
import numpy as np
volumeArray = slicer.util.arrayFromVolume(volumeNode)
# Get position of highest voxel value
point_Kji = np.where(volumeArray == volumeArray.max())
point_Ijk = [point_Kji[2][0], point_Kji[1][0], point_Kji[0][0]]

# Get physical coordinates from voxel coordinates
volumeIjkToRas = vtk.vtkMatrix4x4()
volumeNode.GetIJKToRASMatrix(volumeIjkToRas)
point_VolumeRas = [0, 0, 0, 1]
volumeIjkToRas.MultiplyPoint(np.append(point_Ijk,1.0), point_VolumeRas)

# If volume node is transformed, apply that transform to get volume's RAS coordinates
transformVolumeRasToRas = vtk.vtkGeneralTransform()
slicer.vtkMRMLTransformNode.GetTransformBetweenNodes(volumeNode.GetParentTransformNode(),
↪ None, transformVolumeRasToRas)
point_Ras = transformVolumeRasToRas.TransformPoint(point_VolumeRas[0:3])

# Add a markup at the computed position and print its coordinates
pointListNode.AddControlPoint((point_Ras[0], point_Ras[1], point_Ras[2]), "max")
print(point_Ras)
```

Get the values of all voxels for a label value

If you have a background image called ‘Volume’ and a mask called ‘Volume-label’ created with the Segment Editor you could do something like this:

```
import numpy
volume = array("Volume")
label = array("Volume-label")
points  = numpy.where( label == 1 )  # or use another label number depending on what you␣
↪segmented
values  = volume[points] # this will be a list of the label values
values.mean() # should match the mean value of LabelStatistics calculation as a double-
↪check
```

(continues on next page)

(continued from previous page)

```
numpy.savetxt("values.txt", values)
```

Access values in a DTI tensor volume

This example shows how to access individual tensors at the voxel level.

First load your DWI volume and estimate tensors to produce a DTI volume called 'Output DTI Volume'.

Then open the python window: View->Python console.

Use this command to access tensors through numpy:

```
tensors = array("Output DTI Volume")
```

Type the following code into the Python window to access all tensor components using vtk commands:

```
volumeNode=slicer.util.getNode("Output DTI Volume")
imageData=volumeNode.GetImageData()
tensors = imageData.GetPointData().GetTensors()
extent = imageData.GetExtent()
idx = 0
for k in range(extent[4], extent[5]+1):
  for j in range(extent[2], extent[3]+1):
    for i in range(extent[0], extent[1]+1):
      tensors.GetTuple9(idx)
      idx += 1
```

Change window/level (brightness/contrast) or colormap of a volume

This example shows how to change window/level of the MRHead sample volume.

```
volumeNode = getNode("MRHead")
displayNode = volumeNode.GetDisplayNode()
displayNode.AutoWindowLevelOff()
displayNode.SetWindow(50)
displayNode.SetLevel(100)
```

Change color mapping from grayscale to rainbow:

```
displayNode.SetAndObserveColorNodeID("vtkMRMLColorTableNodeRainbow")
```

Make mouse left-click and drag on the image adjust window/level

In older Slicer versions, by default, left-click and drag in a slice view adjusted window/level of the displayed image. Window/level adjustment is now a new mouse mode that can be activated by clicking on its toolbar button or running this code:

```
slicer.app.applicationLogic().GetInteractionNode().SetCurrentInteractionMode(slicer.
↪vtkMRMLInteractionNode.AdjustWindowLevel)
```

Reset field of view to show background volume maximized

Equivalent to click small rectangle button ("Adjust the slice viewer's field of view...") in the slice view controller.

```
slicer.util.resetSliceViews()
```

Rotate slice views to volume plane

Aligns slice views to volume axes, shows original image acquisition planes in slice views.

```
volumeNode = slicer.util.getNode("MRHead")
layoutManager = slicer.app.layoutManager()
for sliceViewName in layoutManager.sliceViewNames():
  layoutManager.sliceWidget(sliceViewName).mrmlSliceNode().
↪RotateToVolumePlane(volumeNode)
```

Iterate over current visible slice views, and set foreground and background images

```
slicer.util.setSliceViewerLayers(background=mrVolume, foreground=ctVolume)
```

Internally, this method performs something like this:

```
layoutManager = slicer.app.layoutManager()
for sliceViewName in layoutManager.sliceViewNames():
  compositeNode = layoutManager.sliceWidget(sliceViewName).sliceLogic().
↪GetSliceCompositeNode()
  # Setup background volume
  compositeNode.SetBackgroundVolumeID(mrVolume.GetID())
  # Setup foreground volume
  compositeNode.SetForegroundVolumeID(ctVolume.GetID())
  # Change opacity
  compositeNode.SetForegroundOpacity(0.3)
```

Show a volume in slice views

Recommended:

```
volumeNode = slicer.util.getNode("YourVolumeNode")
slicer.util.setSliceViewerLayers(background=volumeNode)
```

or

Show volume in all visible views where volume selection propagation is enabled:

```
volumeNode = slicer.util.getNode("YourVolumeNode")
applicationLogic = slicer.app.applicationLogic()
selectionNode = applicationLogic.GetSelectionNode()
selectionNode.SetSecondaryVolumeID(volumeNode.GetID())
applicationLogic.PropagateForegroundVolumeSelection(0)
```

or

Show volume in selected views:

```
n =  slicer.util.getNode("YourVolumeNode")
for color in ["Red", "Yellow", "Green"]:
  slicer.app.layoutManager().sliceWidget(color).sliceLogic().GetSliceCompositeNode().
↪SetForegroundVolumeID(n.GetID())
```

Change opacity of foreground volume in slice views

```
slicer.util.setSliceViewerLayers(foregroundOpacity=0.4)
```

or

Change opacity in a selected view

```
lm = slicer.app.layoutManager()
sliceLogic = lm.sliceWidget("Red").sliceLogic()
compositeNode = sliceLogic.GetSliceCompositeNode()
compositeNode.SetForegroundOpacity(0.4)
```

Turning off interpolation

You can turn off interpolation for newly loaded volumes with this script from Steve Pieper.

```
def NoInterpolate(caller,event):
  for node in slicer.util.getNodes("*").values():
    if node.IsA("vtkMRMLScalarVolumeDisplayNode"):
      node.SetInterpolate(0)

slicer.mrmlScene.AddObserver(slicer.mrmlScene.NodeAddedEvent, NoInterpolate)
```

You can place this code snippet in your *.slicerrc.py file* to always disable interpolation by default.

Running an ITK filter in Python using SimpleITK

Open the "Sample Data" module and download "MR Head", then paste the following snippet in Python console:

```
import SampleData
import SimpleITK as sitk
import sitkUtils

# Get input volume node
inputVolumeNode = SampleData.SampleDataLogic().downloadMRHead()
# Create new volume node for output
outputVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode",
↪"MRHeadFiltered")

# Run processing
inputImage = sitkUtils.PullVolumeFromSlicer(inputVolumeNode)
filter = sitk.SignedMaurerDistanceMapImageFilter()
outputImage = filter.Execute(inputImage)
sitkUtils.PushVolumeToSlicer(outputImage, outputVolumeNode)

# Show processing result
slicer.util.setSliceViewerLayers(background=outputVolumeNode)
```

More information:

- See the SimpleITK documentation for SimpleITK examples: https://simpleitk.org/doxygen/latest/html/examples.html
- sitkUtils in Slicer is used for pushing and pulling images from Slicer to SimpleITK: https://github.com/Slicer/Slicer/blob/main/Base/Python/sitkUtils.py

Get axial slice as numpy array

An axis-aligned (axial/sagittal/coronal/) slices of a volume can be extracted using simple numpy array indexing. For example:

```
import SampleData
volumeNode = SampleData.SampleDataLogic().downloadMRHead()
sliceIndex = 12

voxels = slicer.util.arrayFromVolume(volumeNode)  # Get volume as numpy array
slice = voxels[sliceIndex:,:]  # Get one slice of the volume as numpy array
```

Get reformatted image from a slice viewer as numpy array

Set up `red` slice viewer to show thick slab reconstructed from 3 slices:

```
sliceNodeID = "vtkMRMLSliceNodeRed"

# Get image data from slice view
sliceNode = slicer.mrmlScene.GetNodeByID(sliceNodeID)
appLogic = slicer.app.applicationLogic()
sliceLogic = appLogic.GetSliceLogic(sliceNode)
sliceLayerLogic = sliceLogic.GetBackgroundLayer()
reslice = sliceLayerLogic.GetReslice()
reslicedImage = vtk.vtkImageData()
reslicedImage.DeepCopy(reslice.GetOutput())

# Create new volume node using resliced image
volumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLScalarVolumeNode")
volumeNode.SetIJKToRASMatrix(sliceNode.GetXYToRAS())
volumeNode.SetAndObserveImageData(reslicedImage)
volumeNode.CreateDefaultDisplayNodes()
volumeNode.CreateDefaultStorageNode()

# Get voxels as a numpy array
voxels = slicer.util.arrayFromVolume(volumeNode)
print(voxels.shape)
```

Combine multiple volumes into one

This example combines two volumes into a new one by subtracting one from the other.

```
import SampleData
[input1Volume, input2Volume] = SampleData.SampleDataLogic().downloadDentalSurgery()

import slicer.util
a = slicer.util.arrayFromVolume(input1Volume)
```

(continues on next page)

(continued from previous page)

```
b = slicer.util.arrayFromVolume(input2Volume)

# `a` and `b` are numpy arrays,
# they can be combined using any numpy array operations
# to produce the result array `c`
c = b - a

volumeNode = slicer.modules.volumes.logic().CloneVolume(input1Volume, "Difference")
slicer.util.updateVolumeFromArray(volumeNode, c)
slicer.util.setSliceViewerLayers(background=volumeNode)
```

Add noise to image

This example shows how to add simulated noise to a volume.

```
import SampleData
import numpy as np

# Get a sample input volume node
volumeNode = SampleData.SampleDataLogic().downloadMRHead()

# Get volume as numpy array and add noise
voxels = slicer.util.arrayFromVolume(volumeNode)
voxels[:] = voxels + np.random.normal(0.0, 20.0, size=voxels.shape)
slicer.util.arrayFromVolumeModified(volumeNode)
```

Mask volume using segmentation

This example shows how to blank out voxels of a volume outside all segments.

```
# Input nodes
volumeNode = getNode("MRHead")
segmentationNode = getNode("Segmentation")

# Write segmentation to labelmap volume node with a geometry that matches the volume node
labelmapVolumeNode = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
slicer.modules.segmentations.logic().
↪ExportVisibleSegmentsToLabelmapNode(segmentationNode, labelmapVolumeNode, volumeNode)

# Masking
import numpy as np
voxels = slicer.util.arrayFromVolume(volumeNode)
mask = slicer.util.arrayFromVolume(labelmapVolumeNode)
maskedVoxels = np.copy(voxels)  # we don't want to modify the original volume
maskedVoxels[mask==0] = 0

# Write masked volume to volume node and show it
maskedVolumeNode = slicer.modules.volumes.logic().CloneVolume(volumeNode, "Masked")
slicer.util.updateVolumeFromArray(maskedVolumeNode, maskedVoxels)
slicer.util.setSliceViewerLayers(background=maskedVolumeNode)
```

Apply random deformations to image

This example shows how to apply random translation, rotation, and deformations to a volume to simulate variation in patient positioning, soft tissue motion, and random anatomical variations. Control points are placed on a regularly spaced grid and then each control point is displaced by a random amount. Thin-plate spline transform is computed from the original and transformed point list.

https://gist.github.com/lassoan/428af5285da75dc033d32ebff65ba940

Thick slab reconstruction and maximum/minimum intensity volume projections

Set up `red` slice viewer to show thick slab reconstructed from 3 slices:

```
sliceNode = slicer.mrmlScene.GetNodeByID("vtkMRMLSliceNodeRed")
appLogic = slicer.app.applicationLogic()
sliceLogic = appLogic.GetSliceLogic(sliceNode)
sliceLayerLogic = sliceLogic.GetBackgroundLayer()
reslice = sliceLayerLogic.GetReslice()
reslice.SetSlabModeToMean()
reslice.SetSlabNumberOfSlices(10) # mean of 10 slices will computed
reslice.SetSlabSliceSpacingFraction(0.3) # spacing between each slice is 0.3 pixel
↪(total 10 * 0.3 = 3 pixel neighborhood)
sliceNode.Modified()
```

Set up `red` slice viewer to show maximum intensity projection (MIP):

```
sliceNode = slicer.mrmlScene.GetNodeByID("vtkMRMLSliceNodeRed")
appLogic = slicer.app.applicationLogic()
sliceLogic = appLogic.GetSliceLogic(sliceNode)
sliceLayerLogic = sliceLogic.GetBackgroundLayer()
reslice = sliceLayerLogic.GetReslice()
reslice.SetSlabModeToMax()
reslice.SetSlabNumberOfSlices(600) # use a large number of slices (600) to cover the
↪entire volume
reslice.SetSlabSliceSpacingFraction(0.5) # spacing between slices are 0.5 pixel
↪(supersampling is useful to reduce interpolation artifacts)
sliceNode.Modified()
```

The projected image is available in a `vtkImageData` object by calling `reslice.GetOutput()`.

Display volume using volume rendering

```
  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)
```

C++:

```
qSlicerAbstractCoreModule* volumeRenderingModule =
  qSlicerCoreApplication::application()->moduleManager()->module("VolumeRendering");
vtkSlicerVolumeRenderingLogic* volumeRenderingLogic =
```

(continues on next page)

(continued from previous page)

```
  volumeRenderingModule ?␣
↪vtkSlicerVolumeRenderingLogic::SafeDownCast(volumeRenderingModule->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);
  }
```

Apply a custom volume rendering color/opacity transfer function

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

Volume rendering logic has utility functions to help you create those transfer functions: SetWindowLevelToVolumeProp, SetThresholdToVolumeProp, SetLabelMapToVolumeProp.

Limit volume rendering to a specific region of the volume

```
displayNode.SetAndObserveROINodeID(roiNode.GetID())
displayNode.CroppingEnabled = True
```

C++:

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

Register a new Volume Rendering mapper

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

C++:

```
void qSlicerMyABCVolumeRenderingModule::setup()
{
  vtkMRMLThreeDViewDisplayableManagerFactory::GetInstance()->
    RegisterDisplayableManager("vtkMRMLMyABCVolumeRenderingDisplayableManager");

  this->Superclass::setup();
```

(continues on next page)

(continued from previous page)

```
  qSlicerAbstractCoreModule* volumeRenderingModule =
    qSlicerCoreApplication::application()->moduleManager()->module("VolumeRendering");
  if (volumeRenderingModule)
    {
    vtkNew<vtkMRMLMyABCVolumeRenderingDisplayNode> displayNode;
    vtkSlicerVolumeRenderingLogic* volumeRenderingLogic =
      vtkSlicerVolumeRenderingLogic::SafeDownCast(volumeRenderingModule->logic());
    volumeRenderingLogic->RegisterRenderingMethod(
      "My ABC Volume Rendering", displayNode->GetClassName());
    }
  else
    {
    qWarning() << "Volume Rendering module is not found";
    }
}
```

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

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:

```
<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 79.2550277709961 0.428571432828903 415.119384765625 1 641 1"␣
↪gradientOpacity="4 0 1 160.25 1" colorTransfer="16 0 0 0 0 98.7223 0.196078431372549 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 79.2550277709961 0.428571432828903 415.119384765625 1 641 1"␣
↪gradientOpacity="4 0 1 160.25 1" colorTransfer="16 0 0 0 0 98.7223 0.196078431372549 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:

```
presetsScenePath = "MyPresets.mrml"

# Read presets scene
customPresetsScene = slicer.vtkMRMLScene()
vrPropNode = slicer.vtkMRMLVolumePropertyNode()
customPresetsScene.RegisterNodeClass(vrPropNode)
customPresetsScene.SetURL(presetsScenePath)
customPresetsScene.Connect()

# Add presets to volume rendering logic
vrLogic = slicer.modules.volumerendering.logic()
presetsScene = vrLogic.GetPresetsScene()
vrNodes = customPresetsScene.GetNodesByClass("vtkMRMLVolumePropertyNode")
vrNodes.UnRegister(None)
for itemNum in range(vrNodes.GetNumberOfItems()):
  node = vrNodes.GetItemAsObject(itemNum)
  vrLogic.AddPreset(node)
```

12.8.23 Batch processing

Iterate through dicom series

This examples shows how to perform an operation on each series in the dicom database.

```
db = slicer.dicomDatabase
patients = db.patients()
patientCount = 0
for patient in patients:
  patientCount += 1
  print(f"Patient {patient} ({patientCount} of {len(patients)})")
  for study in db.studiesForPatient(patient):
    print(f"Study {study}")
    for series in db.seriesForStudy(study):
      print(f"Series {series}")
      temporaryDir = qt.QTemporaryDir()
      for instanceUID in db.instancesForSeries(series):
        qt.QFile.copy(db.fileForInstance(instanceUID), f"{temporaryDir.path()}/
↪{instanceUID}.dcm")
      patientID = slicer.dicomDatabase.instanceValue(instanceUID, '0010,0020')
      outputPath = os.path.join(convertedPath, patientID, study, series, "BatchResult")
      if not os.path.exists(outputPath):
        os.makedirs(outputPath)
      # do an operation here that processes the series into the outputPath
```

Note

It can be helpful for debugging to include a comment with python commands that can be pasted into the console to run the script. With this approach any global variables, such as vtk class instances, defined in the script will exist after the script runs and you can easily inspect them or call methods on them.

```
"""
filePath = "/data/myscript.py"
```

```
exec(open(filePath).read())

"""
```

Extracting volume patches around segments

For machine learning apps, such as MONAI Label it can be helpful to reduce the size of the problem by extracting out subsets of data. This example shows how to iterate through a directory of segmentations, compute their bounding boxes, and save out new volumes and segmentations centered around the segmentation.

