VTK

VTK Developers

May 17, 2024

CONTENTS

VTK is an open-source software system for image processing, 3D graphics, volume rendering, and visualization. Our documentation is a comprehensive resource for both new and seasoned VTK users and includes tutorials, examples, and best practices to help you unlock the full power of VTK's advanced algorithms and rendering techniques.

We invite you to join the VTK community and explore our documentation to find out how you can use VTK to bring your visions to life.

CHAPTER

ONE

ABOUT

1.1 Overview

The Visualization Toolkit (VTK) is a robust and open-source software system that provides advanced features in 3D computer graphics, image processing, modeling, volume rendering, and scientific visualization. It offers threaded and distributed-memory parallel processing for scalability and better performance.

VTK is a cross-platform library that can run on many operating systems, including Windows, macOS, Linux, and even the web and mobile devices.

VTK is widely used in both academic and commercial settings, as well as in government institutions such as Los Alamos National Lab and CINECA. The software was originally published in the textbook titled "The Visualization Toolkit, an Object-Oriented Approach to 3D Graphics" and has grown significantly since its release in 1994 with an extensive worldwide user base.

VTK maintains a high-quality software process, which includes CMake, CTest, CDash, and CPack. The software is written in C++ with additional language bindings to reach a broader audience, with an excellent interoperability with Python.

As open source software, VTK is free to use for any purpose. Technically, VTK has a BSD-style license, which imposes minimal restrictions for both open and closed source applications.

If you're interested in exploring the growth and usage patterns of VTK, we provide you with our statistics. The statistics are available on Open Hub, a platform focused on community-driven software, and PyPI stats, which provides download statistics for VTK packages. By analyzing these statistics, you can gain insights into the community's size, VTK's adoption rates, and popularity. Check out the links below for more information:

- Open Hub
- PyPI stats

1.2 Features

VTK provides a comprehensive set of features that support visualization, modeling, and data analysis. Here are some highlights:

Filters VTK's filter-based architecture processes data by transforming and manipulating it through a pipeline of successive filters. This approach produces derived data that can be rendered using VTK's graphics system. Filters can be combined into a dataflow network, which enables a flexibly configurable workflow.

Graphics System VTK provides a sophisticated rendering abstraction layer over the underlying graphics library (OpenGL with experimental support for WebGL), simplifying the creation of engaging visualizations.

Data Model VTK's core data model has the ability to represent almost any real-world problem related to physical science. The fundamental data structures are particularly well-suited to medical imaging and engineering work that involves finite difference and finite element solutions.

Data Interaction VTK provides several tools for interactive data exploration and analysis, including 3D widgets, interactors, and 2D widget libraries integration like Qt. These enable powerful user interaction capabilities, making it easier to understand the content, shape, and meaning of data.

2D Plots and Charts VTK supports a full set of 2D plot and chart types for tabular data visualization. It also includes picking and selection capabilities, allowing users to query data interactively. VTK's excellent interoperability with Python and Matplotlib further increases its flexibility.

Parallel Processing VTK offers excellent support for scalable distributed-memory parallel processing under MPI. VTK filters implement finer-grained parallelism using vtkSMP for coarse-grained threading and vtk-m for fine-grained processing on many-core and GPU architectures. These parallel processing capabilities make VTK highly efficient and suited for processing large data sets.

1.3 License

VTK is distributed under the OSI-approved BSD 3-clause License. See here for details.

1.4 Citing

When citing VTK in your scientific research, please mention the following work to support increased visibility and dissemination of our software:

```
Schroeder, Will; Martin, Ken; Lorensen, Bill (2006), The Visualization Toolkit (4th ed.),
↪ Kitware, ISBN 978-1-930934-19-1
```

For your convenience here is a bibtex entry:

```
@Book{vtkBook,
  author    = "Will Schroeder and Ken Martin and Bill Lorensen",
  title     = "The Visualization Toolkit (4th ed.)",
  publisher = "Kitware",
  year      = "2006",
  isbn      = "978-1-930934-19-1",
}
```

To cite a specific filter, check for extra references in the included headers or the doxygen documentation of the filter.

1.5 History

2016 - Rendering Backend in ParaView 5.0

See Brand-New Rendering Backend in ParaView 5.0.

2014 - Transition from OpenGL to OpenGL2

See New OpenGL Rendering in VTK.

1993 - Origin

VTK was originally part of the textbook The Visualization Toolkit An Object-Oriented Approach to 3D Graphics. Will Schroeder, Ken Martin, and Bill Lorensen—three graphics and visualization researchers—wrote the book and companion software on their own time, beginning in December 1993, with legal permission from their then-employer, GE R&D. The motivation for the book was to collaborate with other researchers and develop an open framework for creating leading-edge visualization and graphics applications.

VTK grew out of the authors' experiences at GE, particularly with the LYMB object-oriented graphics system. Other influences included the VISAGE visualization system developed by Schroeder et. al; the Clockworks object-oriented computer animation system developed at Rensselaer Polytechnic Institute; and the Object-Oriented Modeling and Design book, which Bill Lorensen co-authored.

After the core of VTK was written, users and developers around the world began to improve and apply the system to real-world problems. In particular, GE Medical Systems and other GE businesses contributed to the system, and researchers such as Dr. Penny Rheinghans began to teach with the book. Other early advocates include Jim Ahrens at Los Alamos National Laboratory and generous oil and gas supporters.

To address what was becoming a large, active, and world-wide community, Ken and Will—along with Lisa Avila, Charles Law, and Bill Hoffman—left GE in 1998 to found Kitware, Inc. Since that time, hundreds of additional developers have turned VTK into what is now the premier visualization system in the world. Sandia National Laboratories, for example, has been a strong supporter and co-developer, revamping 2D charting and information visualization in VTK.

1.6 Acknowledgments

Many institutions have taken part in the development of VTK. Some of the most fundamental work came from the following:

- Kitware
- Los Alamos National Lab (LANL)
- National Library of Medicine (NLM)
- Department of Energy (DOE) ASC Program
- Sandia National Laboratories
- Army Research Laboratory (ARL)

Special thanks to all the contributors !

1.7 Commercial Use

We invite commercial entities to use VTK.

VTK is part of Kitware's collection of commercially supported open-source platforms for software development.

VTK's License makes Commercial Use Available

- VTK is a free open source software distributed under a *BSD style license*.
- The license does not impose restrictions on the use of the software.
- VTK is NOT FDA approved. It is the users responsibility to ensure compliance with applicable rules and regulations.

1.8 Contact Us

We want to hear from you! If you have any questions, suggestions or bug reports regarding VTK, there are several communication channels available for you:

VTK Forum

Visit the VTK Discourse forum for community-driven support, to share your experiences, exchange ideas and best practices, and to discuss challenges.

Issue Tracker

Use our public issue tracker to report any bugs or request enhancements. This tracker is a ticket-based system that allows you to keep track of your issues and follow up on their progress.

Commercial and Confidential Consulting

For commercial or confidential consulting related to VTK or any of our other products and services, please contact Kitware's advanced support team for personalized assistance.

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

Welcome to VTK! We recommend that you start by reading The VTK Book, a comprehensive guide to VTK that covers all aspects of its functionality. Additionally, you may find it helpful to explore the VTK Examples, a collection of useful reference materials that demonstrate how to use VTK's different modules and features.

Before diving into VTK's functionality, ensure that your system meets its *system requirements*. Depending on your programming experience and needs, you can choose different programming languages to work with VTK. We have documentation on how to use VTK with *Python*, *Jupyter*, *C++ and CMake*, *Javascript*, and *WebAssembly*.

Lastly, to help address your specific needs, you may also consider exploring existing free and open-source *frameworks or applications* that already leverage VTK. These frameworks and applications can be extended and customized to work for specific use cases and may provide ready-to-use solutions for your project.

2.2 System requirements

Runtime

- At least Python 3.x to use scripting capabilities
- Minimum macOS version 10.10.
- Minimum OpenGL version is 3.2 but a higher versions may be required for more advanced features.

Build-time

Check the build *prerequisites*.

2.3 Using Python

VTK is available on PyPI for Windows, macOS and Linux.

```
pip install vtk
```

or in a virtual environment if you want to install the package only locally instead of system-wide

Linux

```
python -m venv ./env
source ./env/bin/activate
pip install vtk
```

macOS

```
python -m venv ./env
source ./env/bin/activate
pip install vtk
```

Windows

Using `PowerShell`

```
python -m venv env
.\env\Activate.ps1
pip install vtk
```

or using `cmd.exe`

```
python -m venv env
.\env\activate.bat
pip install vtk
```

To verify the installation try to import vtk from an interactive python environment:

```
>>> import vtk
>>> print(vtk.__version__)
9.2.6
```

That's it ! You may now try some of the tutorials, how to guides or examples.

If you are looking for a higher-level interface to VTK in Python, you may want to explore using PyVista as it exposes VTK in a "Pythonic" manner.

2.4 Using Jupyter

When it comes to rendering with VTK in Jupyter, there are several options.

To harness the full power of VTK in Jupyter, you may want to leverage PyVista and Trame. PyVista exposes a high-level interface to VTK for plotting and when combined with Trame, empowers users to bring the full power of VTK to a Jupyter notebook. We have a post on the VTK discourse about this. See PyVista's documentation for more information on using PyVista's wrappings of VTK in Jupyter.

itkwidgets is one example of a domain-specific Jupyter viewer built on VTK. To try out itkwidgets, check this example.

2.5 Using C++ and CMake

CMake is an open-source platform-independent build system that manages the entire software build process, from source code to executable binary. If you're new to CMake, you can find more information on the CMake website.

Installing a binary release

Pre-built VTK releases maintained by the community exist for a number of distributions, as shown in the following table:

Operating System/ Package manager	Package Name	Version
Fedora Rawhide	vtk-devel	
Fedora 38	vtk-devel	
Fedora 37	vtk-devel	
Ubuntu 23.04 (lunar)	libvtk9-dev	
Ubuntu 22.10 (kinetic)	libvtk9-dev	
Ubuntu 22.04 (jammy)	libvtk9-dev	
Ubuntu 20.04 (focal)	libvtk7-dev	
Debian unstable	libvtk9-devel	
Debian testing	libvtk9-devel	
Debian stable	libvtk9-devel	
Gentoo	vtk	
homebrew	vtk	
vckpg	vtk	
spack	vtk	

Note that these packages may be lacking some optional features such as mpi, qt etc. or, they may not contain the latest VTK features. Check the documentation of each package to verify that the build contains what you need. If what you need is missing you will need to *build vtk from scratch*.

Building an executable

Once VTK is installed using either of the methods above you can use it in your project utilizing the find_package infrastructure of cmake:

```cmake
find_package(VTK
  COMPONENTS
  .. list of vtk modules to link to
)

# your executable
add_executable(testExample ...)

# link to required VTK libraries
target_link_libraries(testExample
  PRIVATE
   ${VTK_LIBRARIES}
)

vtk_module_autoinit(
  TARGETS testExample
  MODULES ${VTK_LIBRARIES}
)
```

vtk_module_autoinit() is responsible for triggering static code construction required for some VTK classes. For more details regarding the autoinit system of VTK see *here*.

The list of required vtk modules depends on the files `#included` in your code. The module a header file belongs to is determined in most cases by its location in the VTK source tree. For, example `vtkXMLPolyDataReader` is located under `IO/XML` so it belongs to the `IOXML` module, to verify check the accompanying `vtk.module` file in the same directory.

The above method works in most cases but it does not express the dependencies that some module have. A better (and easier) way to find the required modules is the VTKModulesForCxx script.

For example, running the script on the CylinderExample we get the following suggestion:

```
find_package(VTK
 COMPONENTS
    CommonColor
    CommonCore
    FiltersSources
    RenderingCore
    #
    # These modules are suggested since they implement an existing module.
    # You may need to uncomment one or more of these.
    # If vtkRenderWindow is used and you want to use OpenGL,
    #   you also need the RenderingOpenGL2 module.
    # If vtkRenderWindowInteractor is used,
    #    uncomment RenderingUI and possibly InteractionStyle.
    # If text rendering is used, uncomment RenderingFreeType
    #
    # InteractionStyle  # implements VTK::RenderingCore
    # RenderingCellGrid # implements VTK::RenderingCore
    # RenderingFreeType # implements VTK::RenderingCore
    # RenderingOpenGL2  # implements VTK::RenderingCore
    # RenderingUI       # implements VTK::RenderingCore
)
```

Based on the suggestions of the script and the template above the relevant sections of the `CMakeLists.txt` are:

```
...
find_package(VTK COMPONENTS
  CommonColor
  CommonCore
  FiltersSources
  InteractionStyle
  RenderingContextOpenGL2
  RenderingCore
  RenderingFreeType
  RenderingGL2PSOpenGL2
  RenderingOpenGL2
)

add_executable(CylinderExample CylinderExample.cxx)
target_link_libraries(CylinderExample PRIVATE ${VTK_LIBRARIES})
# vtk_module_autoinit is needed
vtk_module_autoinit(
  TARGETS CylinderExample
```

(continues on next page)

(continued from previous page)

```
  MODULES ${VTK_LIBRARIES}
)
```

The full source of the example can be found here.

To build the example:

```
mkdir build
cd build
ccmake ../ # or cmake-gui if on Windows
```

Hit `C` if using `ccmake` or the configure button if using `cmake-gui`. If VTK was built from scratch you will need to set `VTK_DIR` to the installation path. If `ccmake/cmake-gui` reports no errors quit `ccmake/cmake-gui` and build the project as follows:

```
cmake --build .
```

To run the example

```
./CylinderExample
```

For more examples check the tutorials, how to guides or examples sections of the vtk examples website.

2.6 Using Javascript

vtk.js is an implementation of VTK in JavaScript that consists of an ES6 class library which can be integrated into any web application. See here to learn more about the differences between VTK C++ and vtk.js.

2.7 Using WebAssembly

VTK-Wasm is a prototype infrastructure that enables the compilation of VTK C++ code to WebAssembly via Emscripten. This feature is still under active development.

To learn more about VTK-Wasm and its capabilities, please take a look at the following resources:

- Examples of WebAssembly applications that use VTK for rendering.
- A collection of VTK web-based benchmark applications.
- A guide on using the experimental WebGPU feature in VTK-Wasm..
- *Instructions for building VTK using Emscripten for WebAssembly.*
- vtk-wasm-docker for building and publishing the `kitware/vtk-wasm` docker images.
- Deep dive into WebAssembly & WebGPU in VTK: presentation from April 28th, 2023. This presentation covers topics such as Emscripten, VTK-wasm Docker image, WASM Dev tools, VTK and WebGPU: PolyData Mapper, API inspection with RenderDoc, and performance profiles.

We welcome your feedback and contributions to this project. Feel free to share your experiences, questions, and ideas in the web/vtk-wasm category of the VTK Discourse forum. Stay tuned for updates and new developments!

2.8 Using existing frameworks and applications

There are many VTK-based, free, open-source applications for scientific, bio-medical and medical image visualization and processing; several of them are extensible frameworks that can be customized for particular use cases. ParaView, Trame, PyVista, and 3D Slicer are examples. Therefore, it is worth evaluating if any of these would allow you to address your challenges. This would save time by avoiding redeveloping everything from scratch and by capitalizing on large communities with thousands of experts.

Generally, the default (complex, but powerful) user interface of these applications allows one to figure out the complete workflow. Once one knows exactly what and how to do it, they can create a small Python scripted module that automates most of the steps and provides a simplified user interface.

CHAPTER

THREE

LEARNING

The VTK textbook offers thorough descriptions of important visualization algorithms and techniques that can be found in VTK along with some examples.

More examples and how-to guides can be found at the examples website. Check also this script for getting examples focused around specific classes.

Community discussion takes place on the VTK Discourse forum.

Commercial support and training are available from Kitware.

There is also a collection of technical guides related to VTK that have been published as blog-posts in the past:

- Improved VTK - numpy integration (part 1)
- Improved VTK - numpy integration (part 2)
- Improved VTK - numpy integration (part 3)
- Improved VTK - numpy integration (part 4)
- Improved VTK - numpy integration (part 5)
- vtkProgrammableFilter
- vtkPythonAlgorithm
- A VTK pipeline primer (part 1)
- A VTK pipeline primer (part 2)
- A VTK pipeline primer (part 3)
- Streaming in VTK: Time
- Streaming in VTK: Spatial
- Spatial Streaming and Compositing

For more posts related to VTK see here.

CHAPTER

FOUR

SUPPORTED DATA FORMATS

Below is a list of all available readers and writers in VTK sorted by extension. Note that for the same extension it could be more than one matching reader/writer since the same extensions are often used across different formats. The list is generated based on a yaml file that contains all the relevant information.

To enable a reader/writer you need to enable the associated module during configuration:

```
cmake -DVTK_MODULE_ENABLE_<module name>=WANT ...
```

or setting the flag value via `ccmake/cmake-qt`.

For example to enable `vtkPNGWriter` which belongs to VTK::IOImage

```
cmake -DVTK_MODULE_ENABLE_VTK_IOImage=WANT ...
```

For more details on enabling module see the module system *api*.

Warning: the list is incomplete, this is work in progress

- Stanford Exploration Project files reader:
  - Extension: .H
  - reader: vtkSEPReader
  - module: VTK::IOImage
- Alembic scene format:
  - Extension: .abc
  - writer: vtkAlembicExporter
  - module: VTK::IOAlembic
- AVI video files (Windows only):
  - Extension: .avi
  - writer: vtkAVIWriter
  - module: VTK::IOMovie
- LIDAR data using PDAL:
  - Extensions: .bin, .bpf, .csd, .csv, .greyhound, .gpkg, .icebride, .las, .laz, .mat, .nitf, .nsf, .ntf, .pcd, .ply, .pts, .qi, .rxp, .sbet, .sqlite, .sid, .tindex, .txt, .h5
  - reader: vtkPDALReader

  - module: VTK::IOPDAL
- Windows BMP file:
  - Extension: .bmp
  - reader: vtkBMPReader
  - writer: vtkBMPWriter
  - module: VTK::IOImage
- FLUENT native format:
  - Extensions: .cas, .dat
  - reader: vtkFLUENTReader
  - module: VTK::IOGeometry
- MotionFX motion definitions cfg files:
  - Extension: .cfg
  - reader: vtkMotionFXCFGReader
  - module: VTK::IOMotionFX
- Computer Graphics Metafile:
  - Extension: .cgm
  - writer: vtkCGMWriter
  - module: VTK::IOGeometry
- CONVERGE CFD CGNS format:
  - Extension: .cgns
  - reader: vtkCONVERGECFDCGNSReader
  - module: VTK::IOCGNSReader
- CGNS format:
  - Extension: .cgns
  - reader: vtkCGNSReader
  - module: VTK::IOCGNSReader
- LS-Dyna databases:
  - Extension: .d3plot
  - reader: vtkLSDynaReader
  - module: VTK::IOLSDyna
- Tabulat data in Tecplot ascii format:
  - Extensions: .dat, .DAT
  - reader: vtkTecplotTableReader
  - module: VTK::IOTecplotTable
- FLUENT CFF format:
  - Extensions: .dat.h5, .cas.h5

  - reader: vtkFLUENTCFFReader
  - module: VTK::IOFLUENTCFF
- DICOM medical images:
  - Extension: .dcm
  - reader: vtkDICOMImageReader
  - module: VTK::IOImage
- Digital Elevation Map File:
  - Extension: .dem
  - reader: vtkDEMReader
  - module: VTK::IOImage
- Movie.BYU files:
  - Extension: .g
  - reader: vtkBYUReader
  - writer: vtkBYUWriter
  - module: VTK::IOGeometry
- IOSS (Sierra IO System), writer supports only Exodus files:
  - Extension: .g .e .h .gc .ex2 .ex2v2 .exo .gen .par .exoII .exii .ex-timeseries .cgns
  - reader: vtkIOSSReader
  - writer: vtkIOSSWriter
  - module: VTK::IOIOSS
- Chaco graph partitioning output files:
  - Extensions: .graph, .coords
  - reader: vtkChacoReader
  - module: VTK::IOGeometry
- VERAout-tools:
  - Extension: .h5
  - reader: vtkVeraOutReader
  - module: VTK::IOVeraOut
- CONVERGE CFD format:
  - Extension: .h5
  - reader: vtkCONVERGECFDReader
  - module: VTK::IOCONVERGECFD
- H5Part particle files:
  - Extension: .h5part
  - reader: vtkH5PartReader
  - module: VTK::IOH5Part

- hdf files generated from xRage, a LANL physics code:
  - Extension: .h5rage
  - reader: vtkH5RageReader
  - module: VTK::IOH5Rage
- GE TRUCHAS format:
  - Extensions: .hdf5, .h5
  - reader: vtkTRUCHASReader
  - module: VTK::IOTRUCHAS
- Radiance HDR file:
  - Extension: .hdr
  - reader: vtkHDRReader
  - module: VTK::IOImage
- AVS UCD Binary/ASCII Files:
  - Extension: .inp
  - reader: vtkAVSucdReader
  - module: VTK::IOGeometry
- JPEG Files:
  - Extensions: .jpg, .jpeg
  - reader: vtkJPEGReader
  - writer: vtkJPEGWriter
  - module: VTK::IOImage
- LIDAR data in LAS format:
  - Extension: .las
  - reader: vtkLASReader
  - module: VTK::IOLAS
- binary UNC meta image data:
  - Extensions: .mhd, .mha
  - reader: vtkMetaImageReader
  - writer: vtkMetaImageWriter
  - module: VTK::IOImage
- NetCDF-based medical image developed at [BIC:
  - Extension: .mnc
  - reader: vtkMINCImageReader
  - writer: vtkMINCImageWriter
  - module: VTK::IOMINC
- H.264-encoded MP4 files (Windows only):

  - Extension: .mp4
  - writer: vtkMP4Writer
  - module: VTK::IOMovie
- MRC Image Files:
  - Extensions: .mrc, .ali, .st, .rec
  - reader: vtkMRCReader
  - module: VTK::IOImage
- NetCDF UGRID file:
  - Extensions: .nc, .ncdf
  - reader: vtkNetCDFUGRIDReader
  - module: VTK::IONetCDF
- CAM NetCDF (Unstructured):
  - Extensions: .nc, .ncdf
  - reader: vtkNetCDFCAMReader
  - module: VTK::IONetCDF
- netCDF files generic and CF conventions:
  - Extensions: .nc, .ncdf
  - reader: vtkNetCDFReader
  - writer: vtkNetCDFCFWriter
  - module: VTK::IONetCDF
- UGRID NetCDF (Unstructured):
  - Extensions: .nc, .ncdf
  - reader: vtkNetCDFUGRIDReader
  - module: VTK::IONetCDF
- MPAS NetCDF (Unstructured):
  - Extensions: .nc, .ncdf
  - reader: vtkMPASReader
  - module: VTK::IONetCDF
- SLAC Data Reader:
  - Extensions: .nc, .ncdf
  - reader: vtkSLACReader
  - module: VTK::IONetCDF
- Particle data file used at SLAC:
  - Extensions: .ncdf, .netcdf
  - reader: vtkSLACParticleReader
  - module: VTK::IONetCDF