Here the ROI is aligned with the volume. See *Segmentations* for examples using oriented bounding boxes and other options.

```
import glob
import os

def segROI(segmentationNode):
  # Compute bounding boxes
  import SegmentStatistics
  segStatLogic = SegmentStatistics.SegmentStatisticsLogic()
  segStatLogic.getParameterNode().SetParameter("Segmentation", segmentationNode.GetID())
  segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪origin_ras.enabled",str(True))
  segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪diameter_mm.enabled",str(True))
  segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_x.enabled",str(True))
  segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_y.enabled",str(True))
  segStatLogic.getParameterNode().SetParameter("LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_z.enabled",str(True))
  segStatLogic.computeStatistics()
  stats = segStatLogic.getStatistics()

  # Draw ROI for each oriented bounding box
  import numpy as np
  for segmentId in stats["SegmentIDs"]:
    # Get bounding box
    obb_origin_ras = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪origin_ras"])
    obb_diameter_mm = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪diameter_mm"])
    obb_direction_ras_x = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_x"])
    obb_direction_ras_y = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_y"])
    obb_direction_ras_z = np.array(stats[segmentId,"LabelmapSegmentStatisticsPlugin.obb_
↪direction_ras_z"])
    # Create ROI
    segment = segmentationNode.GetSegmentation().GetSegment(segmentId)
    roi=slicer.mrmlScene.AddNewNodeByClass("vtkMRMLMarkupsROINode")
    roi.SetName(segment.GetName() + " OBB")
```

(continues on next page)

(continued from previous page)

```
    roi.GetDisplayNode().SetHandlesInteractive(False)  # do not let the user resize the␣
↪box
    roi.SetSize(obb_diameter_mm * 2) # make the ROI twice the size of the segmentation
    # Position and orient ROI using a transform
    obb_center_ras = obb_origin_ras+0.5*(obb_diameter_mm[0] * obb_direction_ras_x + obb_
↪diameter_mm[1] * obb_direction_ras_y + obb_diameter_mm[2] * obb_direction_ras_z)
    boundingBoxToRasTransform = np.row_stack((np.column_stack(((1,0,0), (0,1,0), (0,0,1),
↪ obb_center_ras)), (0, 0, 0, 1)))
    boundingBoxToRasTransformMatrix = slicer.util.
↪vtkMatrixFromArray(boundingBoxToRasTransform)
    roi.SetAndObserveObjectToNodeMatrix(boundingBoxToRasTransformMatrix)
    return roi

labelFiles = glob.glob("/data/imagesTr/labels/final/*.nii.gz")

for labelFile in labelFiles:
  slicer.mrmlScene.Clear()
  print(labelFile)
  baseName = os.path.basename(labelFile)
  ctFile = os.path.join("/data/imagesTr", baseName)
  print(ctFile)
  ct = slicer.util.loadVolume(ctFile)
  seg = slicer.util.loadSegmentation(labelFile)
  roi = segROI(seg)
  cropVolumeParameters = slicer.mrmlScene.AddNewNodeByClass(
↪"vtkMRMLCropVolumeParametersNode")
  cropVolumeParameters.SetInputVolumeNodeID(ct.GetID())
  cropVolumeParameters.SetROINodeID(roi.GetID())
  slicer.modules.cropvolume.logic().Apply(cropVolumeParameters)
  croppedCT = cropVolumeParameters.GetOutputVolumeNode()
  seg.SetReferenceImageGeometryParameterFromVolumeNode(croppedCT)
  segLogic = slicer.modules.segmentations.logic()
  labelmap = slicer.mrmlScene.AddNewNodeByClass("vtkMRMLLabelMapVolumeNode")
  segLogic.ExportAllSegmentsToLabelmapNode(seg, labelmap, slicer.vtkSegmentation.EXTENT_
↪REFERENCE_GEOMETRY)
  slicer.util.saveNode(croppedCT, f"/data/crops/{baseName}")
  slicer.util.saveNode(labelmap, f"/data/crops/labels/final/{baseName}")
  slicer.app.processEvents() # to watch progress
```

12.8.24 WebServer

Create a custom web server instance to serve static content

```
import WebServer
import WebServerLib
# serve content from the temp directory on port 9916 and print all messages
handler = WebServerLib.StaticPagesRequestHandler(docroot=b"/tmp", logMessage=print)
logic = WebServer.WebServerLogic(port=9916, requestHandlers=[handler], logMessage=print)
logic.start()
print(f"Open 'http://localhost:{logic.port}'")
# stop later with logic.stop()
```

Note

Hosting web content is a complex topic, with security implications. Use this feature with caution. It's best to use this only on a trusted network.

Create a custom web server to handle request endpoints

See the WebServerLib for more complete examples.

```
import urllib
import WebServer
import WebServerLib

class ExampleRequestHandler(WebServerLib.BaseRequestHandler):

    def __init__(self, logMessage = None):
        """
        Initialize a new Example request handler instance.
        :param logMessage: An optional external handle for message logging.
        """
        self.logMessage = logMessage

    def canHandleRequest(self, uri: bytes, **_kwargs) -> float:
        """
        Whether the given request is a Example request.
        :param uri: The request URI to parse.
        :return: 0.5 confidence if the request is an Example request, else 0.0
        """
        parsedURL = urllib.parse.urlparse(uri)
        return 0.5 if parsedURL.path.startswith(b"/example") else 0.0

    def handleRequest(self, method: str, uri: bytes, requestBody: bytes, **_kwargs) ->␣
↪tuple[bytes, bytes]:
        """
        Dispatches various example requests.
        :param method: The HTTP request method. 'GET', 'POST', etc.
        :param uri: The request URI to parse.
        :param requestBody: the binary that came with the request
        :return: tuple of content type (based on file ext) and request body binary␣
↪(contents of file)
        """
        parsedURL = urllib.parse.urlparse(uri)
        contentType = b"text/plain"
        responseBody = None
        splitPath = parsedURL.path.split(b"/")
        if len(splitPath) > 2 and splitPath[2] == b"ping":
            self.logMessage("handling collections")
            responseBody = b"pong"
        else:
            self.logMessage("Unhandled Example request path: %s" % parsedURL.path)
            responseBody = b"Unhandled Example request path"

        return contentType, responseBody
```

(continues on next page)

(continued from previous page)

```
# create a server with a custom handler class - here it does nothing, but it
# can access and use anything from the Slicer python environment
PORT = 2042
import WebServer
logMessage = WebServer.WebServerLogic.defaultLogMessage
requestHandlers = [ExampleRequestHandler()]
logic = WebServer.WebServerLogic(port=PORT, logMessage=logMessage, enableSlicer=False,␣
↪enableStaticPages=False, enableDICOM=False, requestHandlers=requestHandlers)

logic.start()
print(f"Open 'http://localhost:{logic.port}/example/ping'")
# stop later with logic.stop()
```

12.9 Build Instructions

12.9.1 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 is also generated in this process.

Users of Slicer application and extensions do not need to build the application and they can download and install pre-built packages instead. Python scripting and development of new Slicer modules in Python does not require building the application either. Only software developers interested in developing Slicer *modules* in C++ language or contributing to the development of Slicer core must build the application.

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.

As Slicer is continuously developed, build instructions may change, too. Therefore, it is recommended to use build instructions that have the same version as the source code.

Extensions for developer builds of Slicer

In general, Slicer versions that a developer builds on his own computer are not expected to work with extensions in the Extensions Server.

Often the Extensions Manager does not show any extensions for a developer build. The reason is that extensions are only built for one Slicer version a day, and so there are many Slicer versions for that no extensions are available.

Even if the Extensions Manager is not empty, the listed extensions are not expected to be compatible with developer builds, created on a different computer, in a different build environment or build options, due to porential ABI incompatibility issues.

If a developer builds the Slicer application then it is expected that the developer will also build all the extension he needs. Building all the extensions after Slicer build is completed *is a simple one-line command*. It is also possible to just *build selected extensions*.

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 the internal, programmatic name of the application.
- `SlicerApp_APPLICATION_DISPLAY_NAME`: Custom application name to be used in display situations. The name is used in user facing text such as window title bar, installation directory name, desktop shortcut names 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_DCMTK_WITH_OPENSSL`: enable/disable building DCMTK 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_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.
- `Slicer_BUILD_WIN32_CONSOLE_LAUNCHER`: Show/hide console (terminal window) on Windows.

Moreoptions are listed in CMake files, such as in SlicerApplicationOptions.cmake and further customization is achievable by using SlicerCustomAppTemplate project maintained by Kitware.

12.9.2 Windows

Note

Slicer relies on a number of large third-party libraries (such VTK, ITK, DCMTK), which take a long time to build and use a lot of disk space. Currently, build requires disk space of 15GB (for release mode) or 60GB (for debug mode). Build time on a desktop computer is typically 3-4 hours, on a laptop it may take 8-12 hours.

Install prerequisites

- CMake version >= 3.20.6.
  - Avoid versions with known Slicer build issues:
    - 3.21.0 (CMake issue 22476)
    - 3.25.0 to 3.25.2 (CMake issues 24180, 24567)

- 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.
- Visual Studio:
  - Download free Community edition (recommended) or a commercial edition of Visual Studio 2022. Visual Studio 2026 with v143 toolset and latest CMake 4.x is usable for building Slicer core and some of its extensions, but it is not an officially supported build configuration yet.
  - Enable `Desktop development with C++` and in installation details
  - Enable the `MSVC v145 - VS2026 C++ x64...` (Visual Studio 2026 v145 toolset with 64-bit support) component - in some distributions, this option is not enabled by default.
  - Enable the `MSVC v143 - VS2022 C++ x64...` (Visual Studio 2022 v143 toolset with 64-bit support) component - in some distributions, this option is not enabled by default.
  - Enable the latest Windows10 SDK component - without this CMake might not find a compiler during configuration step.
- Qt: Download the Qt universal installer and then set up either Qt5 or Qt6 as described below.
  - Qt5 (current stable, officially supported version): Install Qt 5.15.2. In the Select Components tab of the universal installer, click Categories, select Archive, and click Filter to fetch older versions of Qt. Qt version 5.15.2 and its components can be selected by expanding the `Qt` category. Components required: `MSVC2019 64-bit`, `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).
  - Qt6 (experimental): Download the Qt universal installer and run the tool. On the "Customize" page of the Qt Maintenance Tool, expand the "Qt" tree. Then expand the tree matching the latest version of Qt 6 (minimum tested version is Qt-6.9.3, recommended version is the latest supported open-source version). Select `MSVC 2022 64-bit`. Selecting `Sources` and `Qt Debug Information Files` are recommended for debugging (they allow stepping into Qt files with the debugger in debug-mode builds). Expand the "Additional Libraries" tree and select `Qt 5 Compatibility module`, `Qt Multimedia`, `Qt Positioning`, `Qt Shader Tools`, `Qt State Machines`, `Qt WebChannel`. Collapse the "Qt" tree. Expand the "Extensions" tree, then expand "Qt WebEngine" and finally expand the corresponding tree matching the chosen Qt version (e.g. 6.9.3). Select `MSVC 2022 x64`.
  - Note: These are all free, open-source components with LGPL license which allow free usage for any purpose, for any individuals or companies.
- NSIS (optional): Needed if packaging Slicer. Make sure you install the language packs.

Note

Other Visual Studio IDE and compiler toolset versions

- Visual Studio 2019 (v142) toolset is not tested anymore but probably still works.
- Cygwin and Mingw: not tested and not recommended. Building with cygwin gcc not supported, but the cygwin shell environment can be used to run utilities such as git.

Set up source and build folders

- Create source folder. This folder will be referred to as `<Slicer_SOURCE>` in the following. Recommended path: `C:\D\S`
  - 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 following. Recommended path: `C:\D\SR` for release-mode build, `C:\D\SD` 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.
  - How to decide between Debug and Release mode?
    * Release mode build: runs at same speed as official build, requires less disk space (about 15GB); but step-by-step debugging is not available
    * Debug mode build: allows debugging (adding breakpoints, step through the code line by line during execution, watch variables); but it may run 5x or more slower, requires more space (about 60GB), and user interface editing in Qt designer is not available
- 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 .` (note the dot at the end of the command; the `.` is needed because without that git would create a `Slicer` subfolder in the current directory)
- 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

Configure and build Slicer

Build takes several hours. Warnings will appear during the build (it is practically not feasible to have warning-free builds in large multi-platform projects that rely on third-party libraries), but there must not be any errors. If any problems occur, read the *Common errors* section.

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:

```
cd C:\D

"C:\Program Files\CMake\bin\cmake.exe" ^
  -G "Visual Studio 17 2022" -A x64 ^
  -DQt5_DIR:PATH=C:/Qt/5.15.2/msvc2019_64/lib/cmake/Qt5 ^
  -S C:\D\S -B C:\D\SR

"C:\Program Files\CMake\bin\cmake.exe" --build C:\D\SR --config Release
```

Debug mode:

```
cd C:\D

"C:\Program Files\CMake\bin\cmake.exe" ^
  -G "Visual Studio 17 2022" -A x64 ^
  -DQt5_DIR:PATH=C:/Qt/5.15.2/msvc2019_64/lib/cmake/Qt5 ^
  -S C:\D\S -B C:\D\SD

"C:\Program Files\CMake\bin\cmake.exe" --build C:\D\SD --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\5.15.2\msvc2019_64\lib\cmake\Qt5`.
- Click `Configure`
- Select your compiler: `Visual Studio 17 2022`, 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

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).

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`.

Package Slicer (create installer package)

- Start Visual Studio with the launcher:

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

- Select `Release` build configuration.
- In the "Solution Explorer", right click on `PACKAGE` project (in the CMakePredefinedTargets folder) and then select `Build`.

Debug Slicer

- C++ debugging: Visual Studio is recommended on Windows, see *instructions*.
- Python debugging: multiple development environments can be used, see *instructions*.

Common errors

Errors related to Python

Errors due to missing Python libraries (or other Python related errors, such as building a `python-...-requirements` project or Python-wrapping SimpleITK) may be caused by the build system detecting Python installations somewhere on the system, instead of Slicer's own Python environment. To resolve such issues, remove all references to Python in the environment variables (PATH, PYTHONPATH, PYTHONHOME). Alternatively, temporarily rename or remove other Python installations before starting to build Slicer; they can be restored after Slicer build is completed.

Custom Slicer and CTK widgets do not show up in Qt designer

Qt Designer can only use designer plugins (in `c:\D\SR\Slicer-build\bin\designer` and `c:\D\SR\CTK-build\CTK-build\bin\designer`) that are built in the same mode (e.g. `Debug` or `Release`).

In pre-built Qt packages (downloaded from Qt website) Qt Designer is only provided in `Release` mode. If Qt is built from source in `Release` mode (default) or `Release and Debug` mode then Qt designer will be in `Release` mode. In all these cases, Qt Designer can be used only if Slicer is built in `Release` mode.

If Qt is built from source in `Debug` mode mode then Qt designer will be in `Debug` mode. In this case, Qt Designer can be used only if Slicer is built in `Debug` mode.

Other problems

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

12.9.3 macOS

See example *New experimental Recipe for MacOS (Silicon)* below.

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
```

- A CMake version that meets at least the minimum required CMake version here.
- Qt 5: **tested and recommended**.
  - For building Slicer: download and execute qt-online-installer-mac-x64-online.dmg, install Qt 5.15.2, make sure to select the `qtwebengine` component.
  - For packaging and redistributing Slicer: build Qt using qt-easy-build
- Qt 6:
  - For building Slicer: download and execute qt-online-installer-mac-x64-online.dmg, install Qt 6.11.1, make sure to select the `qtwebengine` component.
  - For packaging and redistributing Slicer: build Qt using Build-qt6.sh
- 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.