- GAMBIT GAMBIT ASCII format:
  - Extension: .neu
  - reader: vtkGAMBITReader
  - module: VTK::IOGeometry
- NIfTI-1 and NIfTI-2 medical image files:
  - Extensions: .nii, .img, .hdr
  - reader: vtkNIFTIImageReader
  - writer: vtkNIFTIImageWriter
  - module: VTK::IOImage
- Nrrd Raw Image Files:
  - Extensions: .nrrd, .nhdr
  - reader: vtkNrrdReader
  - module: VTK::IOImage
- MNI surface mesh files:
  - Extension: .obj
  - reader: vtkMNIObjectReader
  - writer: vtkMNIObjectWriter
  - module: VTK::IOMINC
- OggTheora:
  - Extension: .ogv
  - writer: vtkOggTheoraWriter
  - module: VTK::IOOggTheora
- OME TIFF files:
  - Extensions: .ome.tif, .ome.tiff
  - reader: vtkOMETIFFReader
  - module: VTK::IOImage
- OMF:
  - Extension: .omf
  - reader: vtkOMRReader
  - module: VTK::IOOMF
- PIO (Parallel Input Output) data files:
  - Extension: .pio
  - reader: vtkPIOReader
  - module: VTK::IOPIO
- Stanford University PLY format:
  - Extension: .ply

  - reader: vtkPLYReader
  - writer: vtkPLYWriter
  - module: VTK::IOPLY
- PNG file:
  - Extension: .png
  - reader: vtkPNGReader
  - writer: vtkPNGWriter
  - module: VTK::IOImage
- pnm (i.e., portable anymap) file:
  - Extensions: .pnm, .pgm, .ppm
  - reader: vtkPNMReader
  - writer: vtkPNMWriter
  - module: VTK::IOImage
- POP Ocean NetCDF (Rectilinear):
  - Extension: .pop.ncdf .pop.nc
  - reader: vtkNetCDFPOPReader
  - module: VTK::IONetCDF
- PostScript file:
  - Extension: .ps
  - writer: vtkPostScriptWriter
  - module: VTK::IOImage
- SEG-Y:
  - Extensions: .sgy, .segy
  - reader: vtkSegYReader
  - module: VTK::IOSegY
- SLC volume file:
  - Extension: .slc
  - reader: vtkSLCReader
  - module: VTK::IOImage
- VTK Reader for STEP and IGES files using OpenCASCADE:
  - Extensions: .step, .iges
  - reader: vtkOCCTReader
  - module: VTK::IOOCCT
- MNI tag files:
  - Extension: .tag
  - reader: vtkMNITagPointReader

  - writer: vtkMNITagPointWriter
  - module: VTK::IOMINC
- Targa files:
  - Extension: .tga
  - reader: vtkTGAReader
  - module: VTK::IOImage
- Tiff image format:
  - Extensions: .tif, .tiff
  - reader: vtkTIFFReader
  - writer: vtkTIFFWriter
  - module: VTK::IOImage
- OpenVDB:
  - Extension: .vdb
  - reader: vtkOpenVDBReader
  - writer: vtkOpenVDBWriter
  - module: VTK::IOOpenVDB
- VPIC:
  - Extension: .vpc
  - reader: vtkVPCIReader
  - module: VTK::IOVPIC
- MNI transformation files:
  - Extension: .xfm
  - reader: vtkMNITransformReader
  - writer: vtkMNITransformWriter
  - module: VTK::IOMINC
- GE Signa ximg files:
  - Extension: .ximg
  - reader: vtkGESignaReader
  - module: VTK::IOImage
- XDMF (eXtensible Data Model and Format):
  - Extensions: .xmf, .xdmf, .xmf2, .xdmf2
  - reader: vtkXdmfReader
  - writer: vtkXdmfWriter
  - module: VTK::IOXdmf2
- XDMF (eXtensible Data Model and Format):
  - Extensions: .xmf, .xdmf, .xmf3, .xdmf3

- reader: vtkXdmf3Reader
- writer: vtkXdmf3Writer
- module: VTK::IOXdmf3

CHAPTER FIVE

SUPPORTED HARDWARE

VTK can integrate with a number of specialized visualization hardware including:

- Looking Glass, see the latest blog post here. The integration is achieved using an external vtk module that leverages the display's SDK.
- Virtual Reality headsets like Oculus and VIVE as described in this post via the VTK::RenderingOpenVR module.
- Augmented Reality headsets like Hololens as demonstrated here via the VTK::RenderingOpenXRRemoting module.
- Augmented Reality displays like ZSpace via its ParaView integration as demonstrated here.

CHAPTER SIX

MODULES

6.1 VTK::DomainsMicroscopy

6.1.1 vtkOpenSlideReader

- A new image reader for vtk
- Wraps open source openslide library which implements read support for many whole slide image formats
- Mainly from microscopy domain
- Requires openslide libraries for building

6.1.2 Known issues

- Ubuntu 14.04 contains incorrectly patched version of openjpeg (dependency of openslide), and thus openslide is unable to decode certain .svs files. This issue is not present in later versions of ubuntu or fedora 23.

6.2 VTK::FiltersOpenTURNS

This module is based on the OpenTURNS library, which is LGPL licensed. There are some dependencies of OpenTURNS under the GPL license, namely:

- the optional hmat library, under GPL but with an explicit exception for its use within OpenTURNS. This dependency can be deactivated as no part of OpenTURNS used by the VTK module depends on hmat. By the way, the authors of hmat are in the exact same department as the authors of OpenTURNS coming from Airbus
- the poissoninv set of functions for the efficient computation of the Poisson quantile function. This dependency is mandatory and is used within OpenTURNS with a written exception to the GPL license from the author
- the KolmogorovSmirnovDist set of functions for the efficient computation of the exact Kolmogorov-Smirnov distribution. This dependency is mandatory and is used within OpenTURNS with a written exception to the GPL license from the author.

This module (and VTK) cannot be considered to be under the GPL license when using OpenTURNS through the API, since it is an LGPL library which has solved the issue of merging GPL and LGPL code. Thanks to the authors of these dependencies, they are NOT under GPL when used by OpenTURNS!

6.3 VTK::GUISupportQt

There are no restrictions for using this Qt code in any project. To make changes to this code requires Qt 4.5.0 as this was the first version of Qt to be covered under the more liberal LGPL license.

6.4 VTK::IOADIOS2

6.4.1 Goal

Provide readers to data produced by the Adaptable Input Output System version 2, ADIOS2.

Currently used on Paraview Application Server Manager development

Extensions:

- .h = header declaration
- .inl = generic inline template implementations
- .txx = specialized template implementations
- .cxx = implementation

##Public VTK classes:

- **vtkADIOS2ImageCoreReader.h/.cxx** : a generic multiblock reader for image data developed at Kitware Inc. It will use existing arrays to populate dimension of the image, adding timesteps info, point and cell data accordingly. No predefined schema is needed. It can work in serial or MPI mode.
- **vtkADIOS2VTXReader .h/.cxx** : multiblock reader for ImageData and UnstructuredData types using VTK ADIOS2 Readers (VTX) implementation developed at Oak Ridge National Laboratory (ORNL). Reads bp files/streams with a vtk.xml attribute schema the reuses the VTK XML file formats schemas. For more comprehensive documentation refer to this section in the ADIOS2 User Guide.

6.4.2 Core: VTK ADIOS2 CORE READERS

Developed at Kitware Inc

- **vtkADIOS2CoreTypeTraits.h** TypeTraits from adios2 type to vtk type

6.4.3 VTX: VTK ADIOS2 READERS

Developed at Oak Ridge National Laboratory. Reads node (image and unstructured) and cell (unstructured) centered data.

- **common/VTXDataArray .h/.cxx** : wrapper around vtkDataArray with adios2-related relevant information
- **common/VTXHelper .h/.inl/.txx/.cxx** : collection of helper functions used privately in *.cxx
- **common/VTXTypes.h** : header only types definitions including MACROS
- **VTXSchemaManager** : reusable class that manages a reader that is a derived type of VTXSchema
- **schema/VTXSchema .h/.txx/.cxx** : abstract base class for supported schema
  - schema/vtk/VTXvtkBase : Base class for VTK formats
  - schema/vtk/VTXvtkVTI : ImageData VTK format

– schema/vtk/VTXvtkVTU : Unstructured VTK format

6.5 VTK::IOCesium3DTiles

6.5.1 vtk3DTilesWriter - Convert a multiblock dataset to the 3D Tiles format.

Currently, to create a valid 3D Tiles dataset we may need additional conversions: from GLTF to GLB and from GLB to B3DM. We can use JavaScript tools to do these conversions.

Install conversion and validation scripts

- Using node and npm installed on Ubuntu 20.04:
- `cd ~/external/3d-tiles-tools/;npm install 3d-tiles-tools`. Help at: https://github.com/AnalyticalGraphicsInc/3d-tiles-tools/tree/master/tools
- `cd ~/external/gltf-pipeline;npm install gltf-pipeline`. Help at: https://github.com/CesiumGS/gltf-pipeline
- Clone https://github.com/CesiumGS/3d-tiles-samples. and then `npm install`.
- Clone https://github.com/KhronosGroup/glTF-Validator and then follow Building section.

Convert data to GLB or B3DM - Optional

See Testing/Cxx/Test3DTilesWriter for conversions of Jacksonville data stored in OBJs and or Berlin data stored in CityGML. Note that the test saves the 3D Tiles data using GLTF files. If needed, you can use GLB or B3DM, but you'll need to do the following conversions manually: `cd ~/projects/VTK/build/Testing/Temporary/jacksonville-3dtiles/` `cd ~/projects/VTK/build/Testing/Temporary/berlin-3dtiles/`

- Convert gltf to glb

```
find . -name '*.gltf' -exec bash -c 'nodejs ~/external/gltf-pipeline/bin/gltf-pipeline.
↪js -i ${0} -o ${0%.*}.glb' {} \;
find . -name '*.gltf' -exec rm {} \;
find . -name '*.bin' -exec rm {} \;
```

- Check glb validity

```
~/external/glTF-Validator/build/bin/gltf_validator Testing/Temporary/TestGLTFWriter.glb
```

- Convert glb to b3dm

```
find . -name '*.glb' -exec bash -c 'nodejs ~/external/3d-tiles-tools/tools/bin/3d-tiles-
↪tools.js glbToB3dm ${0} ${0%.*}.b3dm' {} \;
find . -name '*.glb' -exec rm {} \;
```

View in Cesium

1. Use 3d-tiles-samples

- Link the tileset created for previous set: `cd ~/external/3d-tiles-samples/tilesets; ln -s ~/projects/VTK/build/Testing/Temporary/jacksonville-3dtiles` `cd ~/external/3d-tiles-samples/tilesets; ln -s ~/projects/VTK/build/Testing/Temporary/berlin-3dtiles`
- Start web server: `cd ..;npm start`

2. `google-chrome jacksonville-3dtiles.html;google-chrome berlin-3dtiles.html`

Test the tilesets using 3d-tiles-validator

```
cd ~/external/3d-tiles-validator/validator/
node ./bin/3d-tiles-validator.js -i ~/projects/VTK/build/Testing/Temporary/jacksonville-
↪3dtiles-points/tileset.json
node ./bin/3d-tiles-validator.js -i ~/projects/VTK/build/Testing/Temporary/jacksonville-
↪3dtiles-colorpoints/tileset.json
```

6.6 VTK::IOFLUENTCFF

This page describes the Fluent CFF IO functionality.

6.6.1 vtkFLUENTCFFReader

Provide a reader for the FluentCFF file format. Provide the Fluent CFF Reader (Common Fluid Format).

The reader supports cartesian grid, unstructured grid (poly, tetra, …), 3D/2D, double and single precision files.

Similarly to the legacy reader (`vtkFLUENTReader`), the Fluent CFF reader requires two files: the case file (`.cas.h5`) and the data file (`.dat.h5`).

The Fluent CFF readers uses the HDF library.

It is worth noting that the Fluent CFF file format is the default format in the latest Fluent version and that ANSYS no longer uses the legacy binary or ASCII formats.

Developed by Arthur Piquet and based on the vtkFLUENTReader class from Brian W. Dotson &

6.6.2 Acknowledgments

Developed by Arthur Piquet and based on the `vtkFLUENTReader` class originally developed from Brian W. Dotson & Terry E. Jordan (Department of Energy, National Energy Technology Laboratory) and Douglas McCorkle (Iowa State University).

6.7 VTK::IOOCCT

The vtkOCCT module was initially developed by Michael Miggliore and Mathieu Westphal in the F3D project, under BSD 3-Clause License. Copyright: Michael Migliore and Mathieu Westphal

6.8 VTK::IOXDMF2

The IO/Xdmf2 directory contains a reduced distribution of the xdmf2/vtk (pv branch) source tree with only the library source code needed by VTK. It is not a submodule; the actual content is part of our source tree and changes can be made and committed directly.

We update from upstream using Git's "subtree" merge strategy. A special branch contains commits of upstream xdmf2/vtk snapshots and nothing else. No Git ref points explicitly to the head of this branch, but it is merged into our history.

Update xdmf2/vtk from upstream as follows. Create a local branch to explicitly reference the upstream snapshot branch head:

```
git branch xdmf2vtk-upstream f40916ae
```

Use a temporary directory to checkout the branch:

```
mkdir xdmf2vtk-tmp cd xdmf2vtk-tmp git init git pull .. xdmf2vtk-upstream rm -rf *
```

Now place the (reduced) xdmf2/vtk content in this directory. See instructions shown by

```
git log f40916ae
```

for help extracting the content from the upstream tarball. Then run the following commands to commit the new version. Substitute the appropriate date and version number:

```
git add –all
```

```
GIT_AUTHOR_NAME='XDMF Developers'
GIT_AUTHOR_EMAIL='xdmf@lists.kitware.com'
GIT_AUTHOR_DATE='2012-08-01 16:14:03 -0500'
git commit -m 'xdmf2/vtk 2012-08-01 (reduced)' && git commit –amend
```

Edit the commit message to describe the procedure used to obtain the content. Then push the changes back up to the main local repository:

```
git push .. HEAD:xdmf2vtk-upstream cd .. rm -rf xdmf2vtk-tmp
```

Create a topic in the main repository on which to perform the update:

```
git checkout -b update-xdmf2 master
```

Merge the xdmf2vtk-upstream branch as a subtree:

```
git merge -s recursive -X subtree=IO/Xdmf2
xdmf2vtk-upstream
```

If there are conflicts, resolve them and commit. Build and test the tree. Commit any additional changes needed to succeed.

Finally, run

```
git rev-parse –short=8 xdmf2vtk-upstream
```

to get the commit from which the xdmf2vtk-upstream branch must be started on the next update. Edit the "git branch xdmf2vtk-upstream" line above to record it, and commit this file.

6.9 VTK::RenderingOpenVR

The `OpenVR` module aims to support PC-based rendering to virtual reality headsets via Valve's `OpenVR` API.

The OpenVR standard has been succeeded by the industry-wide OpenXR standard. See the VTK *`OpenXR` module* for modernized support. The VTK `OpenVR` module is preserved for legacy support.

6.9.1 Supported Devices

Any device that renders with OpenGL and runs from the OpenVR runtime is theoretically supported. Devices include:

- HTC Vive (, Pro)
- Valve Index
- Oculus Rift (, S)
- Meta Quest (1,2,3,Pro)
  - Quest Link or Air Link only)
- HP Reverb G2

6.9.2 Supported Controllers

The VTK `OpenVR` module provides bindings for the following controllers:

- HP Motion Controller json
- Valve Knuckles json
- Oculus Touch json
- HTC Vive Controller json

The VTK `OpenVR` module is considered legacy and not under active development. Please see the VTK `OpenXR` module for support for additional controllers and alternate input mechanisms.

6.9.3 Testing

A minimum OpenVRCone example is available for download on the VTK Examples website.

Tests in the `Testing/Cxx` directory may also be run to demonstrate VTK `RenderingOpenVR` capabilities.

6.10 VTK::RenderingOpenXR

The `OpenXR` module aims to support rendering to a variety of mixed reality devices under the OpenXR industry-wide standard. Detailed information on the OpenXR specification and compliant OpenXR runtimes may be found on the Khronos Group website.

6.10.1 Supported Devices

The OpenXR standard is implemented by most PC-based OpenXR runtimes and devices. The VTK `OpenXR` module aims to support most devices that implement the OpenXR specification and support OpenGL rendering.

The list of possible XR device targets is extensive and constantly expanding. At the time of writing, theoretically supported devices include but are not limited to the following:

- Valve Index
- HTC Vive (, Pro)
- Meta Quest (1,2,3,Pro)
  - Quest Link or Air Link only
- HP Reverb G2

Supported input devices and mechanisms include the following:

- HP Mixed Reality Controller json
- HTC Vive Controller json
- KHR Simple Controller json
- Valve Knuckles json
- Microsoft Hand Interaction json

The `OpenXR` module is commonly tested with the Valve Index and HTC Vive virtual reality headsets.

6.10.2 Adding New Devices

It may be necessary to tell VTK how to handle inputs from a new OpenXR-compatible device. Consider contributing a new JSON input binding specification to add support for a new XR device to the VTK `OpenXR` module.

The OpenXR interaction profile specification is documented here:

https://registry.khronos.org/OpenXR/specs/1.0/html/xrspec.html#semantic-path-interaction-profiles

Input binding JSON files should be added to `OpenXR` and set as default in `vtk_openxr_actions.json`json.

6.10.3 Building

The `OpenXR` module depends on the `OpenXR-SDK` library. `OpenXR-SDK` can be built with CMake via the steps below:

```
> git clone git@github.com:KhronosGroup/OpenXR-SDK.git
> mkdir OpenXR-SDK-build
> cd OpenXR-SDK-build
OpenXR-SDK-build > cmake ../OpenXR-SDK
OpenXR-SDK-build > cmake --build . --config "Release"
```

The `OpenXR` is turned off in VTK by default. Run the following steps to build VTK with OpenXR:

```
VTK-build > cmake -DVTK_MODULE_ENABLE_VTK_RenderingOpenXR:STRING=YES -DOpenXR_INCLUDE_
↪DIR:PATH="path/to/OpenXR-SDK/include/openxr" -DOpenXR_LIBRARY:FILEPATH="path/to/OpenXR-
↪SDK-build/src/loader/Release/openxr_loader.lib" path/to/VTK
VTK-build > cmake --build . --config "Release"
```

6.10.4 Testing

Minimum OpenXR examples are available in the `Testing/Cxx` directory for testing.

To run OpenXR tests, first build VTK with testing enabled.

```
VTK-build > cmake -DVTK_BUILD_TESTING:BOOL=ON path/to/VTK
VTK-build > cmake --build . --config "Release"
```

Then run the test with CTest.

```
VTK-build > ctest -C Release -R <name_of_test>
```

6.10.5 Additional Notes

See *VTK `OpenXRRemoting` documentation* for information on virtual reality rendering to DirectX devices such as the Microsoft HoloLens 2.

Some non-OpenGL devices may be compatible with the WebGL and WebXR specifications. If your XR device does not support OpenGL or OpenXR, we suggest visiting VTK.js WebXR documentation for a web-driven solution.

6.11 VTK::RenderingOpenXRRemoting

6.11.1 VTK - OpenXR Holographic Remoting

Holographic remoting consists in a player application running on the XR device, and a VTK-based remote application running on a standard Windows machine. The remote application receives camera information and rendering resources from the player. It renders the VTK scene before streaming back the resulting texture to the player application. This way we avoid the need to build VTK for Universal Windows Platform (UWP), and we can also keep using VTK's OpenGL-based rendering pipeline. Still, DirectX must be used to fill the texture to be streamed back to the Hololens. This is possible by creating a texture shared by both a DirectX and an OpenGL context, thanks to the NV_DX_interop extension available on almost every recent GPU.

At this time holographic remoting is supported only for the Microsoft HoloLens 2 virtual reality headset.

Player application

- Download the Microsoft MixedReality HolographicRemoting samples and follow the instruction to build the player application. The version number in the branch name **must** match the version of the `Microsoft.Holographic.Remoting.OpenXr` package used below by the remote application.
- Follow the instructions to deploy the player application to the Hololens 2. Alternatively, if you don't have access to a device, you can use the Hololens emulator.
- When the player is deployed, you should see the following message: `Waiting for connection on XX.XX.XX.XX` where *XX.XX.XX.XX* describes the IP address the remote application should connect to.

Remote application

- Enable the CMake option `VTK_MODULE_ENABLE_VTK_RenderingOpenXRRemoting` when building VTK.
- Set the `OpenXRRemoting_BIN_DIR` and `OpenXRRemoting_INCLUDE_DIR` to provide the path to the OpenXR Remoting headers and binary directory. The `Microsoft.Holographic.Remoting.OpenXr` Nuget package that provides this dependency is available here on nuget.org. The version of the `Microsoft.Holographic.Remoting.OpenXr` package must match the branch name of the player application above.
- When successfully built, run the TestOpenXRRemotingInitialization test by sending the IP displayed in the player application as argument: `vtkRenderingOpenXRRemotingCxxTests.exe "TestOpenXRRemotingInitialization" -playerIP XX.XX.XX.XX` Alternatively, the `VTK_PLAYER_IP` environment variable can be used to specify the IP address to connect to. Make sure to provide the content of the OpenXR Remoting binary directory in the system PATH or next to the executable before running the program.
- To use this feature in your own application, use the OpenXR and OpenXRRemoting dedicated rendering stack: `vtkOpenXRRenderer`, `vtkOpenXRRemotingRenderWindow`, `vtkOpenXRRenderWindowInteractor` and `vtkOpenXRCamera`. The address of the player application to connect to must be set using `vtkOpenXRRemotingRenderWindow::SetRemotingIPAddress("XX.XX.XX.XX")` before starting the interactor. See the TestOpenXRRemotingInitialization test for a complete example.

Troubleshooting:

> The OpenXR runtime fails to create and initialize the XrInstance.

To make sure that the player and remote application are compatible, the version of the Microsoft.Holographic.Remoting.OpenXr package must match the version number of the player application branch name.

> The remote application exits with the following output: WARN| Failed to initialize connection strategy.
> ERR| vtkOpenXRRemotingRenderWindow: Failed to initialize OpenXRManager

The remote application could not find the RemotingXR.json or the Microsoft.Holographic.AppRemoting.OpenXr.dll. Make sure to provide the content of the OpenXR Remoting binary directory in the system PATH or next to the executable.

> When running in the Hololens emulator, the connection fails with the following error displayed in the player: "Transport connection was closed due to the requested video format not being supported"

If you have both an Intel and NVidia GPU in your laptop, try disabling the NVidia GPU temporarily under "Display Adaptors" in the "Device Manager".

> When building the player application from VisualStudio, the Hololens emulator does not appear in the list of machine to deploy to.

Add a new x64 solution platform within VisualStudio and switch the current platform from ARM64 to x64. When building, if you now get the error `module machine type 'x64' conflicts with target machine type 'ARM64'`, then edit the project file to remove all occurrence of `/machine:ARM64`.

6.11.2 Additional Notes

See *VTK OpenXR documentation* for information on virtual reality rendering with OpenGL.

6.12 VTK::RenderingVR

6.12.1 vtkRenderingVR - Virtual reality support for VTK

Introduction

The VR module defines an API and support classes for adding virtual reality support to VTK. The OpenVR and OpenXR modules are both subclassed off of this module. For a list of todos and development issues please see

https://gitlab.kitware.com/vtk/vtk/-/issues/18302

Supported Devices

The VR module aims to support runtimes that implement the OpenXR or OpenVR standards.

See *VTK::RenderingOpenXR documentation* for information on rendering with the modern OpenXR specification.

See *VTK::RenderingOpenVR documentation* for information on rendering with the legacy OpenVR specification.