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 the build process, build folders must be located in a location with very short total path length. This is especially critical on Windows and macOS. For example, `/opt/s` has been confirmed to work on macOS.

Check out the code using `git`:

- Clone the github repository

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

The `Slicer` directory is automatically created after cloning Slicer.

- Setup the development environment:

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

Configure and generate Slicer solution files

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

```
mkdir /opt/s
cd /opt/s
cmake \
  -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=14.0 \
  -DCMAKE_BUILD_TYPE:STRING=Debug \
  -DQt5_DIR:PATH=/path/to/Qt/lib/cmake/Qt5 \
  /path/to/source/code/of/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 `/opt/sr` for Release or `/opt/s` for Debug is recommended. If `/opt` does not exist on your machine you need to use sudo for `mkdir` and `chown` in `/opt`.
  - Step-by-step debug instructions
  - Additional configuration options to customize the application are described *here*.

General information

Two projects are generated by either `cmake`, `ccmake` or `cmake-gui`. One of them is in the top-level bin directory `/opt/s` and the other one is in the subdirectory `Slicer-build`:

- `/opt/s` manages all the external dependencies of Slicer (VTK, ITK, Python, …). To build Slicer for the first time, run `make` in `/opt/s`, which will update and build the external libraries and if successful will then build the subproject `Slicer-build`.
- `/opt/s/Slicer-build` is the "traditional" build directory of Slicer. After local changes in Slicer (or after an git update on the source directory of Slicer), only running `make` in `/opt/s/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 (>20GB)

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

Build Slicer

After configuration, start the build process in the `/opt/s` directory

- Start a terminal and type the following (you can replace 4 by the number of processor cores in the computer. You can find out the number of available cores by running `sysctl -n hw.ncpu`):

```
cd ~/opt/s
make -j4
```

Run Slicer

Start a terminal and type the following:

```
/opt/s/Slicer-build/Slicer
```

Test Slicer

After building, run the tests in the `/opt/s/Slicer-build` directory.

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

```
cd /opt/s/Slicer-build
ctest -j4
```

Package Slicer

Warning: Slicer will **only** create a valid package that will run on machines other than it's built on if Qt was built from source.

Start a terminal and type the following:

```
cd /opt/s
cd Slicer-build
make package
```

Debugging the build process

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. To debug the error you can pipe the output of the make command to an external log file like this:

```
make -j10 -k; make 2>&1 | tee /tmp/build.log
```

In some cases when the build fails without explicitly stating what went wrong it's useful to look at error logs created during building of individual packages bundled with Slicer. Running the following command in the `/opt/s` folder

```
find . -name "*rr*.log" | xargs ls -ltur
```

will list such error logs in ordered by the time of latest access. The log that was accessed the last will be the lowest one in the list.

error while configuring PCRE: "cannot run C compiled program"

If the XCode command line tools are not properly set up on macOS, 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
```

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

Path the 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 `/opt/s` folder. To resolve this error, move the build folder to a location with shorter full path and restart the build from scratch (the build tree is not relocatable).

Packaging errors

Fixing @rpath errors during packaging

If an error like

```
warning: target '@rpath/QtGui.framework/Versions/5/QtGui' is not absolute...
warning: target '@rpath/QtGui.framework/Versions/5/QtGui' does not exist...
```

is present during packaging - doublecheck that Slicer was built with Qt that was built from source and not Qt that was installed from a web-installer or homebrew.

Update Homebrew packages

Slicer can be installed with a single command using Homebrew, as described in the *installation documentation*.

Specifically, the `cask` extension is used, which allows management of graphical applications from the command line. Casks are Ruby files (`.rb`) describing the metadata necessary to download, check and install the package.

The cask for the Preview version of Slicer does not need maintenance, as it automatically downloads the latest binaries and therefore the exact version does not need to be specified in the cask. However, the cask for the stable version, which has traditionally been maintained by Slicer users, needs to be manually updated every time a new stable version of Slicer is released. Instructions to update Homebrew casks can be found on the `homebrew-cask` repository. This is an example of a command that can be used to update the cask for a Stable release:

```
$ brew bump-cask-pr --version 4.11.20210226,1442768 slicer
```

To install `brew`, the following command can be run:

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
↪install.sh)"
```

Common errors

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

–

Recipe for MacOS on arm64 (New, work in progress)

Issues are being tracked here: https://github.com/Slicer/Slicer/issues/6811

Example config:

```
frolick@IsabelMacBook % sw_vers
ProductName:            macOS
ProductVersion:         15.7.3
BuildVersion:           24G419

frolick@IsabelMacBook % qmake --version
QMake version 3.1
Using Qt version 6.10.1 in /opt/homebrew/lib

frolick@IsabelMacBook % cmake --version
cmake version 3.31.5

```

(continues on next page)

(continued from previous page)

```
CMake suite maintained and supported by Kitware (kitware.com/cmake).

frolick@IsabelMacBook % system_profiler SPHardwareDataType

Hardware:
    Hardware Overview:
      Model Name: MacBook Pro
      Model Identifier: Mac15,6
      Chip: Apple M3 Pro
      Total Number of Cores: 11 (5 performance and 6 efficiency)
      Memory: 36 GB
```

Prerequisites

Install Qt6

Verify Qt6 Install

```
qmake --version
```

1. If not installed, Install Qt6 (via Homebrew)

```
brew install qt@6

#Add Qt6 to shell environment
echo 'export PATH="/opt/homebrew/opt/qt@6/bin:$PATH"' >> ~/.zshrc
echo 'export CMAKE_PREFIX_PATH="/opt/homebrew/opt/qt@6:$CMAKE_PREFIX_PATH"' >> ~/.zshrc

#Reload shell
source ~/.zshrc

#Verify Qt6 installation
qmake --version
```

Install Xcode Command Line Tools

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

Install XCode (not strictly necessary (?) but I couldn't do it without XCode Desktop)

If installing full Xcode from the App Store:

```
#Set it as the active developer directory
sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer

#Accept the license
sudo xcodebuild -license accept

#Verify
xcodebuild -version
```

(continues on next page)

(continued from previous page)

```
xcode-select -p
#Should return something like: /Applications/Xcode.app/Contents/Developer
```

Else, check the SDK path used by Xcode:

```
xcrun --show-sdk-path

#Should return something like:
/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/
↪MacOSX.sdk
```

Directory Setup

Create build directories and set ownership:

```
sudo mkdir -p /opt/scmake /opt/scd
sudo chown -R $(whoami) /opt/scmake /opt/scd
```

Create Build Script

```
vim build_slicer.sh
```

Paste into script:

```
#!/bin/bash

# Configuration
SLICER_SOURCE_DIR="$HOME/slicer/latest/Slicer"
SLICER_BUILD_DIR="/opt/scd"
SLICER_SUPERBUILD_DIR="/opt/scmake"

# Create directories
mkdir -p "$SLICER_BUILD_DIR"
mkdir -p "$SLICER_SUPERBUILD_DIR"

# Clone Slicer source if it doesn't exist
if [ ! -d "$SLICER_SOURCE_DIR" ]; then
    echo "Cloning Slicer repository..."
    mkdir -p "$(dirname "$SLICER_SOURCE_DIR")"
    git clone https://github.com/Slicer/Slicer.git "$SLICER_SOURCE_DIR"
fi

cd "$SLICER_SUPERBUILD_DIR"

# Configure with CMake
cmake \
  -DCMAKE_OSX_ARCHITECTURES=arm64 \ #CHANGE ME: OSX architecture (uname -m)
  -DSlicer_REQUIRED_QT_VERSION="6.10" \ # CHANGEME: Qt version (qmake --version)
  -DCMAKE_BUILD_TYPE:STRING=Debug \
```

(continues on next page)

(continued from previous page)

```
  -DSlicer_USE_SimpleITK:BOOL=OFF \
  -DSlicer_BUILD_I18N_SUPPORT:BOOL=OFF \
  -DSlicer_BUILD_DICOM_SUPPORT:BOOL=OFF \
  -DSlicer_VTK_SMP_IMPLEMENTATION_TYPE:STRING=Sequential \
  -DCMAKE_OSX_DEPLOYMENT_TARGET:STRING=15.7 \ # CHANGEME: OSX version (About this Mac ->␣
↪macOS)
  -DCMAKE_OSX_SYSROOT:STRING="$(xcrun --show-sdk-path)" \
  "$SLICER_SOURCE_DIR"

# Build (use -j for parallel jobs, -k to keep going on errors)
make -j10 -k
```

Run Build

```
cd /opt/scmake
chmod +x ./build_slicer.sh
./build_slicer.sh
```

Running Slicer Locally

Option 1: Navigate to the build directory, make, launch:

Executable path: /opt/scmake/Slicer-build/Slicer

```
cd /opt/scmake/Slicer-build
make -j8 #Optional rebuild to show local changes
./Slicer
```

Option 2: Run on XCode with Debugging

Create a Dummy Xcode Project Xcode requires an open project in order to attach a debugger.

1. Open Xcode
2. File → New → Project
3. Create a Command Line Tool project (macOS)
4. Enter an Organization Name (required!)

Get the Slicer Process ID (PID)

1. Open local Slicer through the build directory (following the steps in Option (1)
2. In Python Interactor:

   ```
   import os
   os.getpid()
   ```

   Copy the PID

3. Attach the Debugger In Xcode:

   1. Debug → Attach to Process by Name or PID…

2. Paste the PID
3. Now you can attach process and use debugger functionality (breakpoints, etc.) as usual

12.9.4 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* to adapt the process to your system. You can also ask questions related to the building process in the Slicer forum.

Prerequisites

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

- Git 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*:

Qt 5

- Qt5 with the components listed below. Qt version 5.15.2 is recommended; other Qt versions are not tested and may cause build errors or may cause problems when running the application.
  - Multimedia
  - UiTools
  - XMLPatterns
  - SVG
  - WebEngine
  - Script
  - X11Extras
  - Private
- libXt

Qt 6

- Qt6 with the components listed below. Qt version **6.8 or later** is required.
  - Multimedia
  - UiTools
  - SVG

  - WebEngine
  - Core5Compat
  - Private
- libXt

Debian 12 Bookworm (Stable)

Supported until ~2028.

Install the development tools and the support libraries:

Qt 5

```
sudo apt update && sudo apt install git build-essential cmake cmake-curses-gui cmake-qt-
↪gui \
  qtmultimedia5-dev qttools5-dev libqt5xmlpatterns5-dev libqt5svg5-dev qtwebengine5-dev␣
↪qtscript5-dev \
  qtbase5-private-dev libqt5x11extras5-dev libxt-dev libssl-dev
```

Qt 6

Warning

Debian 12 ships Qt 6.4.2, which is below the officially supported minimum (Qt 6.8). It is possible to build Slicer with Qt 6.4.2, but this is **not officially supported and is used at your own risk**. When configuring, you must explicitly override the Qt version check by adding `-DSlicer_REQUIRED_QT_VERSION=6.4` (see the *Configure section*). For a fully supported Qt6 build, install Qt 6.8 or later from the *Qt online installer* and set `Qt6_DIR` accordingly.

```
sudo apt update && sudo apt install git build-essential cmake-curses-gui cmake-qt-gui \
  qt6-base-dev qt6-base-private-dev qt6-multimedia-dev qt6-tools-dev \
  qt6-svg-dev qt6-5compat-dev qt6-webengine-dev qt6-webengine-dev-tools qt6-scxml-dev \
  libxt-dev libssl-dev
```

Note

The CMake version currently included in Debian 12 Bookworm (Stable) (3.25.1) is not compatible with the current development version of Slicer — Slicer explicitly blocks CMake 3.25.0–3.25.2 due to a known CMake bug. You will need to install CMake manually by downloading CMake version >= 3.25.3 from the CMake releases page and following the installation instructions below.

```
wget https://github.com/Kitware/CMake/releases/download/v3.31.6/cmake-3.31.6-linux-
↪x86_64.sh
chmod +x cmake-3.31.6-linux-x86_64.sh
sudo ./cmake-3.31.6-linux-x86_64.sh --prefix=/usr/local --skip-license --exclude-
↪subdir
```

After installation, verify: `cmake --version` should report `3.31.6`.

Note

On Debian 12 Bookworm (Stable), the included OpenSSL version (3.0.x) requires passing `-DSlicer_USE_SYSTEM_OpenSSL=ON` to the cmake configuration command (see the *Configure section* below).

Ubuntu 25.10 (Questing Quokka)

Standard release, EOL Jul 2026.

Install the development tools and the support libraries:

Qt 5

```
sudo apt update && sudo apt install git git-lfs build-essential \
  libqt5x11extras5-dev qtmultimedia5-dev libqt5svg5-dev qtwebengine5-dev␣
↪libqt5xmlpatterns5-dev qttools5-dev qtbase5-private-dev \
  libxt-dev
```

Qt 6

Ubuntu 25.10 ships Qt 6.9.2, which meets the minimum Qt 6.8 requirement.

```
sudo apt update && sudo apt install git git-lfs build-essential \
  qt6-base-dev qt6-base-private-dev qt6-multimedia-dev qt6-tools-dev qt6-tools-dev-tools␣
↪\
  qt6-svg-dev qt6-5compat-dev qt6-webengine-dev qt6-webengine-dev-tools qt6-scxml-dev \
  libxt-dev libssl-dev
```

Note

Ubuntu 25.10 ships GCC 15, which defaults to the C23 standard. This causes a build failure in ITK's MINC library due to old-style function pointer declarations incompatible with C23. Add `-DADDITIONAL_C_FLAGS=-std=gnu11` to the cmake configure command to work around this (see *Configure section* below).

Note

The CMake version currently included in Ubuntu 25.10 is CMake 3.31.6 which is compatible with the current development version of Slicer. **Last time tested: 2026-03-26.**

Ubuntu 24.04 (Noble Numbat)

LTS, supported until Apr 2029.

Install the development tools and the support libraries:

Qt 5

```
sudo apt update && sudo apt install git git-lfs build-essential \
  libqt5x11extras5-dev qtmultimedia5-dev libqt5svg5-dev qtwebengine5-dev␣
↪libqt5xmlpatterns5-dev qttools5-dev qtbase5-private-dev \
  libxt-dev
```

Qt 6

> **Warning**
>
> Ubuntu 24.04 ships Qt 6.4.2, which is below the officially supported minimum (Qt 6.8). It is possible to build Slicer with Qt 6.4.2, but this is **not officially supported and is used at your own risk**. When configuring, you must explicitly override the Qt version check by adding `-DSlicer_REQUIRED_QT_VERSION=6.4` (see the *Configure section*). For a fully supported Qt6 build, install Qt 6.8 or later from the *Qt online installer* and set `Qt6_DIR` accordingly.