Coordinate Systems

With VR the transformations between coordinate systems can quickly become confusing. To help with this note that most matrices in the VR code are stored in vtk convention. That is `a = Mx` where x is a column vector in homogeneous coordinates. Matrices are named according to what spaces they transform between. For example PhysicalToLeftEyeMatrix. Some common coordinate systems are listed below in order of coordinate flow.

Note that in vtkMatrix4x4 multiplcations are done from right to left so to compute a matrix from spaces A to C you would do

`vtkMatrix4x4::Multiply4x4(BtoCMatrixInput, AtoBMatrixInput, AtoCMatrixOutput)`

Model -> World -> Physical -> Left/RightEye -> Projection

- Model - what an actor's data is in
- World - common coordinate system for all actors
- Physical - the physical VR space in meters with 0,0,0 being the center of the floor of the room
- Device - the viewpoint (position and orientation) of a device such as a controller
- LeftEye (and RightEye) - the viewpoint of the left and right eye
- Projection - in clip space, the expected output space for vertex shaders

The matrices that go between these spaces are as follows and they can be inverted as desired. You will also find some additional matrices that combine some of these transformations into a single matrix for convenience such as VRHMDCamera->WorldToLeftEyeMatrix.

- Model -> World = the actor's matrix
- World -> Physical = inverse of VRRenderWindow->GetPhysicalToWorldMatrix()
- Physical -> LeftEye = VRHMDCamera->PhysicalToLeftEyeMatrix

- LeftEye -> Projection = VRHMDCamera->LeftEyeToProjectionMatrix
- Physical -> Device = inverse of VRRenderWindow->GetDeviceToPhysicalMatrixForDevice()

There are some other matrices used in the camera that are stored in OpenGL format (transpose of VTK format) using an older naming convention. These are names such as WCDCMatrix, the names correspond to

- MC = model coordinates (same as above)
- WC = world coordinates (same as above)
- VC = view coordinates, world coordinates translated and rotated to the camera, similar to the LeftEye space
- DC = device coordinates (device in this context is a GPU, so same as projection coordinates above)

6.13 VTK::RenderingWebGPU

6.13.1 vtkRenderingWebGPU - WebGPU backend for rendering

Description

This module contains the WebGPU native backend for `RenderingCore`. At the moment, only polygonal geometry can be rendered in different representations with point/cell scalar mapped colors.

Available features

Here is a list of currently implemented features:

1. Polygonal geometry rendering with point, line and triangle primitives.
2. Point scalar mapped coloring of surfaces.
3. Cell scalar mapped coloring.
4. Draw actors with the actor representation = `VTK_POINTS`, `VTK_WIREFRAME`, `VTK_SURFACE` and `VTK_SURFACE` with edge visibility.
5. Lighting based on VTK headlights and point/cell normals.
6. Point size adjustments.
7. Line width adjustments for wireframe and surface with edges.
8. `vtkSDL2WebGPURenderWindow` is a reference implementation of `vtkWebGPURenderWindow` that works on WebAssembly and desktop.
9. `vtkXWebGPURenderWindow` is an implementation of `vtkWebGPURenderWindow` that uses X11 for Linux desktop rendering.
10. Depth testing.

Future work

Since WebGPU is already an abstraction over graphics APIs, this module doesn't create another level of abstraction. It uses WebGPU's C++ flavor for it's object-oriented API and RAII. There are helper classes in the `vtkWebGPUInternals...` files for convenience and to make the bind group initialization code look clean.

A lot of work remains to be done. Selections, volume mappers, textures, dual-depth peeling, fancy lights, platform native render windows are few that come to mind.

References

Here are some very interesting references to learn WebGPU from examples if you prefer code over spec.

1. https://toji.github.io/webgpu-gltf-case-study/ A case-study that slowly builds up an efficient gltf renderer in WebGPU using javascript. The author describes downfalls in certain methods and proposes alternative ways when applicable.
2. https://github.com/samdauwe/webgpu-native-examples A curated list of single file examples if you want to see how to do X with Y like constraints using WebGPU C API.
3. https://eliemichel.github.io/LearnWebGPU/index.html Similar to LearnOpenGL or the vulka-tutorial.com. Walks you through getting a window, triangle, buffers, textures and 3D rendering. This tutorial has good coverage and the author provides a simple to use WebGPU C++ distribution.
4. https://sotrh.github.io/learn-wgpu/ A very nice coverage of the beginner concepts of webgpu. This tutorial uses wgpu.rs
5. https://alain.xyz/blog/raw-webgpu Another small tutorial that lets you break the ice with WebGPU and get comfy with the concepts. This tutorial targets javascript API.
6. https://carmencincotti.com/2022-12-19/how-to-render-a-webgpu-triangle-series-part-three-video/ A detailed, yet fun to read explanation of the swapchain and image presentation process. The author has several other targeted posts on WebGPU concepts.
7. https://webgpu.rocks/ You want to look at the WebGPU API, but are afraid of reading the spec and do not want to read C headers. This website presents the WebGPU API and WGSL summary in a fancy way with syntax highlights.

Finally, for wgsl, the spec does a good job https://www.w3.org/TR/WGSL/

How to build VTK with Dawn (Highly experimental)

Things you'll need:

1. git
2. depot_tools

This module uses Dawn-C++ WebGPU implementation when VTK is built outside emscripten. First grab Dawn and follow their build instructions using `gn`, not CMake.

To build VTK with Dawn, you need to build Dawn at commit 3a00a9e5c4179d789cfe89ba09c329b57d39f947. Subsequent commits have changed the public API of Dawn to a great extent, making it incompatible with VTK. Dawn uses the Chromium build system and dependency management so you need to install depot_tools and add it to the PATH.

```
# Clone the repo as "dawn"
git clone https://dawn.googlesource.com/dawn dawn && cd dawn
git checkout 3a00a9e5c4
```

(continues on next page)

(continued from previous page)

```
# Bootstrap the gclient configuration
cp scripts/standalone.gclient .gclient

# Fetch external dependencies and toolchains with gclient
gclient sync
```

Build Dawn with `gn` and Ninja

It is important to set `is_component_build=true`. Otherwise the dawn native shared libraries will not be built.

```
mkdir -p out/Debug
gn gen out/Debug --target_cpu="x64" --args="is_component_build=true is_debug=true is_
↪clang=true"
autoninja -C out/Debug
```

Configure and build VTK

```
$ cmake \
-S /path/to/vtk/src \
-B /path/to/vtk/build \
-GNinja \
-DVTK_ENABLE_WEBGPU=ON \
-DDAWN_SOURCE_DIR=/path/to/dawn/src \
-DDAWN_INCLUDE_DIR=/path/to/dawn/include \
-DDAWN_BINARY_DIR=/path/to/dawn/out/Debug \
-DVTK_BUILD_TESTING=ON

$ cmake --build
```

Run the WebGPU tests

These are not regression tested with image comparisons.

```
$ ./bin/vtkRenderingWebGPUCxxTests
Available tests:
  0. TestCellScalarMappedColors
  1. TestConesBenchmark
  2. TestLineRendering
  3. TestPointScalarMappedColors
  4. TestSurfacePlusEdges
  5. TestTheQuad
  6. TestTheQuadPointRepresentation
  7. TestTheQuadWireframeRepresentation
  8. TestTheTriangle
  9. TestTheTrianglePointRepresentation
 10. TestTheTriangleWireframeRepresentation
```

(continues on next page)

(continued from previous page)

```
11. TestVertexRendering
12. TestWireframe
```

Run the Rendering Core tests

The RenderingCore vtk.module can be edited to link the unit tests with `VTK::RenderingWebGPU` module. After uncommenting the module name under `TEST_DEPENDS`, rebuild and run the tests. Very few of these pass.

```
$ export VTK_GRAPHICS_BACKEND=WEBGPU
$ ./bin/vtkRenderingCoreCxxTests
```

6.14 VTK::WrappingPythonCore

6.14.1 Python Wrapper Core Classes

This directory provides the core support classes that are used by VTK's Python wrappers. The classes can be split into two broad categories: the *PyVTK* classes provide C APIs for Python types, while the *vtkPython* classes are C++ utility classes.

The Python Classes

PyVTKObject

This defines APIs for creating and managing *PyVTKClass* objects, which are Python extension types that wrap vtkObjectBase-derived classes, and *PyVTKObject* objects, which are instances of the these extension types.

PyVTKSpecialObject

Similarly, *PyVTKSpecialType* objects are Python extension types that wrap C++ classes that are *not* derived from vtkObjectBase, and *PyVTKSpecialObject* wraps the instances. These object are reference counted on the Python side, but not on the C++ side. In general they are lightweight objects that are cheap to copy.

PyVTKTemplate

These objects represent C++ class templates. The wrappers instantiate the templates over a limited set of template parameters, and *PyVTKTemplate* is a container for the template instantiations. It is implemented as a dictionary that maps template parameters to template instantiations.

PyVTKEnum

This provides an API for managing subtypes of the Python *int* type that represent named C++ enum types.

PyVTKNamespace

This provides an API for managing subtypes of the Python *module* type that represent C++ namespaces.

PyVTKReference

Python does not support C++-style pass-by-reference, but pass-by-reference can be simulated by passing a typed container whose contents can be modified. The *PyVTKReference* type defines such containers. Within Python, this type can be accessed as *vtkmodules.vtkCommonCore.reference*.

PyVTKMethodDescriptor

In Python, a method descriptor is an object that customizes method lookup, specifically it customizes *object.method* and *class.method* method access. The *PyVTKMethodDescriptor* customizes the access of *PyVTKClass* methods. It handles bound method calls, unbound method calls, static method calls, and calls to overloaded methods.

PyVTKExtras

This one is not actually a class, it is a helper function that adds utility methods and types like the previously-mentioned *reference* type to the vtkCommonCore module. Everything in this file becomes part of vtkCommonCore.

The C++ Classes

vtkPythonUtil

This is a singleton that keeps track of all the vtk-python extension modules that have been loaded, and all of the vtk-python objects that have been instantiated. It contains all of the machinery that is needed for moving VTK objects from C++ to Python and back again.

vtkPythonCommand

This is a subclass of vtkCommand that allows Python methods to be used as VTK observer callbacks.

vtkPythonArgs

When a method call is performed in the wrappers, vtkPythonArgs does the conversion of the arguments from Python to C++, and it also converts the return value from C++ to Python.

vtkPythonOverload

When an overloaded method is called from Python, this class uses the method arguments to decide which overload to use.

vtkPythonCompatibility

This is actually just a header, not a class. It contains macros that make it easier to write code that is compatible with different versions of the Python C API.

vtkSmartPyObject

Whereas the other classes in this directory are for using VTK C++ objects through Python, this class is for using Python objects through C++. This class is a C++ smart pointer that handles Python reference counting.

VTK library is a dynamic C++ toolkit built around the concept of "modules". Each module may have dependencies to other VTK module or external libraries.

Foundational dependencies have been wrapped into convenient "module".

6.15 Enabling or Disabling Modules

To enable a module set

```
cmake -DVTK_MODULE_ENABLE_<module name>=WANT ...
```

during the *configuration* stage.

Disabling a module can be done as follows:

```
cmake -DVTK_MODULE_ENABLE_<module name>=DONT_WANT ...
```

Enabling a module may cause more to be enabled due to dependencies. For more details about the module infrastructure in VTK see the *Module System* section.

6.16 Available Modules

Here is a complete list of the available vtk modules:

Module Name	Description
VTK::AcceleratorsVTKmCore	VTKm data structures
VTK::AcceleratorsVTKmDataModel	VTKm data structures
VTK::AcceleratorsVTKmFilters	VTKm filters and algorithms
VTK::ChartsCore	Charts and plots
VTK::CommonArchive	
VTK::CommonColor	Color palette and named color support classes
VTK::CommonComputationalGeometry	Parametric splines and curves
VTK::CommonCore	The base VTK library

continues on next page

Table 1 – continued from previous page

Module Name	Description
VTK::CommonDataModel	Core data types
VTK::CommonExecutionModel	Core algorithms and execution
VTK::CommonMath	Linear algebra types
VTK::CommonMisc	Assorted utility classes
VTK::CommonPython	
VTK::CommonSystem	Filesystem and networking support
VTK::CommonTransforms	Linear algebra transformations
VTK::DICOMParser	
VTK::DomainsChemistry	Algorithms used in chemistry
VTK::DomainsChemistryOpenGL2	OpenGL support for chemistry data
VTK::DomainsMicroscopy	File readers for microscopy file formats
VTK::DomainsParallelChemistry	Parallel versions of algorithms used in chemistry
VTK::FiltersAMR	Adaptive mesh refinement filters and algorithms
VTK::FiltersCellGrid	Filters and cell-types for vtkCellGrid objects
VTK::FiltersCore	Common filters for VTK data types
VTK::FiltersExtraction	Filters for selecting subsets data
VTK::FiltersFlowPaths	Filters and algorithms for streamlines
VTK::FiltersGeneral	Filters for transforming data
VTK::FiltersGeneric	Filters for selecting subsets of data at arbitrary points
VTK::FiltersGeometry	Geometric transformation filters
VTK::FiltersGeometryPreview	Filters for creating a preview of the geometry of a dataset.
VTK::FiltersHybrid	
VTK::FiltersHyperTree	Hypertree filters and algorithms
VTK::FiltersImaging	Filters and algorithms for images
VTK::FiltersModeling	
VTK::FiltersOpenTURNS	
VTK::FiltersParallel	
VTK::FiltersParallelDIY2	
VTK::FiltersParallelFlowPaths	
VTK::FiltersParallelGeometry	
VTK::FiltersParallelImaging	
VTK::FiltersParallelMPI	
VTK::FiltersParallelStatistics	
VTK::FiltersParallelVerdict	
VTK::FiltersPoints	
VTK::FiltersProgrammable	
VTK::FiltersPython	
VTK::FiltersReduction	
VTK::FiltersReebGraph	
VTK::FiltersSMP	
VTK::FiltersSelection	
VTK::FiltersSources	
VTK::FiltersStatistics	
VTK::FiltersTemporal	
VTK::FiltersTensor	Filters for tensor manipulation
VTK::FiltersTexture	
VTK::FiltersTopology	
VTK::FiltersVerdict	
VTK::GUISupportMFC	
VTK::GUISupportQt	

continues on next page

Table 1 – continued from previous page

Module Name	Description
VTK::GUISupportQtQuick	
VTK::GUISupportQtSQL	
VTK::GeovisCore	
VTK::GeovisGDAL	
VTK::IOADIOS2	
VTK::IOAMR	
VTK::IOAlembic	
VTK::IOAsynchronous	
VTK::IOCGNSReader	
VTK::IOCONVERGECFD	
VTK::IOCatalystConduit	Catalyst implementation for VTK, including Conduit to/from VTK conversion.
VTK::IOCellGrid	
VTK::IOCesium3DTiles	
VTK::IOChemistry	File readers used in chemistry
VTK::IOCityGML	
VTK::IOCore	
VTK::IOERF	
VTK::IOEnSight	
VTK::IOEngys	Reader for Engys files
VTK::IOExodus	
VTK::IOExport	
VTK::IOExportGL2PS	
VTK::IOExportPDF	
VTK::IOFDS	A module for handling I/O for the Fire Dynamics Simulator (FDS) output format.
VTK::IOFFMPEG	
VTK::IOFLUENTCFF	Reader for the FluentCFF file format
VTK::IOFides	The base Fides reader library
VTK::IOGDAL	
VTK::IOGeoJSON	
VTK::IOGeometry	
VTK::IOH5Rage	
VTK::IOH5part	
VTK::IOHDF	
VTK::IOIOSS	
VTK::IOImage	
VTK::IOImport	
VTK::IOInfovis	
VTK::IOLAS	
VTK::IOLSDyna	
VTK::IOLegacy	
VTK::IOMINC	
VTK::IOMPIImage	
VTK::IOMPIParallel	
VTK::IOMotionFX	
VTK::IOMovie	
VTK::IOMySQL	
VTK::IONetCDF	
VTK::IOOCCT	OCCT bridge for VTK, see README for more info
VTK::IOODBC	
VTK::IOOMF	The base OMF Reader library

continues on next page

Table 1 – continued from previous page

Module Name	Description
VTK::IOOggTheora	
VTK::IOOpenVDB	
VTK::IOPDAL	
VTK::IOPIO	
VTK::IOPLY	
VTK::IOParallel	
VTK::IOParallelExodus	
VTK::IOParallelLSDyna	
VTK::IOParallelNetCDF	
VTK::IOParallelXML	
VTK::IOParallelXdmf3	
VTK::IOPostgreSQL	
VTK::IOSQL	
VTK::IOSegY	
VTK::IOTRUCHAS	
VTK::IOTecplotTable	
VTK::IOVPIC	
VTK::IOVeraOut	
VTK::IOVideo	
VTK::IOXML	
VTK::IOXMLParser	
VTK::IOXdmf2	
VTK::IOXdmf3	
VTK::ImagingColor	
VTK::ImagingCore	
VTK::ImagingFourier	
VTK::ImagingGeneral	
VTK::ImagingHybrid	
VTK::ImagingMath	
VTK::ImagingMorphological	
VTK::ImagingOpenGL2	
VTK::ImagingSources	
VTK::ImagingStatistics	
VTK::ImagingStencil	
VTK::InfovisBoost	
VTK::InfovisBoostGraphAlgorithms	
VTK::InfovisCore	
VTK::InfovisLayout	
VTK::InteractionImage	
VTK::InteractionStyle	
VTK::InteractionWidgets	
VTK::Java	
VTK::ParallelCore	
VTK::ParallelDIY	DIY utility classes to simplify DIY-based filters
VTK::ParallelMPI	
VTK::ParallelMPI4Py	
VTK::Python	
VTK::PythonContext2D	
VTK::PythonInterpreter	
VTK::RenderingAnari	

continues on next page

Table 1 – continued from previous page

Module Name	Description
VTK::RenderingAnnotation	
VTK::RenderingCellGrid	
VTK::RenderingContext2D	
VTK::RenderingContextOpenGL2	
VTK::RenderingCore	
VTK::RenderingExternal	
VTK::RenderingFFMPEGOpenGL2	
VTK::RenderingFreeType	
VTK::RenderingFreeTypeFontConfig	
VTK::RenderingGL2PSOpenGL2	
VTK::RenderingHyperTreeGrid	
VTK::RenderingImage	
VTK::RenderingLICOpenGL2	
VTK::RenderingLOD	
VTK::RenderingLabel	
VTK::RenderingMatplotlib	
VTK::RenderingOpenGL2	
VTK::RenderingOpenVR	
VTK::RenderingOpenXR	
VTK::RenderingOpenXRRemoting	
VTK::RenderingParallel	
VTK::RenderingParallelLIC	
VTK::RenderingQt	
VTK::RenderingRayTracing	
VTK::RenderingSceneGraph	
VTK::RenderingTk	
VTK::RenderingUI	
VTK::RenderingVR	
VTK::RenderingVolume	
VTK::RenderingVolumeAMR	
VTK::RenderingVolumeOpenGL2	
VTK::RenderingVtkJS	
VTK::RenderingWebGPU	
VTK::RenderingZSpace	
VTK::SerializationManager	
VTK::TestingCore	
VTK::TestingDataModel	
VTK::TestingGenericBridge	
VTK::TestingIOSQL	
VTK::TestingRendering	
VTK::UtilitiesBenchmarks	
VTK::ViewsContext2D	
VTK::ViewsCore	
VTK::ViewsInfovis	
VTK::ViewsQt	
VTK::WebAssembly	
VTK::WebCore	
VTK::WebGLExporter	
VTK::WebPython	
VTK::WrappingPythonCore	

continues on next page

Table 1 – continued from previous page

Module Name	Description
VTK::catalyst	
VTK::kwiml	
VTK::metaio	
VTK::mpi	
VTK::octree	
VTK::opengl	
VTK::vtksys	

CHAPTER

SEVEN

BUILDING

This page describes how to build and install VTK. It covers building for development, on both Unix-type systems (Linux, HP-UX, Solaris, macOS), and Windows. Note that Unix-like environments such as Cygwin and MinGW are not officially supported. However, patches to fix problems with these platforms will be considered for inclusion. It is recommended that users which require VTK to work on these platforms to submit nightly testing results for them.

A full-featured build of VTK depends on several open source tools and libraries such as Python, Qt, CGNS, HDF5, etc. Some of these are included in the VTK source itself (e.g., HDF5), while others are expected to be present on the machine on which VTK is being built (e.g., Python, Qt).

VTK supports all of the common generators supported by CMake. The Ninja, Makefiles, and Visual Studio generators are the most well-tested however.

Note that VTK does not support in-source builds, so you must have a build tree that is not the source tree.

7.1 Obtaining the sources

There are two approaches:

Release Download

1. Download the source release `VTK-X.Y.Z.tar.gz` from https://vtk.org/download/.
2. Create a folder for VTK.
3. Extract the contents of the VTK folder in the downloaded archive to the subfolder called `source`

Git Clone

To obtain VTK's sources locally, clone the VTK repository using Git.

Open `Git Bash` on Windows or a terminal on Linux and macOS and execute the following:

```
mkdir -p ~/vtk
git clone --recursive https://gitlab.kitware.com/vtk/vtk.git ~/vtk/source
```

To use the latest features being developed or to make changes and contribute to VTK, download the source using **Git Clone**.

7.2 Prerequisites

VTK only requires a few packages in order to build in general, however specific features may require additional packages to be provided to VTK's build configuration.

Required:

- CMake
  - Version 3.12 or newer, however, the latest version is always recommended. If the system package management utilities do not offer cmake or if the offered version is too old Precompiled binaries available on CMake's download page.
- Supported compiler
  - GCC 4.8 or newer
  - Clang 3.3 or newer
  - Apple Clang 7.0 (from Xcode 7.2.1) or newer
  - Microsoft Visual Studio 2015 or newer
  - Intel 14.0 or newer

7.2.1 Optional Additions

- **ffmpeg** When the ability to write `.avi` files is desired, and writing these files is not supported by the OS, VTK can use the ffmpeg library. This is generally true for Unix-like operating systems. Source code for ffmpeg can be obtained from the website.
- **MPI** To run VTK in parallel, an MPI implementation is required. If an MPI implementation that exploits special interconnect hardware is provided on your system, we suggest using it for optimal performance. Otherwise, on Linux/Mac, we suggest either OpenMPI or MPICH. On Windows, Microsoft MPI is required.
- **Python** In order to use scripting, Python is required. The minimum supported version is **3.4**. The instructions are using the system Python. On Ubuntu/Debian the required package is `python3-dev`. If you use a different Python implementation or a virtual environment make sure the environment you use is activated. On Ubuntu/Debian the required package for creating virtual environments is `python3-venv`.
- **Qt5** VTK uses Qt as its GUI library (if the relevant modules are enabled). Precompiled binaries are available on Qt's website. Note that on Windows, the compiler used for building VTK must match the compiler version used to build Qt. Version **5.9** or newer is required.
- **OSMesa** Off-screen Mesa can be used as a software-renderer for running VTK on a server without hardware OpenGL acceleration. This is usually available in system packages on Linux. For example, the `libosmesa6-dev` package on Debian and Ubuntu. However, for older machines, building a newer version of Mesa is likely necessary for bug fixes and support. Its source and build instructions can be found on its website.