```
sudo apt update && sudo apt install git git-lfs build-essential \
  qt6-base-dev qt6-base-private-dev qt6-multimedia-dev qt6-tools-dev \
  qt6-svg-dev qt6-5compat-dev qt6-webengine-dev qt6-webengine-dev-tools qt6-scxml-dev \
  libxt-dev libssl-dev
```

> **Note**
>
> The CMake version currently included in Ubuntu 24.04 is CMake 3.28.3 which is compatible with the current development version of Slicer. **Last time tested: 2026-03-20.**

Ubuntu 22.04 (Jammy Jellyfish)

LTS, supported until Apr 2027.

Install the development tools and the support libraries:

Qt 5

```
sudo apt update && sudo apt install git build-essential cmake cmake-curses-gui cmake-qt-
↪gui \
  libqt5x11extras5-dev qtmultimedia5-dev libqt5svg5-dev qtwebengine5-dev␣
↪libqt5xmlpatterns5-dev qttools5-dev qtbase5-private-dev \
  libxt-dev libssl-dev
```

Qt 6

> **Warning**
>
> Ubuntu 22.04 ships Qt 6.2.4, which is below the officially supported minimum (Qt 6.8). Qt6 builds on Ubuntu 22.04 require installing Qt 6.8 or later from the *Qt online installer*.
>
> Install the system dependencies first:

```
sudo apt update && sudo apt install git build-essential cmake cmake-curses-gui cmake-
↪qt-gui \
  libxt-dev libssl-dev
```

Then follow the *Any Distribution* section to install Qt 6.8 or later, and configure the build with `-DQt6_DIR` pointing to the installer location.

ArchLinux

Warning

ArchLinux uses a rolling-release package distribution approach. This means that the versions of the packages will change over time and the following instructions might not be actual. **Last time tested: 2024-01-01.**

You could build Slicer using the `PKGBUILD` from AUR: 3dslicer and 3dslicer-git.

Any Distribution

This section describes how to install Qt as distributed by *The QT Company*, which can be used for any GNU/Linux distribution.

Important

This process requires an account in qt.io

Download the Qt Linux online installer and make it executable:

```
curl -LO https://download.qt.io/official_releases/online_installers/qt-online-installer-
↪linux-x64-online.run
chmod +x qt-online-installer-linux-x64-online.run
```

Qt 5

You can run the installer and follow the instructions in the GUI. Keep in mind that the components needed by 3D Slicer are: `qt.qt5.5152.gcc_64`, `qt.qt5.5152.qtwebengine` and `qt.qt5.5152.qtwebengine.gcc_64`.

Alternatively, you can request the installation of the components with the following command (you will be prompted for license agreements and permissions):

```
export QT_ACCOUNT_LOGIN=<set your qt.io account email here>
export QT_ACCOUNT_PASSWORD=<set your password here>
./qt-online-installer-linux-x64-online.run \
  install \
    qt.qt5.5152.gcc_64 \
    qt.qt5.5152.qtwebengine \
    qt.qt5.5152.qtwebengine.gcc_64 \
  --root /opt/qt \
  --email $QT_ACCOUNT_LOGIN \
  --pw $QT_ACCOUNT_PASSWORD
```

Hint

When configuring the Slicer build project, set the CMake variable `Qt5_DIR` to the full path of the Qt5 installation directory ending with `5.15.2/gcc_64/lib/cmake/Qt5`. For example, assuming Qt is installed in `/opt/qt`:

```
cmake -DCMAKE_BUILD_TYPE:STRING=Release -DQt5_DIR:PATH=/opt/qt/5.15.2/gcc_64/lib/
↪cmake/Qt5 ../Slicer
```

Qt 6

You can run the installer and follow the instructions in the GUI. The components needed by 3D Slicer include the base Qt6 libraries, Qt WebEngine, and Qt5Compat. Select Qt version 6.8 or later.

Alternatively, you can request the installation of the components with the following command (you will be prompted for license agreements and permissions):

```
export QT_ACCOUNT_LOGIN=<set your qt.io account email here>
export QT_ACCOUNT_PASSWORD=<set your password here>
./qt-online-installer-linux-x64-online.run \
  install \
    qt.qt6.680.gcc_64 \
    qt.qt6.680.addons.qtwebengine \
    qt.qt6.680.addons.qt5compat \
  --root /opt/qt \
  --email $QT_ACCOUNT_LOGIN \
  --pw $QT_ACCOUNT_PASSWORD
```

Hint

When configuring the Slicer build project, set the CMake variable `Qt6_DIR` to the full path of the Qt6 installation directory ending with `6.8.0/gcc_64/lib/cmake/Qt6`. For example, assuming Qt is installed in `/opt/qt`:

```
cmake \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DSlicer_REQUIRED_QT_VERSION=6.4 \
  -DQt6_DIR:PATH=/opt/qt/6.8.0/gcc_64/lib/cmake/Qt6 \
  ../Slicer
```

Checkout Slicer source files

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

```
git clone https://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`.

Tip

It is highly recommended to **avoid** the use of the **space** character in the name of the `source directory` or any of its parent directories.

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

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

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 behavior 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.

The following folders will be used in the instructions below:

To obtain a default configuration of the Slicer build project, create the **build** folder and use `cmake`:

```
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:

```
cmake -DCMAKE_BUILD_TYPE:STRING=Release ../Slicer
```

Qt 5

If Qt5 was installed from the *Qt Company online installer*, the CMake variable `Qt5_DIR` must be set:

```
cmake -DCMAKE_BUILD_TYPE:STRING=Release -DQt5_DIR:PATH=/opt/qt/5.15.2/gcc_64/lib/cmake/
↪Qt5 ../Slicer
```

Note

On Ubuntu 25.10 (GCC 15), add `-DADDITIONAL_C_FLAGS=-std=gnu11` to work around a C23 incompatibility in ITK's MINC library:

```
cmake -DCMAKE_BUILD_TYPE:STRING=Release -DADDITIONAL_C_FLAGS=-std=gnu11 ../Slicer
```

Qt 6

To build with Qt6, set `-DQt6_DIR` pointing to the Qt6 cmake directory. Slicer will automatically enforce the minimum required Qt6 version (6.8). When using distribution-provided Qt6 packages (e.g. Ubuntu 25.10), `Qt6_DIR` is typically `/usr/lib/x86_64-linux-gnu/cmake/Qt6`:

```
cmake \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DQt6_DIR:PATH=/usr/lib/x86_64-linux-gnu/cmake/Qt6 \
  ../Slicer
```

Note

On Ubuntu 25.10 (GCC 15), add `-DADDITIONAL_C_FLAGS=-std=gnu11` to work around a C23 incompatibility in ITK's MINC library:

```
cmake \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DADDITIONAL_C_FLAGS=-std=gnu11 \
  -DQt6_DIR:PATH=/usr/lib/x86_64-linux-gnu/cmake/Qt6 \
  ../Slicer
```

If Qt6 was installed from the *Qt Company online installer*, set `Qt6_DIR` to the installer location instead:

```
cmake \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DQt6_DIR:PATH=/opt/qt/6.8.0/gcc_64/lib/cmake/Qt6 \
  ../Slicer
```

Warning

On distributions shipping Qt6 < 6.8 (e.g. Debian 12, Ubuntu 24.04), the cmake configure step will fail with a version error. To bypass the check and build anyway — **at your own risk and without official support** — explicitly set `Slicer_REQUIRED_QT_VERSION=6.4`:

```
cmake \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DSlicer_REQUIRED_QT_VERSION=6.4 \
  -DQt6_DIR:PATH=/usr/lib/x86_64-linux-gnu/cmake/Qt6 \
  ../Slicer
```

Warning

On Debian 12 Bookworm (Stable), the included OpenSSL version (3.0.x) is not compatible with the OpenSSL versions (1.0 - 1.1) used in Slicer, and attempting to run Slicer will emit the following warning, indicating that SSL support is disabled:

```
qt.network.ssl: Incompatible version of OpenSSL (built with OpenSSL >= 3.x, runtime version is < 3.x)
[SSL] SSL support disabled - Failed to load SSL library !
[SSL] Failed to load Slicer.crt
QSslSocket::connectToHostEncrypted: TLS initialization failed
```

To enable SSL, one can use the system OpenSSL as follows:

```
cmake -DSlicer_USE_SYSTEM_OpenSSL=ON ../Slicer
```

Tip – Interfaces to change 3D Slicer configuration variables

Instead of `cmake`, one can use `ccmake`, which provides a text-based interface, or `cmake-gui`, which provides a graphical user interface. These applications will also provide a list of variables that can be changed.

Tip – Speed up 3D Slicer build with ccache

`ccache` is a compiler cache that can speed up subsequent builds of 3D Slicer. This can be useful if 3D Slicer is built often and there are no large divergences between subsequent builds. This requires `ccache` installed on the system (e.g., `sudo apt install ccache`).

The first time `ccache` is used, the compilation time can marginally increase as it includes the first caching. After the first build, subsequent build times will decrease significantly.

`ccache` is not detected as a valid compiler by the 3D Slicer building process. You can generate local symbolic links to disguise the use of `ccache` as valid compilers:

```
ln -s /usr/bin/ccache ~/.local/bin/c++
ln -s /usr/bin/ccache ~/.local/bin/cc
```

Then, the Slicer build can be configured to use these compilers:

```
cmake \
  -DCMAKE_BUILD_TYPE:STRING=Release \
  -DCMAKE_CXX_COMPILER:STRING=$HOME/.local/bin/c++ \
  -DCMAKE_C_COMPILER:STRING=$HOME/.local/bin/cc \
  ../Slicer
```

Build Slicer

Once the Slicer build project files have been generated, the Slicer project can be built by running this command in the **build** folder

```
make
```

Tip – Parallel build

Building Slicer will generally take a long time, particularly on the first build or upon code/configuration changes. To help speed up the process, one can use `make -j<N>`, where <N> is the number of parallel builds. As a rule of thumb, many use the `number of CPU threads - 1` as the number of parallel builds.

Warning

Increasing the number of parallel builds generally increases the memory required for the build process. In the event that the required memory exceeds the available memory, the process will either fail or start using swap memory, which may make the system freeze.

> **Tip – Error detection during parallel build**
>
> Using parallel builds makes finding compilation errors difficult due to the fact that all parallel build processes use the same screen output, as opposed to sequential builds, where the compilation process will stop at the error. A common technique to have parallel builds and easily find errors is to launch a parallel build followed by a sequential build. For the parallel build, it is advised to run `make -j<N> -k` to have the parallel build keep going as far as possible before doing the sequential build with `make`.

Run Slicer

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

The application can be launched by these commands:

```
cd Slicer-build
./Slicer`
```

Test Slicer

After building, run the tests in the **inner-build** folder.

Type the following (you can replace 4 with the number of processor cores in the computer):

```
ctest -j4
```

Package Slicer

Start a terminal and type the following in the **inner-build** folder:

```
make package
```

Common errors

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

12.9.5 Common errors

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.

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.

'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.

12.9.6 Other errors

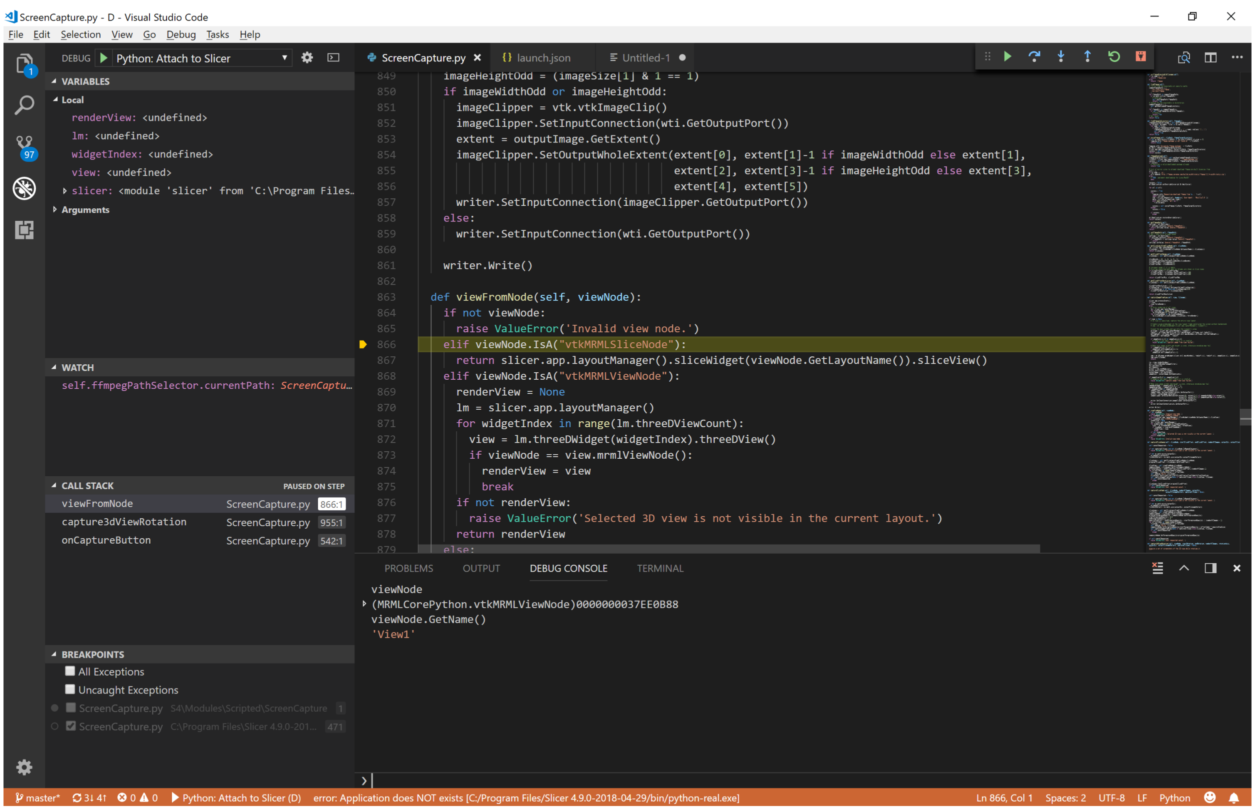
If you encounter any other error then you can ask help on the Slicer forum. Post your question in `Developer` category, use the `build` tag.

Usually seeing the full build log is needed for investigating the issue. The full build log can be found in the build tree, its exact location depend on the operating system and how Slicer was built, so the easiest way to find it is to look for large `*.log` files in the build tree. For example, you may find the full build log in `<Slicer_BUILD>\Testing\Temporary\LastBuild_<date>-<time>.log`. Large amount of text cannot be included in forum posts, therefore it is recommended to upload the build log to somewhere (Dropbox, OneDrive, Google drive, etc.) and add the download link to the forum post.

12.10 Debugging

12.10.1 Python debugging

Python code running in Slicer can be debugged (execute code line-by-line, inspect variables, browse the call stack, etc.) by attaching a debugger to the running Slicer application. Detailed instructions are provided in documentation of DebuggingTools extension.



Profiling

The following code snippet can be used to get detailed performance analysis Python code:

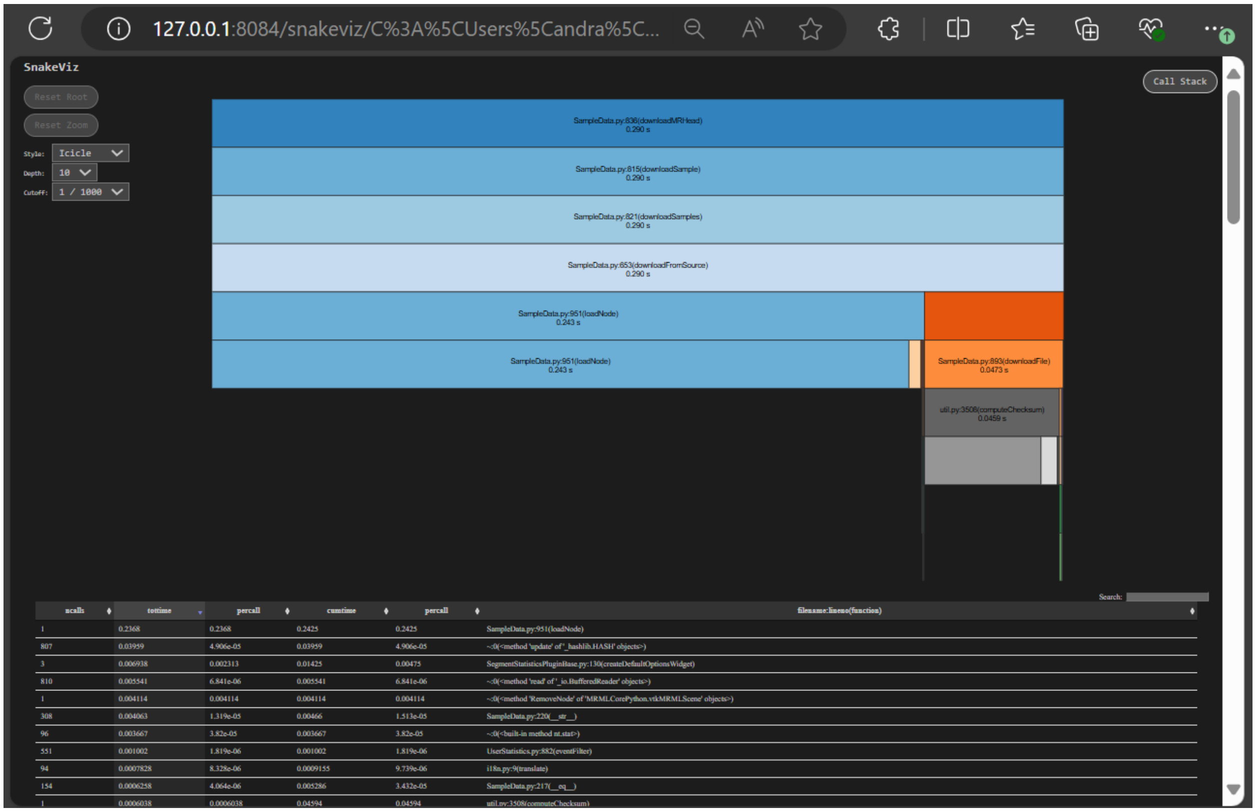
```
# Start profiling
detailedProfiling = True  # set to False to just get elapsed time
if detailedProfiling:
    import cProfile
```

(continues on next page)

(continued from previous page)

```
    cp=cProfile.Profile()
    cp.enable()
else:
    startTime = time.time()

# ---------------------------------------------
# Put the tested code here, for example:
import SampleData
sampleDataLogic = SampleData.SampleDataLogic()
mrHead = sampleDataLogic.downloadMRHead()
import SegmentStatistics
segStatLogic = SegmentStatistics.SegmentStatisticsLogic()
# ---------------------------------------------