7.3 Creating the Build Environment

Windows

- Install CMake
- Install Visual Studio Community Edition
- During installation select the “desktop development with C++” workload.
- Use “x64 Native Tools Command Prompt” for the installed Visual Studio version to configure with CMake and to build with ninja.
- Get ninja. Unzip the binary and put it in `PATH`. Note that newer Visual Studio releases come with a version of `ninja` already and should already exist in `PATH` within the command prompt.

Linux (Ubuntu/Debian)

Install the following packages:

```
$ sudo apt install \
build-essential \
cmake \
cmake-curses-gui \
mesa-common-dev \
mesa-utils \
freeglut3-dev \
ninja-build
```

macOS

- Install CMake
- Install XCode
- Ensure XCode command line tools are installed:

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

Note: `ninja` is a more efficient alternative to `Makefiles` or Visual Studio solution files. The speed increase is the most noticeable when doing incremental build.

7.4 Configure

In order to build, CMake requires two steps, configure and build. VTK itself does not support what are known as in-source builds, so the first step is to create a build directory.

Windows (Ninja)

Open "x64 Native Tools Command Prompt" for the installed Visual Studio:

```
ccmake -GNinja -S %HOMEPATH%\vtk\source -B %HOMEPATH%\vtk\build
```

Note that CMake GUI must also be launched from the "Native Tools Command Prompt".

Windows (Visual Studio)

Use CMake to generate a Visual Studio solution file (`.sln`).

1. Open CMake GUI, either by typing `cmake-gui` on the command prompt or from the start-menu.
2. Enter the source and build directories
3. Click `[Configure]`
4. You will now get a selection screen in which you can specify your "generator". Select the one you need.
5. We are now presented with a few options that can be turned on or off as desired.
6. Click `[Configure]` to apply the changes.
7. Click `[Generate]`. This will populate the "build" sub-folder.
8. Finally, click `[Open Project]` to open the generated solution in Visual Studio.

Linux/macOS

```
mkdir -p ~/vtk/build
cd ~/vtk/build
ccmake -GNinja ../path/to/vtk/source
```

The parameter `-GNinja` may be skipped to use the default generator (e.g `Unix Makefiles`).

Missing dependencies

CMake may not find all dependencies automatically in all cases. The steps needed to find any given package depends on the package itself.

For general assistance, please see the documentation for `find_package`'s search procedure and the relevant Find module (as available).

Hint: Different features can be enabled/disabled by setting the *Build Settings* during the configure stage.

7.5 Building

To build VTK:

Windows (Ninja)

```
cmake --build %HOMEPATH%\vtk\build --config Release
```

Windows (Visual Studio)

Open the generated solution file.

1. Set the configuration to "Release"
2. On the menu bar, choose `Build`, and then choose `Build Solution`.

Linux/macOS

```
cmake --build ~/vtk/build
```

7.5.1 Build Settings

VTK has a number of settings available for its build. The common variables to modify include:

- `BUILD_SHARED_LIBS` (default `ON`): If set, shared libraries will be built. This is usually what is wanted.
- `VTK_USE_CUDA` (default `OFF`): Whether CUDA support will be available or not.
- `VTK_USE_MPI` (default `OFF`): Whether MPI support will be available or not.
- `VTK_WRAP_PYTHON` (default `OFF`; requires `VTK_ENABLE_WRAPPING`): Whether Python support will be available or not.

Less common, but variables which may be of interest to some:

- `VTK_BUILD_EXAMPLES` (default `OFF`): If set, VTK's example code will be added as tests to the VTK test suite.
- `VTK_ENABLE_LOGGING` (default `ON`): If set, enhanced logging will be enabled.
- `VTK_LOGGING_TIME_PRECISION` (default `3`; requires `VTK_ENABLE_LOGGING`): Change the precision of times output when `VTK_ENABLE_LOGGING` is on.
- `VTK_BUILD_TESTING` (default `OFF`): Whether to build tests or not. Valid values are `OFF` (no testing), `WANT` (enable tests as possible), and `ON` (enable all tests; may error out if features otherwise disabled are required by test code).
- `VTK_ENABLE_KITS` (default `OFF`; requires `BUILD_SHARED_LIBS`): Compile VTK into a smaller set of libraries. Can be useful on platforms where VTK takes a long time to launch due to expensive disk access.
- `VTK_ENABLE_WRAPPING` (default `ON`): Whether any wrapping support will be available or not.
- `VTK_WRAP_JAVA` (default `OFF`; requires `VTK_ENABLE_WRAPPING`): Whether Java support will be available or not.
- `VTK_WRAP_SERIALIZATION` (default `OFF`; requires `VTK_ENABLE_WRAPPING`): Whether serialization code will be auto generated or not.

- `VTK_SMP_IMPLEMENTATION_TYPE` (default `Sequential`): Set which SMPTools will be implemented by default. Must be either `Sequential`, `STDThread`, `OpenMP` or `TBB`. The backend can be changed at runtime if the desired backend has his option `VTK_SMP_ENABLE_<backend_name>` set to `ON`.
- `VTK_ENABLE_CATALYST` (default `OFF`): Enable catalyst-dependent modules including the VTK catalyst implementation. Depends on an external Catalyst.

OpenGL-related options:

Note that if OpenGL is used, there must be a "sensible" setup. Sanity checks exist to make sure a broken build is not being made. Essentially:

- at least one rendering environment (X, Cocoa, SDL2, OSMesa, EGL, etc.) must be available;
- OSMesa and EGL conflict with each other; and
- OSMesa only supports off-screen rendering and is therefore incompatible with Cocoa, X, and SDL2.
  - `VTK_USE_COCOA` (default `ON`; requires macOS): Use Cocoa for render windows.
  - `VTK_USE_X` (default `ON` for Unix-like platforms except macOS, iOS, and Emscripten, `OFF` otherwise): Use X for render windows.
  - `VTK_USE_SDL2` (default `ON` for Emscripten, `OFF` otherwise): Use SDL2 for render windows.
  - `VTK_OPENGL_HAS_OSMESA` (default `OFF`): Use to indicate that the OpenGL library being used supports offscreen Mesa rendering (OSMesa).
  - `VTK_OPENGL_USE_GLES` (default `OFF`; forced `ON` for Android): Whether to use OpenGL ES API for OpenGL or not.
  - `VTK_OPENGL_HAS_EGL` (default `ON` for Android, `OFF` otherwise): Use to indicate that the OpenGL library being used supports EGL context management.
  - `VTK_DEFAULT_EGL_DEVICE_INDEX` (default `0`; requires `VTK_OPENGL_HAS_EGL`): The default EGL device to use for EGL render windows.
  - `VTK_ENABLE_WEBGPU` (default `OFF`; required if using Emscripten): Enable WebGPU rendering support.
  - `VTK_DEFAULT_RENDER_WINDOW_OFFSCREEN` (default `OFF`): Whether to default to offscreen render windows by default or not.
  - `VTK_USE_OPENGL_DELAYED_LOAD` (default `OFF`; requires Windows and CMake >= 3.13): If set, use delayed loading to load the OpenGL DLL at runtime.
  - `VTK_DEFAULT_RENDER_WINDOW_HEADLESS` (default `OFF`; only available if applicable): Default to a headless render window.
  - `VTK_USE_WIN32_OPENGL` (default `ON` for Windows, forced `OFF` otherwise): Use Win32 APIs for render windows (typically only relevant for OSMesa on Windows builds).

More advanced options:

- `VTK_ABI_NAMESPACE_NAME` (default `<DEFAULT>` aka `""`): If set, VTK will wrap all VTK public symbols in an `inline namespace <VTK_ABI_NAMESPACE_NAME>` to allow runtime co-habitation with different VTK versions. Some C ABIs are also wrapped in this namespace using macro expansion `#define c_abi VTK_ABI_NAMESPACE_MANGLE(c_abi)`
- `VTK_ABI_NAMESPACE_ATTRIBUTES` (default `<DEFAULT>` aka `""`): If set, VTK will inject these attributes into the `inline namespace`. i.e. `inline namespace <VTK_ABI_NAMESPACE_ATTRIBUTES> <VTK_ABI_NAMESPACE_NAME>` The `VTK_ABI_NAMESPACE_ATTRIBUTES` is only applied the the APIs inside of the namespace, not to C APIs.
- `VTK_BUILD_DOCUMENTATION` (default `OFF`): If set, VTK will build its API documentation using Doxygen.

- VTK_BUILD_SPHINX_DOCUMENTATION (default OFF): If set, VTK will build its sphinx documentation website.
- VTK_BUILD_ALL_MODULES (default OFF): If set, VTK will enable all modules not disabled by other features.
- VTK_ENABLE_REMOTE_MODULES (default ON): If set, VTK will try to build remote modules (the Remote directory). If unset, no remote modules will build.
- VTK_ENABLE_EXTRA_BUILD_WARNINGS (default OFF; requires CMake >= 3.19): If set, VTK will enable additional build warnings.
- VTK_ENABLE_EXTRA_BUILD_WARNINGS_EVERYTHING (default OFF; requires VTK_ENABLE_EXTRA_BUILD_WARNINGS and -Weverything support): If set, VTK will enable all build warnings (with some explicitly turned off).
- VTK_USE_EXTERNAL (default OFF): Whether to prefer external third party libraries or the versions VTK's source contains.
- VTK_TARGET_SPECIFIC_COMPONENTS (default OFF): Whether to install files into target-specific components (<TARGET>-runtime, <TARGET>-development, etc.) or general components (runtime, development, etc.)
- VTK_VERSIONED_INSTALL (default ON): Whether to add version numbers to VTK's include directories and library names in the install tree.
- VTK_CUSTOM_LIBRARY_SUFFIX (default depends on VTK_VERSIONED_INSTALL): The custom suffix for libraries built by VTK. Defaults to either an empty string or X.Y where X and Y are VTK's major and minor version components, respectively.
- VTK_INSTALL_SDK (default ON): If set, VTK will install its headers, CMake API, etc. into its install tree for use.
- VTK_FORBID_DOWNLOADS (default OFF): If set, VTK will error on any network activity required during the build (namely remote modules and testing data).
- VTK_DATA_STORE (default is complicated): If set or detected, points to where VTK external data will be stored or looked up.
- VTK_DATA_EXCLUDE_FROM_ALL (default is complicated, but generally OFF): If set or detected, data downloads will only happen upon explicit request rather than through the build's default target.
- VTK_RELOCATABLE_INSTALL (default ON): If set, the install tree will be relocatable to another path. If unset, the install tree may be tied to the build machine with absolute paths, but finding dependencies in non-standard locations may require work without passing extra information when consuming VTK.
- VTK_UNIFIED_INSTALL_TREE (default OFF): If set, the install tree is stipulated to be a unified install tree of VTK and all of its dependencies; a unified tree usually simplifies things including, but not limited to, the Python module paths, library search paths, and plugin searching. This option is irrelevant if a relocatable install is requested as such setups assume that dependencies are set up either via a unified tree or some other mechanism such as modules).
- VTK_ENABLE_SANITIZER (default OFF): Whether to enable sanitization of the VTK codebase or not.
- VTK_SANITIZER (default address; requires VTK_ENABLE_SANITIZER): The sanitizer to use.
- VTK_USE_LARGE_DATA (default OFF; requires VTK_BUILD_TESTING): Whether to enable tests which use "large" data or not (usually used to reduce the amount of data downloading required for the test suite).
- VTK_USE_HIP (default OFF; requires CMAKE >= 3.21 and NOT VTK_USE_CUDA) Whether HIP support will be available or not.
- VTK_LEGACY_REMOVE (default OFF): If set, VTK will disable legacy, deprecated APIs.
- VTK_LEGACY_SILENT (default OFF; requires VTK_LEGACY_REMOVE to be OFF): If set, usage of legacy, deprecated APIs will not cause warnings.