# Finish profiling and display result
if detailedProfiling:
    profileFilePath = os.path.join(slicer.app.temporaryPath, 'profile.prof')
    cp.disable()
    cp.dump_stats(profileFilePath)
    # Import visualization tool
    try:
      import snakeviz
    except ImportError:
      pip_install("snakeviz")
    # Show profiling results in the web browser
    import shutil
    pythonSlicerExecutablePath = shutil.which("PythonSlicer")
    commandLine = [pythonSlicerExecutablePath, "-m", "snakeviz", profileFilePath]
    proc = slicer.util.launchConsoleProcess(commandLine, useStartupEnvironment=False)
    # proc.kill()
else:
    stopTime = time.time()
    print(f"Elapsed time: {stopTime-startTime} s")
```

Detailed profiling result is shown in the default web browser:



12.10.2 C++ debugging

Debugging C++ code requires building 3D Slicer in Debug mode by following the *build instructions*.

The executable Slicer application (`Slicer` or `Slicer.exe`) is the launcher of the real application binary (`SlicerApp-real`). The launcher sets up paths for dynamically-loaded libraries that on Windows and Linux are required to run the real application library.

C++ debugging on Windows

Debugging using Visual Studio

Prerequisites:

- Build 3D Slicer in Debug by following the *build instructions*.

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>/Slicer-build folder:

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

> **Note**
>
> - 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:

```
.\SD\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
```

2. 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`

3. 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.

Debugging tests

Once VisualStudio is open with the Slicer environment loaded, it is possible to run and debug tests. See general information about running tests *here*. Specific instructions for Visual Studio:

- To debug a test, find its project in the Solution explorer tree.
- Make the project the startup project (right-click -> Set As Startup Project).
- Specify test name and additional input arguments:
  - 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`) followed by additional arguments (if any).
- Start debugging by choosing `Start debugging` in Edit menu.

Tip

To run all tests, build the `RUN_TESTS` project.

Debugging using cross-platform IDEs

On Windows, Visual Studio is the most feature-rich and powerful debugger, but in case somebody wants to use cross-platform tools then these instructions may help:

- *Debugging using Qt Creator*
- *Debugging using Visual Studio Code*

C++ debugging on GNU/Linux systems

GDB debug by attaching to running process (recommended)

1. Starting with Ubuntu 10.10, ptracing of non-child processes by non-root users as been disabled -ie. only a process which is a parent of another process can ptrace it for normal users. More details here.

   - You can temporarily disable this restriction by:

     ```
     $ echo 0 | sudo tee /proc/sys/kernel/yama/ptrace_scope
     ```

   - To permanently allow it to edit `/etc/sysctl.d/10-ptrace.conf` and change the line:

     ```
     kernel.yama.ptrace_scope = 1
     ```

     to read:

     ```
     kernel.yama.ptrace_scope = 0
     ```

2. Running Slicer with the following command line argument will allow you to easily obtain the associated PID:

   ```
   $ ./Slicer --attach-process
   ```

   This will bring up a window with the `PID` before loading any modules, which is also helpful for debugging the loading process.

3. Then, you can attach the process to `gdb` using the following command:

   ```
   $ gdb --pid $PIDABOVE
   ```

4. Finally type the following gdb command

   ```
   (gdb) continue
   ```

   If not using the `--attach-process`, the `PID` could be obtain using the following command:

   ```
   $ ps -Afww | grep SlicerApp-real
   ```

> **Tip**
>
> How to print QString using GDB?
>
> See http://silmor.de/qtstuff.printqstring.php.

GDB debug with launch arguments

The Slicer app launcher provides options to start other programs with the Slicer environment settings.

- `--launch <executable> [<parameters>]`: executes an arbitrary program. For example, `Slicer --launch /usr/bin/gnome-terminal` starts gnome-terminal (then run GDB directly on SlicerApp-real)
- `--gdb`: runs GDB then executes SlicerApp-real from within the debugger environment.

GDB debug by following the forked process

```
gdb Slicer
```

gdb should warn you that there are no debug symbols for Slicer, which is true because Slicer is the launcher. Now we need to set gdb to follow the forked process SlicerApp-real and run the launcher:

```
(gdb) set follow-fork-mode child
(gdb) run
```

gdb will run Slicer and attach itself to the forked process SlicerApp-real

GDB debug by using exec-wrapper

An alternative approach is to use a wrapper script to emulate the functionality of the app launcher. This will allow you to use gdb or a gdb-controlling program such as an IDE, in order to interactively debug directly from GDB without attaching.

The general idea of the wrapper is to set all of the appropriate environment variables as they would be set by the app launcher. From SlicerLauncherSettings:

- `[LibraryPath]` contents should be in `LD_LIBRARY_PATH`
- `[Paths]` contents should be in `PATH`
- `[EnvironmentVariables]` should each be set

Now, start gdb and do the following:

```
(gdb) set exec-wrapper ./WrapSlicer
(gdb) exec-file ./bin/SlicerQT-real
(gdb) run
```

Since VTK and ITK include many multithreaded filters, by default you will see lots of messages like the following from gdb during rendering and processing:

```
[New Thread 0x7fff8378f700 (LWP 20510)]
[Thread 0x7fff8b0aa700 (LWP 20506) exited]
```

These can be turned off with this command:

```
set print thread-events off
```

Example wrapper script

To make a wrapper script, run

```
Slicer --launch `which gnome-terminal`
```

or

```
Slicer --gnome-terminal
```

and check the value of `LD_LIBRARY_PATH` (note, it doesn't seem to be set using xterm). Then check the other environment variables as listed in `[EnvironmentVariables]`.

See the examples below.

SlicerLaunchSettings.ini

```
[General]
launcherSplashImagePath=/cmn/git/Slicer4/Applications/SlicerQT/Resources/Images/
↪SlicerSplashScreen.png
launcherSplashScreenHideDelayMs=3000
additionalLauncherHelpShortArgument=-h
additionalLauncherHelpLongArgument=--help
additionalLauncherNoSplashArguments=--no-splash,--help,--version,--home,--program-path,--
↪no-main-window

[Application]
path=<APPLAUNCHER_DIR>/bin/./SlicerQT-real
arguments=

[ExtraApplicationToLaunch]

designer/shortArgument=
designer/help=Start Qt designer using Slicer plugins
designer/path=/usr/bin/designer-qt4
designer/arguments=

gnome-terminal/shortArgument=
gnome-terminal/help=Start gnome-terminal
gnome-terminal/path=/usr/bin/gnome-terminal
gnome-terminal/arguments=

xterm/shortArgument=
xterm/help=Start xterm
xterm/path=/usr/bin/xterm
xterm/arguments=

ddd/shortArgument=
ddd/help=Start ddd
ddd/path=/usr/bin/ddd
ddd/arguments=

gdb/shortArgument=
gdb/help=Start gdb
gdb/path=/usr/bin/gdb
gdb/arguments=


[LibraryPaths]
1\path=/cmn/git/Slicer4-sb/VTK-build/bin/.
2\path=/cmn/git/Slicer4-sb/CTK-build/CTK-build/bin/.
3\path=/usr/lib
4\path=/cmn/git/Slicer4-sb/ITKv3-build/bin/.
5\path=/cmn/git/Slicer4-sb/SlicerExecutionModel-build/ModuleDescriptionParser/bin/.
6\path=/cmn/git/Slicer4-sb/teem-build/bin/.
7\path=/cmn/git/Slicer4-sb/LibArchive-install/lib
8\path=<APPLAUNCHER_DIR>/bin/.
9\path=../lib/Slicer-4.0/qt-loadable-modules
```

(continues on next page)

(continued from previous page)

```
10\path=<APPLAUNCHER_DIR>/lib/Slicer-4.0/cli-modules/.
11\path=<APPLAUNCHER_DIR>/lib/Slicer-4.0/qt-loadable-modules/.
12\path=/cmn/git/Slicer4-sb/tcl-build/lib
13\path=/cmn/git/Slicer4-sb/OpenIGTLink-build
14\path=/cmn/git/Slicer4-sb/OpenIGTLink-build/bin/.
15\path=/cmn/git/Slicer4-sb/CTK-build/PythonQt-build/.
16\path=/cmn/git/Slicer4-sb/python-build/lib
17\path=/cmn/git/Slicer4-sb/python-build/lib/python2.6/site-packages/numpy/core
18\path=/cmn/git/Slicer4-sb/python-build/lib/python2.6/site-packages/numpy/lib
size=18

[Paths]
1\path=<APPLAUNCHER_DIR>/bin/.
2\path=/cmn/git/Slicer4-sb/teem-build/bin/.
3\path=/usr/bin
4\path=<APPLAUNCHER_DIR>/lib/Slicer-4.0/cli-modules/.
5\path=/cmn/git/Slicer4-sb/tcl-build/bin
size=5

[EnvironmentVariables]
QT_PLUGIN_PATH=<APPLAUNCHER_DIR>/bin<PATHSEP>/cmn/git/Slicer4-sb/CTK-build/CTK-build/bin
↪<PATHSEP>/usr/lib/qt4/plugins
SLICER_HOME=/cmn/git/Slicer4-sb/Slicer-build
PYTHONHOME=/cmn/git/Slicer4-sb/python-build
PYTHONPATH=<APPLAUNCHER_DIR>/bin<PATHSEP><APPLAUNCHER_DIR>/bin/Python<PATHSEP>/cmn/git/
↪Slicer4-sb/python-build/lib/python2.6/site-packages<PATHSEP><APPLAUNCHER_DIR>/lib/
↪Slicer-4.0/qt-loadable-modules/.<PATHSEP><APPLAUNCHER_DIR>/lib/Slicer-4.0/qt-loadable-
↪modules/Python
TCL_LIBRARY=/cmn/git/Slicer4-sb/tcl-build/lib/tcl8.4
TK_LIBRARY=/cmn/git/Slicer4-sb/tcl-build/lib/tk8.4
TCLLIBPATH=/cmn/git/Slicer4-sb/tcl-build/lib/itcl3.2 /cmn/git/Slicer4-sb/tcl-build/lib/
↪itk3.2
```

It's easier to just check `LD_LIBRARY_PATH` using `--launch`. If this is somehow not available, the above settings translate directly to:

```
#!/bin/bash
BASE_DIR=/cmn/git/Slicer4-sb/
APPLAUNCHER_DIR=$BASE_DIR/Slicer-build

LD_PATHS="
/cmn/git/Slicer4-sb/VTK-build/bin/.
/cmn/git/Slicer4-sb/CTK-build/CTK-build/bin/.
/usr/lib
/cmn/git/Slicer4-sb/ITKv3-build/bin/.
/cmn/git/Slicer4-sb/SlicerExecutionModel-build/ModuleDescriptionParser/bin/.
/cmn/git/Slicer4-sb/teem-build/bin/.
/cmn/git/Slicer4-sb/LibArchive-install/lib
$APPLAUNCHER_DIR/bin/.
../lib/Slicer-4.0/qt-loadable-modules
$APPLAUNCHER_DIR/lib/Slicer-4.0/cli-modules/.
$APPLAUNCHER_DIR/lib/Slicer-4.0/qt-loadable-modules/.
```

(continues on next page)

(continued from previous page)

```
/cmn/git/Slicer4-sb/tcl-build/lib
/cmn/git/Slicer4-sb/OpenIGTLink-build
/cmn/git/Slicer4-sb/OpenIGTLink-build/bin/.
/cmn/git/Slicer4-sb/CTK-build/PythonQt-build/.
/cmn/git/Slicer4-sb/python-build/lib
/cmn/git/Slicer4-sb/python-build/lib/python2.6/site-packages/numpy/core
/cmn/git/Slicer4-sb/python-build/lib/python2.6/site-packages/numpy/lib
"
for STR in $LD_PATHS; do LD_LIBRARY_PATH="${STR}:${LD_LIBRARY_PATH}"; done

QT_PLUGIN_PATH=$APPLAUNCHER_DIR/bin:/cmn/git/Slicer4-sb/CTK-build/CTK-build/bin:/usr/lib/
↪qt4/plugins
SLICER_HOME=/cmn/git/Slicer4-sb/Slicer-build
PYTHONHOME=/cmn/git/Slicer4-sb/python-build
PYTHONPATH=$APPLAUNCHER_DIR:/bin:$APPLAUNCHER_DIR:/bin/Python:/cmn/git/Slicer4-sb/python-
↪build/lib/python2.6/site-packages:$APPLAUNCHER_DIR/lib/Slicer-4.0/qt-loadable-modules/.
↪:$APPLAUNCHER_DIR/lib/Slicer-4.0/qt-loadable-modules/Python
TCL_LIBRARY=/cmn/git/Slicer4-sb/tcl-build/lib/tcl8.4
TK_LIBRARY=/cmn/git/Slicer4-sb/tcl-build/lib/tk8.4
TCLLIBPATH=/cmn/git/Slicer4-sb/tcl-build/lib/itcl3.2:/cmn/git/Slicer4-sb/tcl-build/lib/
↪itk3.2

export QTPLUGIN_PATH=$QT_PLUGIN_PATH
export SLICER_HOME=$SLICER_HOME
export PYTHONHOME=$PYTHONHOME
export PYTHONPATH=$PYTHONPATH
export TCL_LIBRARY=$TCL_LIBRARY
export TK_LIBRARY=$TK_LIBRARY
export TCLLIBPATH=$TCLLIBPATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH

exec "$@"
```

Analyze a segmentation fault

In the build tree:

```
$ ulimit -c unlimited
$ ./Slicer
... make it crash
$ ./Slicer --gdb ./bin/SlicerApp-real
(gdb) core core
(gdb) backtrace
...
```

For an installed Slicer:

```
$ ulimit -c unlimited
$ ./Slicer
... make it crash
$ ./Slicer --launch bash
$ gdb ./bin/SlicerApp-real
```

(continues on next page)

(continued from previous page)

```
(gdb) core core
(gdb) backtrace
...
```

Note

GDB requires Python. However, Python that is linked into GDB is not the same as Slicer's Python, which may cause issues. If GDB does not start because `_sysconfigdata__linux_x86_64-linux-gnu.py` file is missing then Slicer's sysconfigdata file must be copied to the expected filename. For example:

```
cd ~/Slicer-4.13.0-2021-09-10-linux-amd64/lib/Python/lib/python3.6
cp _sysconfigdata_m_linux2_.py _sysconfigdata__linux_x86_64-linux-gnu.py
```

With systemd

In linux distros with systemd, coredumps are managed by the systemd daemon. And stored, in a compressed format (.lz4), in

```
/var/lib/systemd/coredump/core.SlicerApp-real.xxxx.lz4
```

It can happen that even with ulimit -c unlimited, the coredump files are still truncated. You can check latest coredumps, and the corresponding PID with:

```
coredumpctl list
Thu 2018-05-24 16:37:46 EDT   22544  1000  1000  11 missing   /usr/lib/firefox/firefox
Fri 2018-05-25 15:50:52 EDT   14721  1000  1000   6 truncated /path/Slicer-build/bin/
↪SlicerApp-real
Mon 2018-05-28 11:35:43 EDT   17249  1000  1000   6 present   /path/Slicer-build/bin/
↪SlicerApp-real
```

You can modify systemd coredump to increase the default max file size in `/etc/systemd/coredump.conf`:

```
[Coredump]
#Storage=external
#Compress=yes
ProcessSizeMax=8G
ExternalSizeMax=8G
JournalSizeMax=6G
#MaxUse=
#KeepFree=
```

After that change, make Slicer crash again, and the core file will be present, instead of truncated.

You can launch gdb with a coredump file with the command:

```
coredumpctl gdb $PID
```

If no $PID is provided, the latest coredump is used. PID can be retrieved using `coredumpctl list`.

To decompress the coredump file to use with other IDE or ddd, change the coredump.conf Compress option, or use:

```
coredumpctl dump $PID > /tmp/corefile
```

Debugging using cross-platform IDEs

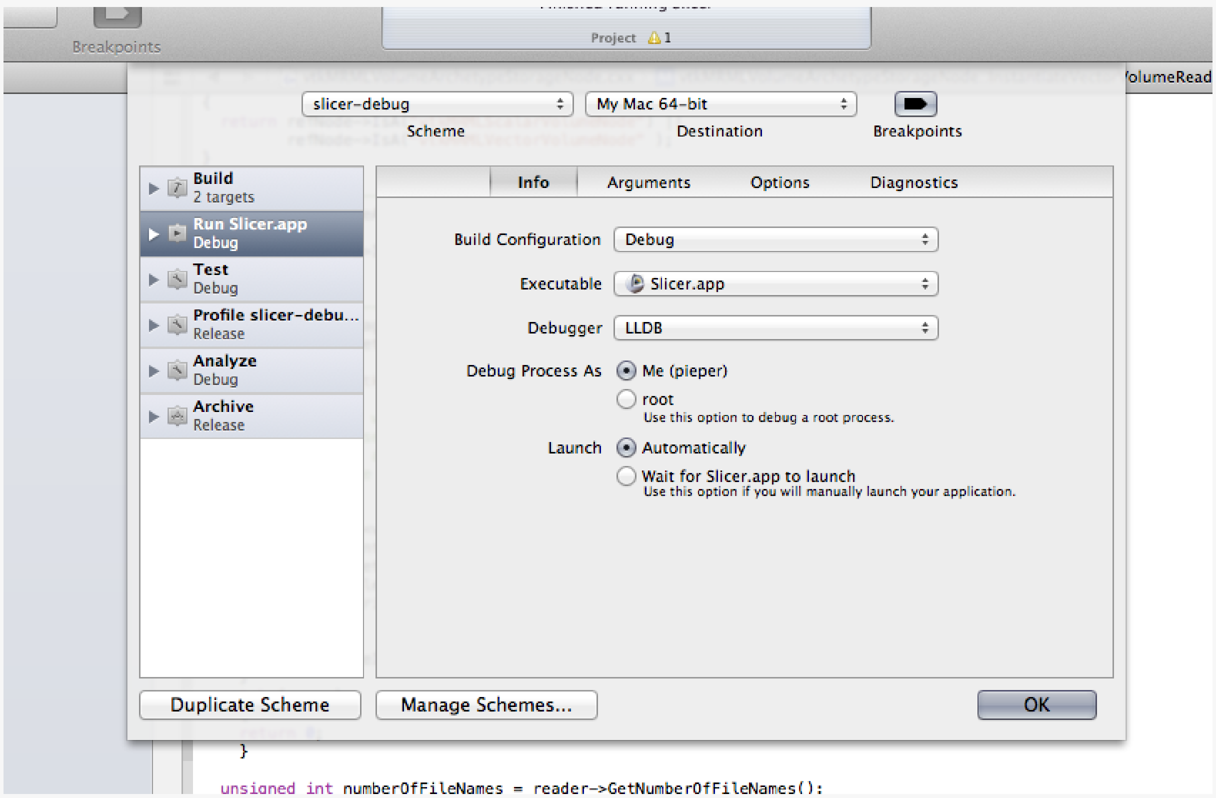
- *Debugging using CodeLite*
- *Debugging using Qt Creator*
- *Debugging using Visual Studio Code*

C++ debugging on macOS

Debugging using Xcode

We do not support building using Xcode, however if you build Slicer using traditional unix Makefiles you can still debug using the powerful source debugging features of Xcode.

> **Note**
>
> You need to set up a dummy project in Xcode just to enable some of the options. This means that Xcode will create a macOS code template, but you won't use it directly (instead you will point to the code you build with cmake/make).



Attaching to a running process

You can use the Attach to Process option in the Xcode Product menu for debugging. This will allow you to get browsable stack traces for crashes.

Steps:

- Start Slicer
- Start Xcode

- Use the Product->Attach to Process… menu

Setting Breakpoints

To set a breakpoint in code that is not crashing, you can set it via the command line interface. For the lldb debugger, first attach to the process and break the program. Then at the (lldb) prompt, stop at a method as follows:

```
breakpoint set -M vtkMRMLScene::Clear
```

then you can single step and/or set other breakpoints in that file.

Alternatively, you can use the Product->Debug->Create a Symbolic Breakpoint… option. This is generally preferable to the lldb level debugging, since it shows up in the Breakpoints pane and can be toggled/deleted with the GUI.

Debugging the Startup Process

To debug startup issues you can set up a *Scheme* and select the `Slicer.app` in your `Slicer-build` directory (see screenshot). You can set up multiple schemes with different command line arguments and executables (for example to debug tests).

Since you are using a dummy project but want to debug the Slicer application, you need to do the following:

- Start Xcode and open dummy project
- Pick Product->Edit Scheme…
- Select the Run section
- Select the Info tab
- Pick Slicer.app as the Executable
- Click OK
- Now the Run button will start Slicer for debugging

Tip

You can create multiple schemes, each various command line options to pass to Slicer so you can easily get back to a debugging environment.

Profiling

The Instruments tool is very useful for profiling.

Debugging using cross-platform IDEs

On Windows, Visual Studio is the most feature-rich and powerful debugger, but in case somebody wants to use cross-platform tools then these instructions may help:

- *Debugging using Qt Creator*
- *Debugging using Visual Studio Code*

C++ debugging with Visual Studio Code

Visual Studio Code is a cross-platform IDE that can be used for debugging C++ and Python code.

> **Tip**
>
> For debugging on Windows, Visual Studio is recommended as it has much better performance and many more features than Visual Studio Code.

Prerequisites

- Build 3D Slicer in Debug mode, outside of Visual Studio Code, by following the *build instructions*.
- Install Visual Studio Code
- Install the C/C++ extension in Visual Studio Code

> **Note**
>
> Visual Studio Code uses GDB for debugging on Linux, which requires Python. However, Python that is linked into GDB is not the same as Slicer's Python, which may cause issues. If GDB does not start because `_sysconfigdata__linux_x86_64-linux-gnu.py` file is missing then Slicer's sysconfigdata file must be copied to the expected filename. For example:
>
> ```
> cd ~/D/Slicer-SuperBuild-Debug/python-install/lib/python3.6/
> cp _sysconfigdata_m_linux2_.py _sysconfigdata__linux_x86_64-linux-gnu.py
> ```

Running the debugger on Linux

1. From a terminal, launch Visual Studio Code through the Slicer launcher to setup environment to set up the directory paths necessary for running the Slicer application. For example:

```
cd ~/D/Slicer-SuperBuild-Debug/Slicer-build
./Slicer --launch code
```

2. Set up settings.json. For example:

```
{
    "launch": {
        "configurations": [
                    {
                      "name": "Slicer debug",
                      "type": "cppdbg",
                      "request": "launch",
                      "program": "/home/perklab/D/Slicer-SuperBuild-Debug/Slicer-
↪build/bin/SlicerApp-real",
                      "args": [],
                      "stopAtEntry": false,
                      "cwd": ".",
                      "environment": [],
                      "externalConsole": false,
                      "MIMode": "gdb",
                      "setupCommands": [
```

(continues on next page)

(continued from previous page)

```
                        {
                          "description": "Enable pretty-printing for gdb",
                          "text": "-enable-pretty-printing",
                          "ignoreFailures": true
                        }
                      ],
                      "miDebuggerPath": "/usr/bin/gdb"
                    }
          ]
      }
}
```

3. Choose Run / Start debugging in the menu to start Slicer application with the debugger attached

C++ debugging with Qt Creator

Qt Creator is a cross-platform IDE that fully integrates Qt into the development of applications. Slicer CMake project is supported by Qt Creator, the following items aim at describing how it could be used.

Tip

For debugging on Windows, Visual Studio is recommended as it has much better performance and many more features than Visual Studio Code.

Prerequisites

- Build 3D Slicer in Debug mode, outside of Qt Creator, by following the *build instructions*.
- Install Qt SDK with Qt Creator.

Running the debugger

1. From a terminal, launch QtCreator through Slicer to setup environment. This will allow qtcreator to design UI using CTK and Slicer custom designer plugins.
   - Linux:

   ```
   cd /path/to/Slicer-Superbuild/Slicer-build
   ./Slicer --launch /path/to/qtcreator
   ```

   - Windows:

   ```
   cd c:\path\to\Slicer-Superbuild\Slicer-build
   .\Slicer.exe --launch /path/to/qtcreator.exe
   ```

2. Open `/path/to/Slicer-src/CMakeLists.txt` in qtcreator, when prompted, choose the build directory where Slicer was configured and compiled

   Note

   You can select either the binary tree of the SuperBuild or the binary tree of the Slicer-build that is inside the binary tree of the SuperBuild.

- choosing superbuild build directory `/path/to/Slicer-Superbuild`: allows building all of the packages that Slicer depends on and build Slicer itself, all from within Qt Creator.
- choosing inner build directory `/path/to/Slicer-Superbuild/Slicer-build`: provides a better IDE experience when working on Slicer itself (recognizing types, pulling up documentation, cross-referencing the code, . . .).

3. Click the `Run CMake` button (no arguments needed), wait until CMake has finished, then click the `Finish` button
4. Optional steps:
   - Specify make arguments (for example: `-j8`) by clicking the `Projects` tab on the left hand side, click the `Build Settings` tab at the top, click the `Details` button beside the Make build step, and add your additional arguments. This is useful if you want to build from within Qt Creator.
   - Specify run configuration by clicking the `Projects` tab on the left hand side, click the `Run Settings` tab at the top, and select your Run configuration (ex. choose SlicerApp-real). This is useful if you want to run Slicer from within Qt Creator.

C++ debugging with CodeLite

CodeLite is a relatively lightweight, cross-platform IDE.

Configure build

Right-click on project name and select Settings, then Customize

- Enable Custom Build: check
- Working Directory: enter path to Slicer-build, for example `~/Slicer-SuperBuild-Debug/Slicer-build`
- For target `Build`: enter `make`

To configure the binary for the Run command, set Program: `~/Slicer-SuperBuild-Debug/Slicer-build/Slicer` under the General tab.

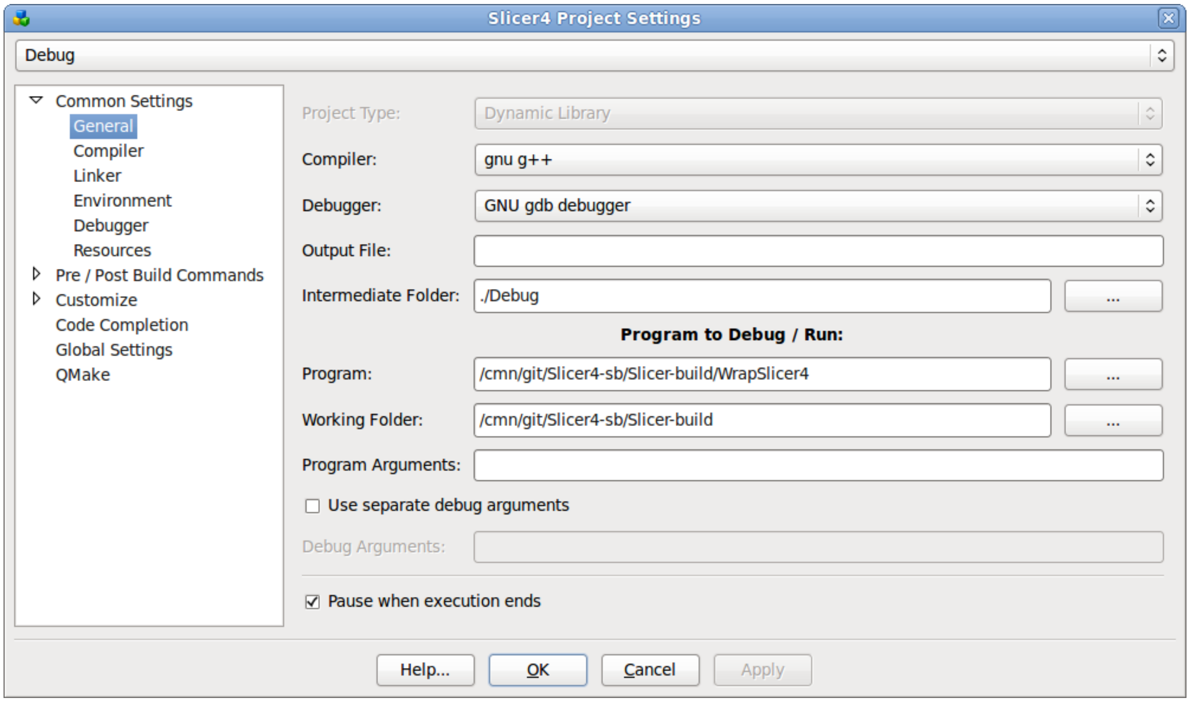
Configure debugger

This requires the use of a wrapper script, as *detailed here*.

After setting up the wrapper script (`WrapSlicer` below), change the following options:

Under Settings->General:

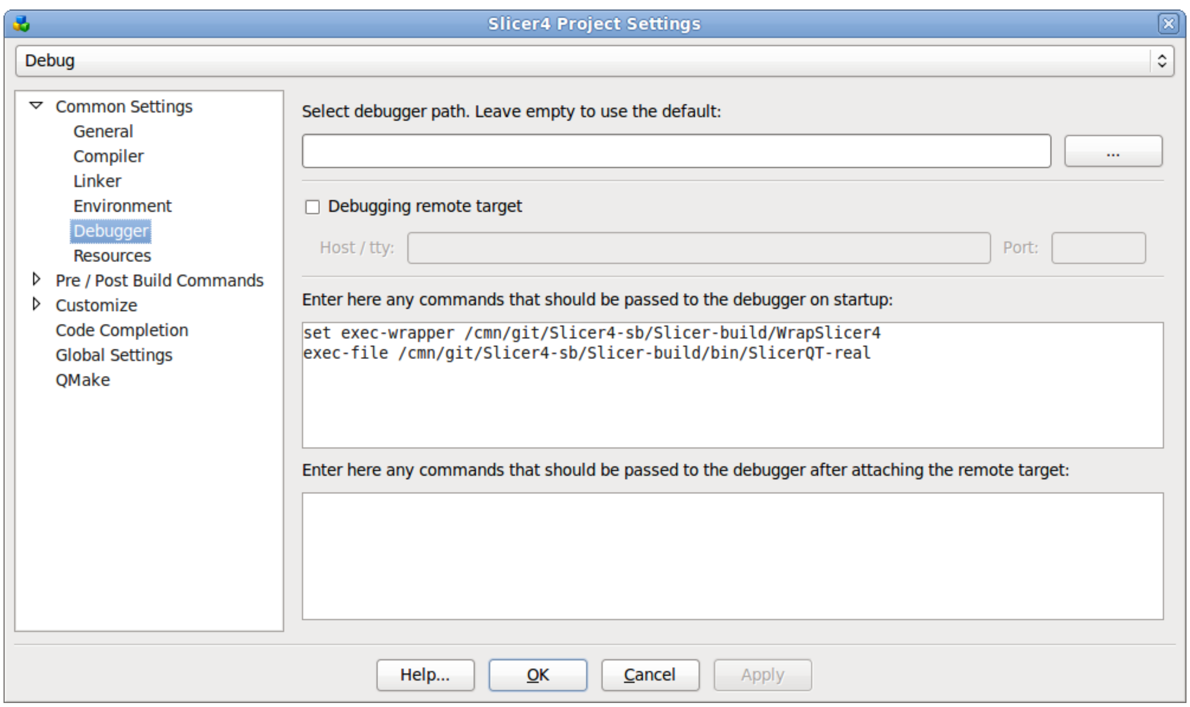
- Program: `~/Slicer-SuperBuild-Debug/Slicer-build/WrapSlicer`
- Working folder: `~/Slicer-SuperBuild-Debug/Slicer-build`



Under Settings->Debugger

- "Enter here any commands passed to debugger on startup:"

```
set exec-wrapper `~/Slicer-SuperBuild-Debug/Slicer-build/WrapSlicer`
exec-file `~/Slicer-SuperBuild-Debug/Slicer-build/bin/SlicerApp-real`
```



12.10.3 Tips and tricks

Debugging tests

- To debug a test, find the test executable:
  - `Libs/MRML/Core` tests are in the `MRMLCoreCxxTests` project
  - CLI module tests are in `<MODULE_NAME>Test` project (e.g. `ThresholdScalarVolumeTest`)
  - Loadable module tests are in `qSlicer<MODULE_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`)
- Make the project the startup project (right-click -> Set As Startup Project)
- Specify test name and additional input arguments:
  - 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.
- Start Debugging (F5)

Debugging memory leaks

See some background information in VTK leak debugging in Slicer3 and Strategies for Writing and Debugging Code in Slicer3 pages.

1. If you build the application from source, make sure VTK_DEBUG_LEAKS CMake flag is set to ON. Slicer Preview Releases are built with this flag is ON, while in Slicer Stable Releases the flag is OFF.
2. Reproduce the memory leak. When the application exits, it logs a message like this:

```
vtkDebugLeaks has detected LEAKS!
Class "vtkCommand or subclass" has 1 instance still around.
```

3. Add the listed classes to the `VTK_DEBUG_LEAKS_TRACE_CLASSES` environment variable (separated by commas, if there are multiple). For example:

```
VTK_DEBUG_LEAKS_TRACE_CLASSES=vtkCommand
```

In Visual Studio, environment variable can be added to `SlicerApp` project properties -> Debugging -> Environment.

4. Reproduce the memory leak. It will print the stack trace of the call that allocated the object, which should be enough information to determine which object instance in the code it was. For example:

```
vtkDebugLeaks has detected LEAKS!
Class "vtkCommand or subclass" has 1 instance still around.

Remaining instance of object 'vtkCallbackCommand' was allocated at:
 at vtkCommand::vtkCommand in C:\D\S4D\VTK\Common\Core\vtkCommand.cxx line 28
 at vtkCallbackCommand::vtkCallbackCommand in C:\D\S4D\VTK\Common\Core\
↪vtkCallbackCommand.cxx line 20
 at vtkCallbackCommand::New in C:\D\S4D\VTK\Common\Core\vtkCallbackCommand.h line 49
 at vtkNew<vtkCallbackCommand>::vtkNew<vtkCallbackCommand> in C:\D\S4D\VTK\Common\Core\
↪vtkNew.h line 89
 at qSlicerCoreApplicationPrivate::init in C:\D\S4\Base\QTCore\qSlicerCoreApplication.
↪cxx line 325
 at qSlicerApplicationPrivate::init in C:\D\S4\Base\QTGUI\qSlicerApplication.cxx line 232
 at qSlicerApplication::qSlicerApplication in C:\D\S4\Base\QTGUI\qSlicerApplication.cxx␣
↪line 393
 at `anonymous namespace'::SlicerAppMain in C:\D\S4\Applications\SlicerApp\Main.cxx line␣
↪40
 at main in C:\D\S4\Base\QTApp\qSlicerApplicationMainWrapper.cxx line 57
 at invoke_main in D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl␣
↪line 79
 at __scrt_common_main_seh in D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe_
↪common.inl line 288
 at __scrt_common_main in D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe_
↪common.inl line 331
 at mainCRTStartup in D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe_main.cpp␣
↪line 17
 at BaseThreadInitThunk
 at RtlUserThreadStart
```

Why is my VTK actor/widget not visible?

- Add a breakpoint in RenderOpaqueGeometry() check if it is called. If not, then:
  - Check its vtkProp::Visibility value.
    - For vtkWidgets, it is the visibility of the representation.
  - Check its GetBounds() method. If they are outside the camera frustum, the object won't be rendered.
    - For vtkWidgets, it is the bounds of the representation.

Debugging Slicer application startup issues

See instructions *here* for debugging application startup issues.

Disabling features

It may help pinpointing issues if Slicer is started with as few features as possible:

- Disable Slicer options via the command line

  ```
  ./Slicer --no-splash --ignore-slicerrc --disable-cli-module --disable-loadable-
  ↪module --disable-scriptedmodule
  ```

  - Look at all the possible options On Linux and macOS:

```
./Slicer --help
```

On Windows:

```
Slicer.exe --help | more
```

- Disable ITK plugins
  - CLI modules silently load the ITK plugins in lib/Slicer-4-13/ITKFactories. These plugins are used to share the volumes between Slicer and the ITK filter without having to copy them on disk.
  - rename lib/Slicer-4.13/ITKFactories into lib/Slicer-4.13/ITKFactories-disabled
- Disable Qt plugins
  - rename lib/Slicer-4.13/iconengine into lib/Slicer-4.13/iconengine-disabled

Console output on Windows

On Windows, by default the application launcher is built as a Windows GUI application (as opposed to a console application) to avoid opening a console window when starting the application.

If the launcher is a Windows GUI application, it is still possible to show the console output by using one of these options:

Option A. Run the application with capturing and displaying the output using the `more` command (this captures the output of both the launcher and the launched application):

```
Slicer.exe --help 2>&1 | more
```

The `2>&1` argument redirects the error output to the standard output, making error messages visible on the console, too.

Option B. Instead of `more` command (that requires pressing space key after the console window is full), `tee` command can be used (that continuously displays the output on screen and also writes it to a file). Unfortunately, `tee` is not a standard command on Windows, therefore either a third-party implementation can be used (such as `wtee`) or the built-in `tee` command in Windows powershell:

```
powershell ".\Slicer.exe 2>&1 | tee out.txt"
```

Option C. Run the application with a new console (the launcher sets up the environment, creates a new console, and starts the SlicerApp-real executable directly, which can access this console):

```
Slicer.exe --launch %comspec% /c start SlicerApp-real.exe
```

To add console output permanently, the application launcher can be switched to a console application by setting `Slicer_BUILD_WIN32_CONSOLE_LAUNCHER` CMake variable to ON when configuring the application build.

12.11 Contributing to Slicer

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

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

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

12.11.1 Submitting a Pull Request

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` described in the GitHub collaborative development documentation.

> **Tip**
>
> 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.

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.

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.

Examples:

- `BUG: Check pointer validity before dereferencing` -> implementation detail, self-explanatory (by looking at the code)
- `BUG: Fix crash in Module X when clicking Apply button`
- `ENH: More work in qSlicerXModuleWidget` -> more work is too vague, qSlicerXModuleWidget is too low level
- `ENH: Add float image outputs in module X`
- `COMP: Typo in cmake variable` -> implementation detail, self-explanatory
- `COMP: Fix compilation error with Numpy on Visual Studio`

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

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 one reviewer approved the changes and at least one business day passed without anyone objecting.
- To accommodate developers explicitly asking for more time to test the proposed changes, integration time can be delayed by few more days.

Tip

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.

Automatic testing of pull requests

Every pull request (PR) is automatically tested using GitHub Actions each time a commit is pushed. The GitHub interface prevents merging until all required CI workflows have completed successfully, ensuring that changes meet the project's testing and quality standards.

The testing infrastructure was migrated from CircleCI to GitHub Actions, but the principles described in the blog post 3D Slicer Improves Testing for Pull Requests Using Docker and CircleCI remain relevant for understanding the overall approach and concepts.

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, macOS and Windows. Results are published daily on CDash:

- Stable
- Preview)

Developers that introduced changes resulting in build or test failures are notified by email.

12.12 Style Guide

12.12.1 General

If you are editing code, take a few minutes to look at the code around you and determine its style. If they use spaces around their if clauses, you should, too. If their comments have little boxes of stars around them, make your comments have little boxes of stars around them too.

The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you are saying, rather than on how you are saying it. We present global style rules here so people know the vocabulary. But local style is also important. If code you add to a file looks drastically different from the existing code around it, the discontinuity throws readers out of their rhythm when they go to read it. Try to avoid this.

Slicer uses and extends libraries and toolkits multiple languages, which each has its own conventions. As a general rule, follow these conventions. For example, to get number of items in a VTK class use GetNumberOfXXX, but use XXXCount in a Qt class.

Line length: Preferably keep lines shorter than 80 characters. Always keep lines shorter than 120 characters. Markdown (.md) files are excluded from this restriction, to allow editing the text without worrying about line breaks.

Toolkits and libraries

- VTK coding conventions
- Qt style guide
- Python style guide

Languages

Python

- Follow the Slicer repository Flake8 configuration and run `python -m pre_commit run --all-files` to confirm compliance.
- Text encoding: UTF-8 is preferred, Latin-1 is acceptable
- Comparisons:
  - To singletons (e.g. None): use 'is' or 'is not', never equality operations.
  - To Booleans (True, False): don't ever compare with True or False (for further explanation, see PEP 8).
- Prefix class definitions with two blank lines
- Imports

- Grouped in order of scope/commonallity
  - Standard library imports
  - Related third party imports
  - Local apps/library specific imports
    - Slicer application imports and local/module imports may be grouped independently.
- One package per line (with or without multiple function/module/class imports from the package)

- Naming conventions: when PEP 8 and Slicer naming conventions conflict, Slicer wins.

C++

- Use the old-style VTK indentation (braces are in new line, indented by two spaces), until the entire Slicer code base will be updated to use current VTK indentation style.
- Use VTK naming conventions:
  - Local variable should start with a lower case. Use: `void vtkSlicerSliceLogic::SetForegroundLayer(vtkSlicerSliceLayerLogic* foregroundLayer)` Instead of: `void vtkSlicerSliceLogic::SetForegroundLayer(vtkSlicerSliceLayerLogic* ForegroundLayer) // wrong!`
  - Member variable should start with a capital letter, and in implementation should be used in conjunction with `this->` convention.

Example:

```
class Node
{
  Object &Foo();
  Object Bla;
};
Object& Node::Foo()
{
  return this->Bla;
}
```

Useful information about some coding style decisions: https://google.github.io/styleguide/cppguide.html

Source file name rules:

- C++ source files files shall have a `.cxx` extension.
- C++ Headers with declarations shall have a `.h` extension, and header definition files shall have a `.hxx` extension. The `.hxx` files shall only be included from the corresponding `.h` files or `.cxx` files.

CMake

- Macros/functions should be lower case and words separated with underscores

  `include_directories(${CMAKE_CURRENT_SOURCE_DIR}/Logic)` instead of `INCLUDE_DIRECTORIES(${CMAKE_CURRENT_SOURCE_DIR}/Logic)`

  `find_package(VTK REQUIRED)` instead of `FIND_PACKAGE(VTK REQUIRED)` or `Find_Package(VTK REQUIRED)`

- Global variables are uppercase and words separated with underscores

  `CMAKE_CURRENT_SOURCE_DIR` instead of `cmake_current_source_dir`

- Local variables are lowercase and words separated with underscores

  `foreach (file ${FILES_TO_CONFIGURE})...` instead of `foreach (FILE ${FILES_TO_CONFIGURE})...`

12.12.2 Naming conventions

- Acronyms should be written with the same case for each letter (all uppercase or all lowercase).
  - `RASToSlice` not `RasToSlice`
  - `vtkMRML` not `vtkMrml`
  - `vtkSlicer` not `vTKSlicer`
- Words should be spelled out and not abbreviated
  - `GetWindow` not `GetWin`
- File names must follow the Camel case convention
  - `TestMyFeature.cxx` not `Test-My_Feature.cxx`
- Use US English words and spelling
  - "Millimeter" not "Millimetre"
  - "Color" not "Colour"

12.12.3 Comments

- Include extensive comments in the header files
- Keep the comments up to date as the code changes
- Use the keyword \todo to flag spots in the code that need to be revisited
- Do not leave blocks of commented out code in the source file – if needed insert links to prior svn versions as comments

12.12.4 Functions

Don't mix different levels of abstraction

Examples:

When dealing with files names and path, use:

- kwsys::SystemTools in VTK classes
- QFileInfo/QDir in Qt classes
- os.path in Python.

Instead of doing string manipulation manually:

```
QString filePath = directoryPath + "/" + fileName + ".exe"
```

Prefer instead:

- VTK:

  SystemTools::JoinPath(), SystemTools::GetFilenameName()…

- Qt:

  QFileInfo(QDir directory, QString fileName), QFileInfo::suffix(), QFileInfo::absoluteFilePath()…

- Python:

  os.path.join(), os.path.splitext(), os.path.abspath()…

References: Clean Code from `Robert C. Martin: Mixing levels of abstraction within a function is always confusing. Readers may not be able to tell whether a particular expression is an essential concept or a detail. Worse, like broken windows, once details are mixed with essential concepts, more and more details tend to accrete within the functions.`

2. Use STL where you can, but:

- In VTK classes follow the [https://www.vtk.org/Wiki/VTK_FAQ#Can_I_use_STL_with_VTK.3F VTK guidelines]
  - Note that a vtkCollection is somewhat equivalent to `std::list<vtkSmartPointer<vtkObject*>>`
- In Qt classes prefer Qt Container classes

12.12.5 File layout

Includes

- Only include the necessary files, no more.
- Group includes per library
- Alphabetically sort files within groups
- Order groups from local to global: for example, module then MRML then CTK then Qt then VTK then ITK then STL.
- Implementation files should include the header files first

Example:

```
// header
// ...
// end header

#include "qSlicerMyModule.h"

// MyModule includes
#include "qSlicerMyModuleWidget.h"
#include "vtkSlicerMyModuleLogic.h"

// MRML includes
#include "vtkMRMLScene.h"

// Qt includes
#include <QDialog>

// VTK includes
#include <vtkSmartPointer.h>

// STD includes
#include <vector>
```

12.12.6 Library Dependencies

- MRML classes should only depend on vtk and itk (not Slicer Logic or Qt)
- Logic classes depend on MRML to store state
- Logic classes should encapsulate vtk and itk pipelines to accomplish specific slicer tasks (such as resampling volumes for display)
- GUI classes can depend on MRML and Logic and Qt

12.12.7 Development Practices

While developing code, enable VTK_DEBUG_LEAKS (ON by default) in your vtk build and be sure to clean up any leaks that arise from your contributions.

12.12.8 Coordinate Systems

- World space for 3D Views is in RAS (Right Anterior Superior) space. See the *Coordinate systems* page for details.
- All units are expressed in millimeters (mm)

12.12.9 String encoding: UTF-8 everywhere

Slicer uses UTF-8 everywhere: all strings in `std::string`, `char[]` arrays, files, etc. are in UTF-8 (except in rare exceptions where this is very clearly indicated). We don't use code pages or any other unicode encoding. If this leads to incorrect behavior anywhere then the underlying issue must be fixed (e.g., if a VTK function does not work correctly with UTF-8 encoded string input then a fix has to be submitted to VTK).

On Windows, process code page of all Slicer executables (main application, CLI modules, tests, etc.) are explicitly set to UTF-8 by using "ctk_add_executable_utf8" function instead of plain "add_executable" in CMake. This makes all standard API functions to use UTF-8 encoding, even in third-party libraries. This mechanism requires Windows Version 1903 (May 2019 Update) or later. On Linux and Mac, encoding is already expected to be UTF-8 (it is currently not checked or enforced in any way).

Conversion from std::string to QString:

```
std::string ss = ...
QString qs1 = QString::fromUtf8(ss); // this is slightly preferred, as it is very clear␣
↪and explicit
QString qs2 = QString(ss); // same result as fromUtf8, acceptable, as it is a bit␣
↪simpler and used throughout the code base anyway
```

Conversion from QString to std::string:

```
QString qs = ...
std::string ss  = QString::toUtf8(qs);
```

Printing to console: in general, VTK, Qt, or ITK logging macros are preferred but if for some reason text must be printed on console then use qPrintable macro. This macro converts the string

```
std::cerr << "Failed to create file " << qPrintable(filePath) << std::endl;
```

Qt logging macros (qDebug, qWarning, QFatal): these macros expect UTF-8 encoded strings, therefore do not use qPrintable macro.

File management: All filenames have to be passed to file functions (such as fopen) must be UTF-8 encoded. All text file content is expected to be UTF-8 encoded, except very rare cases when a different encoding is explicitly specified in the file (for example in incoming DICOM files may use different encoding).

12.12.10 Error and warning messages

The ITK, VTK, Qt, std::cout, std::cerr .. all appear in the error log and can easily be filtered according to their type (debug/warning/error).

- **Errors**: Error should be used to signal something that should not happen. They usually mean that the execution of the current function/code should be stopped.
- **Warnings**: Warning should be used to signal potentially dangerous behavior. Also consider using these in deprecated methods to warn your fellow developers.
- **Debugs**: For general debug and developer aimed information, one can use the debug messages.

In Qt-based classes

For error messages, use qCritical():

```
if (somethingWrongHappened)
  {
  qCritical() << "I encountered an error";
  return;
  }
```

For warnings, use qWarning():

```
qWarning() << "Be careful here, this is dangerous";
```

For debug, use qDebug():

```
qDebug() << "This variable has the value: "<< value;
```

In VTK-based classes

- For error messages, use vtkErrorMacro():

```
if (somethingWrongHappened)
  {
  vtkErrorMacro("I encountered an error");
  return;
  }
```

For warnings, use vtkWarningMacro():

```
vtkWarningMacro("Be careful here, this is dangerous");
```

For debug, use vtkDebugMacro():

```
vtkDebugMacro("This variable has the value: "<< value);
```

12.12.11 Commits

- Separate the subject from body with a blank line
- Limit the subject line to 78 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line
- Wrap the body at 72 characters
- Use the body to explain what and why vs. how. If there was important/useful/essential conversation or information, copy or include a reference
- Prefix the commit message title with "BUG:", "COMP:", "DOC:", "ENH:", "STYLE:". Note the ':' (colon) character.
- When possible, one keyword to scope the change in the subject (i.e. "STYLE: README: …", "BUG: vtkMRMLSliceLogic: …")

Commit message prefix

Commits to the Slicer repository require commit type in the comment.

Valid commit types are:

```
BUG:   - a change made to fix a runtime issue
         (crash, segmentation fault, exception, or incorrect result,
COMP:  - a fix for a compilation issue, error or warning,
DOC:   - a documentation change,
ENH:   - new functionality added to the project,
PERF:  - a performance improvement,
STYLE: - a change that does not impact the logic or execution of the code.
         (improve coding style, comments).
```

Note that the ':'(colon) directly follows the commit tag. For example, it is: "STYLE:" not "STYLE :"

Message content

A good commit message title (first line) should **explain what the commit does for the user, not ''how'' it is done**. *How* can be explained in the body of the commit message (if looking at the code of the commit is not self explanatory enough).

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`

If the commit is related to an issue (bug or feature request), you can mention it at the end of the message body by preceding the issue number with a `#` (pound) character:

```
BUG: Fix crash in Volume Rendering module when switching view layout

vtkSetAndObserveMRMLNodeEventsMacro can't be used for observing all types of vtkObjects,
only vtkMRMLNode is expected by vtkMRMLAbstractLogic::OnMRMLNodeModified(...)

see #1641
```

Where `1641` refers to the issue number in the issue tracker. Use `see` before the issue number to refer to an issue. If the commit fixes an issue and no further testing is needed then `fixed #1641` can be added to the body, which automatically closes the issue when merged.

Importing changes from external project/repository

When you update the git tag of any external project, explain in the commit message what the update does instead of just mentioning that an update in made.

This will avoid having a Slicer commit history made of uninformative messages such as:

```
r19180 - ENH: Update git tag
r19181 - BUG: Update svn revision
r19182 - ENH: revision updated
```

Ideally it should be the same message than the commit(s) in the external repository.

Read Project forks page for an exhaustive list of recommendations.

Example:

```
COMP: Update MultiVolumeExplorer to fix unused-local-typedefs warnings

$ git shortlog 17a9095..d68663f --no-merges
Jean-Christophe Fillion-Robin (1):
      COMP: Fix unused-local-typedefs warnings
```

See r23377.

Resources

- Read more on How to Write a Git Commit Message
- Discussion section of git-commit(1)

12.12.12 UI Design Guidelines

See UI Style Guide.

12.12.13 Logging

The following log levels are used in Slicer:

- Error: detected errors, conditions that should never occur
- Warning: potential errors, possible computation inaccuracies
- Info: important events, application state changes (helps to determine what the steps lead to a certain error or warning); one user action should not generate more than 1-2 info level messages
- Debug: any information that may be useful for debugging and troubleshooting

In VTK classes:

- `vtkErrorMacro("vtkMRMLClipModelsNode:: Invalid Clip Type");`
- `vtkWarningMacro("Model " << modelNode->GetName() << "'s display node is null\n");`
- `vtkDebugMacro("CreateWidget: found a glyph type already defined for this node: " << iter->second);`

In QT classes:

- `qCritical() << "qSlicerUtils::setPermissionsRecursively: Failed to set permissions on file" << info.filePath();`
- `qWarning() << "qSlicerIOManager::openScreenshotDialog: Unable to get Annotations module (annotations), using the CTK screen shot dialog.";`
- `qDebug() << "qMRMLSceneFactoryWidget::deleteNode(" <<className <<") no node";`

In Python:

- `logging.error("This is an error message. It is printed on the console (to standard error) and to the application log.")`
- `logging.warning("This is a warning message. It is printed on the console (to standard error) and to the application log.")`
- `logging.info("This is an information message. It is printed on the console (to standard output) and to the application log.")`
- `logging.debug("This is a debug message. It is only recorded in the application log but not displayed in the console. File name and line number is added to the log record.")`

12.13 Advanced Topics

12.13.1 Memory Management

Pointers to VTK object

If you are not familiar with VTK's memory management read this general introduction and this page describing usage of smart pointers. See Slicer-specific use cases and recommendations below.

Calling the plain `New()` method of VTK objects and storing the returned pointer in a plain pointer should be avoided, as this very often causes memory leaks.

Bad, should be avoided:

```
vtkMRMLScalarVolumeNode* vol = vtkMRMLScalarVolumeNode::New();
// ... do something, such as vol->GetImageData(), someObject->SetVolume(vol)...
vol->Delete();
vol=NULL;
```

Recommended:

```
vtkNew<MRMLScalarVolumeNode> vol;
//... do something, such as vol->GetImageData(), someObject->SetVolume(vol.GetPointer()).
↪..
```

Also good:

```
vtkSmartPointer<MRMLScalarVolumeNode> vol=vtkSmartPointer<MRMLScalarVolumeNode>::New();
///... do something, such as vol->GetImageData(), someObject->SetVolume(vol)...
```

vtkNew is preferred in general for new object creation, as it has a simpler syntax and used almost exclusively in the Slicer core source code. A slight inconvenience is that when we need use the `GetPointer()` method to get the actual pointer value.

A `vtkSmartPointer` pointer can be created without actually creating an object, so `vtkSmartPointer` should be used when:

- we don't know the exact object type that we want to create at the time of the pointer creation (e.g., we create a `vtkSmartPointer<vtkMRMLVolumeNode>` and later set it to point to a `vtkMRMLScalarVolumeNode` or `vtkMRMLVectorVolumeNode`), or
- we need to take the ownership of an already created object (using `vtkSmartPointer::Take(...)`; see examples below)

Factory methods

Similarly to VTK, Slicer contains some "factory" methods:

- `vtkMRMLScene::CreateNodeByClass()`
- `vtkMRMLScene::GetNodesByClass()`
- ...

Factory methods return a pointer to a VTK object (with a reference count of 1) that the caller "owns", so the caller must take care of releasing the object to avoid memory leak.

In C++, it is recommended to make a smart pointer take the ownership of the returned raw pointer. For example:

```
// GetNodesByClass is a factory method, therefore a smart pointer is used to take the␣
↪ownership of the returned object
vtkSmartPointer<vtkCollection> nodes = vtkSmartPointer<vtkCollection>::Take(scene->
↪GetNodesByClass("vtkMRMLModelNode"));
```

In Python, the returned Python object maintains a reference to the underlying VTK object, therefore an extra reference is no longer needed and it is recommended to be immediately removed using the `UnRegister` method:

```
nodes = scene.GetNodesByClass("vtkMRMLModelNode")
nodes.UnRegister(None) # GetNodesByClass method is NOT marked with VTK_NEWINSTANCE,␣
↪manual unregistration is needed
```

Here are the different naming conventions for such "factory" methods:

- `GetXXX`: Return an existing object, reference count is not changed, the caller is not responsible for the object.
- `NewXXX`: Solely instantiate an object (typically a factory method). The caller is responsible to decrement the reference count.
- `CreateXXX`: Instantiate and configure an object. The caller is responsible to decrement the reference count. If `XXX` is a MRML node, it is NOT added into the scene.
- `CreateAndAddXXX`: Instantiate, configure and add into the scene a MRML node. The caller is not responsible to decrement the reference count.

VTK_NEWINSTANCE wrapper hint

If a factory method is marked with the VTK_NEWINSTANCE hint then the ownership is transferred to Python where garbage collection takes care of deleting the object when it is no longer needed. Calling object.UnRegister(None) is prohibited, as it would prematurely delete the object and may crash the application. In C++, the VTK_NEWINSTANCE hint has no effect, the caller of the factory method must still take the ownership of the returned object the same way as without the hint.

```
box = roiNode.CreateROIBoxPolyDataWorld()
# no need to call UnRegister, as CreateROIBoxPolyDataWorld method is marked with VTK_
↪NEWINSTANCE
```

Loadable modules (C++)

If storing in a new variable:

```
vtkSmartPointer<vtkCollection> nodes = vtkSmartPointer<vtkCollection>::Take(mrmlScene->
↪GetNodesByClass("vtkMRMLLinearTransformNode"));
```

If the variable is created already:

```
vtkSmartPointer<vtkCollection> nodes;
nodes.TakeReference(mrmlScene->GetNodesByClass("vtkMRMLLinearTransformNode"));
```

Unsafe, legacy method without using smart pointers (not recommended, because the Delete() method may be forgotten or skipped due to an early return from the function):

```
vtkCollection* nodes = mrmlScene->GetNodesByClass("vtkMRMLLinearTransformNode");
// ...
nodes->Delete();
```

Python scripts and scripted modules

Factory methods return an object that the caller owns (and thus the caller has to delete, with reference count >0) and Python adds an additional reference to this object when stored in a Python variable, resulting in a reference count of >1. To make sure that the object is deleted when the Python variable is deleted, we have to remove the additional reference that the factory method added. There is currently no automatic mechanism to remove that additional reference, so it has to be done manually by calling UnRegister (the reference count shouldn't be explicitly set to any specific value, it is only allowed to increment/decrement it using Register/UnRegister):

```
nodes = slicer.mrmlScene.GetNodesByClass('vtkMRMLLinearTransformNode')
nodes.UnRegister(slicer.mrmlScene) # reference count is increased by both the factory␣
↪method and the python reference; unregister to keep only the python reference
# ...
```

To avoid forgetting the UnRegister call, it is better to avoid factory methods whenever it is possible.

For example, instead of using the CreateNodeByClass factory method and call UnRegister manually:

```
n = slicer.mrmlScene.CreateNodeByClass('vtkMRMLLinearTransformNode')
slicer.mrmlScene.AddNode(n)
n.UnRegister(slicer.mrmlScene)
```

this should be used:

```
n = slicer.mrmlScene.AddNewNodeByClass('vtkMRMLLinearTransformNode')
```

Note: MRML scene's `CreateNodeByClass` creates a node with the default settings set in the scene for that node type (using vtkMRMLScene::AddDefaultNode).

12.13.2 Working directory

Similarly to other software, the current directory associated with Slicer corresponds to the folder the application executable is started from.

Since the current working directory can be changed anytime by any module or Python package (e.g., to more conveniently write files in a specific directory) and it is not possible to enforce that the directory is restored to the original.

Slicer provides a way to reliably access the working directory at startup time, through the `startupWorkingPath` application property:

In Python:

```
slicer.app.startupWorkingPath
```

In C++:

```
qSlicerCoreApplication::startupWorkingPath()
```

12.13.3 View layout definition

View layout definition describes what views (3D, slice, plot, table, etc.) should be displayed and where. It is specified by an XML string.

A layout may contain multiple *viewports*, each viewport is a separate window, which may be displayed in the main application window or separately, for example on a second screen.

XML elements and their attributes:

- `viewports`: optional, if specified then it must be the root element. It can be used for specifying multiple viewports. It contains nested `layout` elements.
- `layout`: The layout elements describe widget containers that embed one or multiple items. Arrangement of items are specified by type attribute of the layout element. It may be root element or it may be nested in `viewports` or `item` element.
  - `type`: `vertical`, `horizontal`, `grid`, `tab`.
  - `split`: `true` or `false` (default), if `true` then the user can resize the views in it by dragging the splitter between the views. Default size can be set using `splitSize` attribute of child items. Only for `vertical` and `horizontal` layout type.
  - `name`: unique name of the layout, required if there are multiple viewports. If name is not specified then the empty string will be used as name. The empty string is a valid name, which refers to the default viewport, i.e., that is displayed in the application's main window.
  - `label`: optional, if specified then this will be used as
  - `dockable`: `true` (default) or `false` display the viewport as a dockable widget.
  - `dockPosition`: If dockable, this can make the widget be docked by default. Valid choices are `floating` (default), `top`, `bottom`, `left`, `right`, `bottom-left`, `bottom-right`, `top-left`, `top-right`.
- `item`: container for view(s) or layouts, nested in `layout` element.
  - `splitSize`: default size if split is enabled in the layout

- `view`: view widget, nested in `item` element.
  - `name`: this name is displayed in the view's title bar
  - `horizontalStretch` or `verticalStretch`: Relative size of views can be adjusted by specifying these stretch factors. The stretch factor must be an integer in the range of [0, 255].
  - `row` and `column`: row and column index. Only for `grid` layout type.
  - `class`: class of the view node, such as `vtkMRMLSliceNode`, `vtkMRMLViewNode`, `vtkMRMLTableViewNode`, `vtkMRMLPlotViewNode`.
  - `singletontag`: layout name of the view node (`1`, `2`, … for 3D views; `Red`, `Yellow`, … for slice views)
- `property`: contains view properties
  - `name`: property name, such as `viewlabel` (displayed in the view's title bar), `orientation` (for slice views),
  - element text: property value

Example: Simple 4-up view layout

```xml
<layout type="vertical" split="true">
  <item>
  <view class="vtkMRMLViewNode" singletontag="1">
    <property name="viewlabel" action="default">1</property>
  </view>
  </item>
  <item>
  <view class="vtkMRMLSliceNode" singletontag="Red">
    <property name="orientation" action="default">Axial</property>
    <property name="viewlabel" action="default">R</property>
    <property name="viewcolor" action="default">#F34A33</property>
  </view>
  </item>
</layout>
```

Example: Layout containing two viewports

```xml
<viewports>
 <!--default viewport-->
 <layout type="horizontal">
  <item>
   <view class="vtkMRMLSliceNode" singletontag="Red">
    <property name="orientation" action="default">Axial</property>
    <property name="viewlabel" action="default">R</property>
    <property name="viewcolor" action="default">#F34A33</property>
   </view>
  </item>
  <item>
   <view class="vtkMRMLViewNode" singletontag="1">
    <property name="viewlabel" action="default">1</property>
   </view>
  </item>
 </layout>
 <!--second dockable viewport-->
```

(continues on next page)

(continued from previous page)

```
 <layout name="views+" type="horizontal" label="Views+" dockable="true" dockPosition=
↪"bottom">>
  <item>
   <view class="vtkMRMLSliceNode" singletontag="Red+">
    <property name="orientation" action="default">Axial</property>
    <property name="viewlabel" action="default">R+</property>
    <property name="viewcolor" action="default">#f9a99f</property>
    <property name="viewgroup" action="default">1</property>
   </view>
  </item>
  <item>
   <view class="vtkMRMLViewNode" singletontag="1+" type="secondary">
    <property name="viewlabel" action="default">1+</property>
    <property name="viewgroup" action="default">1</property>
   </view>
  </item>
 </layout>
</viewports>
```

12.13.4 Event broker

The basic idea of the EventBroker is to avoid the following kind of code that shows up in GUIs and other classes that depend on events generated by other classes:

```
node->AddObserver(vtkCommand::ModifiedEvent, callbackCommand)
```

Problem

The problems with this:

- node 'owns' the observer, but the callbackCommand is opaque so it doesn't know anything about what will happen when the event is invoked
- the GUI needs to explicitly remove the observer before it is destroyed
- node is not introspectable; you cannot get a list of observers on the node (all data and methods are private)
- there's no easy way to know what side effects will happen for any Set call (either a priori or experimentally).
- there's no way to collapse events or disable them

Solution

The EventBroker solve this by introducing a singleton that manages all observations.

The EventBroker is a singleton, available as:

```
vtkEventBroker* broker = vtkEventBroker::GetInstance();
```

with the broker, you can make a call like the following:

```
broker->AddObservation(node, vtkCommand::ModifiedEvent, this, callbackCommand);
```

where `node` is the subject of the observation.

> **Tip**
>
> The broker is also available for use from Python, but for adding observers in Python, it is recommended to use the higher-level `slicer.util.VTKObservationMixin` instead.

The broker does the following:

- adds `DeleteEvent` observers to both `node` and `this` so it can remove the observation automatically when either side is destroyed
- keeps an introspectable list of all observers it knows about
- has an option to keep a log of all event invocations for debugging and performance analysis
- has an option to turn off all event invocations
- has an *asynchronous* option to queue all event invocations and invoke them later (off by default)
- has option to collapse redundant events in the queue

Future options include:

- add a timer to log the amount of time taken to process each event
- specify that some observations must be handled synchronously (e.g. for progress events that shouldn't be collapsed)

See the file vtkEventBroker API documentation for more information on logging options and introspection features.

How to work with the vtkObserverManager

Discussion of Jan 27 2012:

[. . .] the problem wasn't that the Event Broker was used instead of going through `vtkObserverManager`, but that the `vtkObject::AddObserver()` was used with the callback command of the observer manager (that was later deleted which made the client data of the callback dirty when the event was fired).

We should eventually review the entire "observation" mechanism, there are currently 3 ways of observing vtkObjects:

- via the `vtkObserverManager` API (please everybody use that one for now)
- via the `EventBroker` directly
- via `vtkObject::AddObserver` directly

All three methods are valid but inconsistent with each other. In addition, it is still not user friendly to observe MRML nodes (using static functions as callback. . . how to synchronize a MRML node with a VTK widget in a displayable manager? how to observe with Python. . .).

Bugs related to observers:

- Volume Rendering (Slicer issue #1572, Slicer issue #1744)
  - Use AddObserver with vtkObserverManager callback that is deleted before an observed event is fired, crash when trying to dereference the deleted observer.
- Python (Slicer issue #1656)
  - PyObject of the observer is deleted before an observed event is fired. crash in the callback
- Current design notes
  - For logics, `vtk[SetAnd]ObserveMRMLNode[Events]Macro` can only observe MRML nodes (see `vtkMRMLAbstractLogic::MRMLNodesCallback`)

- For `QObjects` listening to `vtkObjects`, destruction must unlink observations (modules must call `setMRMLScene(0)` in destructor?)

References

The following reference provided insight on requirements and design precedents.

- Wikipedia definition of Observer Pattern
- A C++ implementation
- The Qt implementation
- Java Message Service (JMS) API

12.14 Credits

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

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

P

Q

R

S

T

U

V

W