- VTK_USE_FUTURE_CONST (default OFF): If set, the VTK_FUTURE_CONST macro expands to const; otherwise it expands to nothing. This is used to incrementally add more const correctness to the codebase while making it opt-in for backwards compatibility.
- VTK_USE_FUTURE_BOOL (default OFF): If set, the vtkTypeBool typedef is defined to bool; otherwise it's int. VTK was created before C++ even had bool, and so its oldest code used int. Set to ON to opt in to using more real bools, set to OFF only if required for backwards compatibility.
- VTK_USE_TK (default OFF; requires VTK_WRAP_PYTHON): If set, VTK will enable Tkinter support for VTK widgets.
- VTK_BUILD_COMPILE_TOOLS_ONLY (default OFF): If set, VTK will compile just its compile tools for use in a cross-compile build.
- VTK_SERIAL_TESTS_USE_MPIEXEC (default OFF): Used on HPC to run serial tests on compute nodes. If set, it prefixes serial tests with "${MPIEXEC_EXECUTABLE}" "${MPIEXEC_NUMPROC_FLAG}" "1" ${MPIEXEC_PREFLAGS}
- VTK_WINDOWS_PYTHON_DEBUGGABLE (default OFF): Set to ON if using a debug build of Python.
- VTK_WINDOWS_PYTHON_DEBUGGABLE_REPLACE_SUFFIX (default OFF): Set to ON to use just a _d suffix for Python modules.
- VTK_BUILD_PYI_FILES (default OFF): Set to ON to build .pyi type hint files for VTK's Python interfaces.
- VTK_DLL_PATHS (default "" or VTK_DLL_PATHS from the environment): If set, these paths will be added via Python 3.8's os.add_dll_directory mechanism in order to find dependent DLLs when loading VTK's Python modules. Note that when using the variable, paths are in CMake form (using /) and in the environment are a path list in the platform's preferred format.
- VTK_ENABLE_VR_COLLABORATION (default OFF): If ON, includes support for multi client VR collaboration. Requires libzmq and cppzmq external libraries.
- VTK_SMP_ENABLE_<backend_name> (default OFF if needs an external library otherwise ON): If set, builds with the specified SMPTools backend implementation that can be changed on runtime with VTK_SMP_BACKEND_IN_USE environment variable.
- VTK_USE_VIDEO_FOR_WINDOWS (default OFF; requires Windows): Enable the vtkAVIWriter class in the VTK::IOMovie module.
- VTK_USE_VIDEO_FOR_WINDOWS_CAPTURE (default OFF; requires Windows): Enable the vtkWin32VideoSource class in the VTK::IOVideo module.
- VTK_USE_MICROSOFT_MEDIA_FOUNDATION (default OFF; requires Windows): Enable the vtkMP4Writer class in the VTK::IOMovie module.
- VTK_USE_64BIT_TIMESTAMPS (default OFF; forced on for 64-bit builds): Build with 64-bit vtkMTimeType.
- VTK_USE_64BIT_IDS (default OFF for 32-bit builds; ON for 64-bit builds): Whether vtkIdType should be 32-bit or 64-bit.
- VTK_DEBUG_LEAKS (default OFF): Whether VTK will report leaked vtkObject instances at process destruction or not.
- VTK_DEBUG_RANGE_ITERATORS (default OFF; requires a Debug build): Detect errors with for-range iterators in VTK (note that this is very slow).
- VTK_ALWAYS_OPTIMIZE_ARRAY_ITERATORS (default OFF; requires NOT VTK_DEBUG_RANGE_ITERATORS): Optimize for-range array iterators even in Debug builds.
- VTK_ALL_NEW_OBJECT_FACTORY (default OFF): If ON, classes using vtkStandardNewMacro will use vtkObjectFactoryNewMacro allowing overrides to be available even when not explicitly requested through vtkObjectFactoryNewMacro or vtkAbstractObjectFactoryNewMacro.

- `VTK_ENABLE_VTKM_OVERRIDES` (default `OFF`): If `ON`, enables factory override of certain VTK filters by their VTK-m counterparts. There is also a runtime switch that can be used to enable/disable the overrides at run-time (on by default). It can be accessed using the static function `vtkmFilterOverrides::SetEnabled(bool)`.
- `VTK_GENERATE_SPDX` (default `OFF`): If `ON`, SPDX file will be generated at build time and installed for each module and third party, in order to be able to create a SBOM. See *SPDX files generation* and *SPDX & SBOM* for more info.
- `VTK_ANARI_ENABLE_NVTX` (default `OFF`; requires CUDA Toolkit): If `ON`, enables the NVIDIA Tools Extension Library (NVTX) for profiling the ANARI rendering code and visualizing these events in tools like NSight Systems.

`vtkArrayDispatch` related options:

The `VTK_DISPATCH_<array_type>_ARRAYS` options (default `OFF` for all but AOS) enable the specified type of array to be included in a dispatch type list. Explicit arrays (such as AOS, SOA, Typed, and implicit arrays) are included in the `vtkArrayDispatchTypeList.h` The implicit array framework is included in the `CommonCore` module. The following array types currently exist for use with the VTK dispatch mechanism:

- `VTK_DISPATCH_AOS_ARRAYS` (default `ON`): includes dispatching for the commonly used "array-of-structure" ordered arrays derived from `vtkAOSDataArrayTemplate`
- `VTK_DISPATCH_SOA_ARRAYS` (default `OFF`): includes dispatching for "structure-of-array" ordered arrays derived from `vtkSOADataArrayTemplate`
- `VTK_DISPATCH_TYPED_ARRAYS` (default `OFF`): includes dispatching for arrays derived from `vtkTypedDataArray`
- `VTK_DISPATCH_AFFINE_ARRAYS` (default `OFF`): includes dispatching for linearly varying `vtkAffineArray`s as part of the implicit array framework
- `VTK_DISPATCH_CONSTANT_ARRAYS` (default `OFF`): includes dispatching for constant arrays `vtkConstantArray` as part of the implicit array framework
- `VTK_DISPATCH_STD_FUNCTION_ARRAYS` (default `OFF`): includes dispatching for arrays with an `std::function` backend `vtkStdFunctionArray` as part of the implicit array framework

The outlier in terms of dispatch support is the family of arrays derived from `vtkScaledSOADataArrayTemplate` which are automatically included in dispatch when built setting the `VTK_BUILD_SCALED_SOA_ARRAYS`.

Warning: Adding increasing numbers of arrays in the dispatch mechanism can greatly slow down compile times.

The VTK module system provides a number of variables to control modules which are not otherwise controlled by the other options provided.

- `VTK_MODULE_USE_EXTERNAL_<name>` (default depends on `VTK_USE_EXTERNAL`): Use an external source for the named third-party module rather than the copy contained within the VTK source tree.

  Warning: Activating this option within an interactive cmake configuration (i.e. ccmake, cmake-gui) could end up finding libraries in the standard locations rather than copies in non-standard locations.

  It is recommended to pass the variables necessary to find the intended external package to the first configure to avoid finding unintended copies of the external package. The variables which matter depend on the package being found, but those ending with `_LIBRARY` and `_INCLUDE_DIR` as well as the general CMake `find_package` variables ending with `_DIR` and `_ROOT` are likely candidates.

  Example:

  ```
  ccmake -D HDF5_ROOT:PATH=/home/user/myhdf5 ../vtk/sources
  ```

- `VTK_MODULE_ENABLE_<name>` (default `DEFAULT`): Change the build settings for the named module. Valid values are those for the module system's build settings (see below).
- `VTK_GROUP_ENABLE_<name>` (default `DEFAULT`): Change the default build settings for modules belonging to the named group. Valid values are those for the module system's build settings (see below).

For variables which use the module system's build settings, the valid values are as follows:

- `YES`: Require the module to be built.
- `WANT`: Build the module if possible.
- `DEFAULT`: Use the settings by the module's groups and `VTK_BUILD_ALL_MODULES`.
- `DONT_WANT`: Don't build the module unless required as a dependency.
- `NO`: Do not build the module.

If any `YES` module requires a `NO` module, an error is raised.

CHAPTER

EIGHT

API

8.1 C++

Reference documentation for VTK can be found in the Doxygen Manual.

8.2 Python

8.2.1 Native Python documentation

Python-style documentation is available for the following packages:

`vtkmodules`

Currently, this package is experimental and may change in the future.

Subpackages

`vtkmodules.util`

Utility modules for the VTK-Python wrappers.

Submodules

`vtkmodules.util.vtkImageExportToArray`

vtkImageExportToArray - a NumPy front-end to vtkImageExport

This class converts a VTK image to a numpy array. The output array will always have 3 dimensions (or 4, if the image had multiple scalar components).

To use this class, you must have numpy installed (http://numpy.scipy.org)

Methods

SetInputConnection(vtkAlgorithmOutput) – connect to VTK image pipeline SetInputData(vtkImageData) – set an vtkImageData to export GetArray() – execute pipeline and return a numpy array

Methods from vtkImageExport

GetDataExtent() GetDataSpacing() GetDataOrigin()

Module Contents

Classes

vtkImageExportToArray

API

class vtkmodules.util.vtkImageExportToArray.**vtkImageExportToArray**

Initialization

`__typeDict`

None

`__sizeDict`

None

`SetConvertUnsignedShortToInt`(*yesno*)

`GetConvertUnsignedShortToInt`()

`ConvertUnsignedShortToIntOn`()

`ConvertUnsignedShortToIntOff`()

`SetInputConnection`(*input*)

`SetInputData`(*input*)

`GetInput`()

`GetArray`()

`GetDataExtent`()

`GetDataSpacing`()

`GetDataOrigin`()

`vtkmodules.util.colors`

Module Contents

Data

antique_white
azure
bisque
blanched_almond
cornsilk
eggshell
floral_white
gainsboro
ghost_white
honeydew
ivory
lavender
lavender_blush
lemon_chiffon
linen
mint_cream
misty_rose
moccasin
navajo_white
old_lace
papaya_whip
peach_puff
seashell
snow
thistle
titanium_white
wheat
white
white_smoke
zinc_white
cold_grey
dim_grey
grey
light_grey
slate_grey
slate_grey_dark
slate_grey_light
warm_grey
black
ivory_black
lamp_black
alizarin_crimson
brick
cadmium_red_deep
coral
coral_light
deep_pink
english_red
firebrick
geranium_lake
hot_pink
indian_red

continues on next page

Table 2 – continued from previous page

light_salmon
madder_lake_deep
maroon
pink
pink_light
raspberry
red
rose_madder
salmon
tomato
venetian_red
beige
brown
brown_madder
brown_ochre
burlywood
burnt_sienna
burnt_umber
chocolate
deep_ochre
flesh
flesh_ochre
gold_ochre
greenish_umber
khaki
khaki_dark
light_beige
peru
rosy_brown
raw_sienna
raw_umber
sepia
sienna
saddle_brown
sandy_brown
tan
van_dyke_brown
cadmium_orange
cadmium_red_light
carrot
dark_orange
mars_orange
mars_yellow
orange
orange_red
yellow_ochre
aureoline_yellow
banana
cadmium_lemon
cadmium_yellow
cadmium_yellow_light
gold

continues on next page

Table 2 – continued from previous page

goldenrod
goldenrod_dark
goldenrod_light
goldenrod_pale
light_goldenrod
melon
naples_yellow_deep
yellow
yellow_light
chartreuse
chrome_oxide_green
cinnabar_green
cobalt_green
emerald_green
forest_green
green
green_dark
green_pale
green_yellow
lawn_green
lime_green
mint
olive
olive_drab
olive_green_dark
permanent_green
sap_green
sea_green
sea_green_dark
sea_green_medium
sea_green_light
spring_green
spring_green_medium
terre_verte
viridian_light
yellow_green
aquamarine
aquamarine_medium
cyan
cyan_white
turquoise
turquoise_dark
turquoise_medium
turquoise_pale
alice_blue
blue
blue_light
blue_medium
cadet
cobalt
cornflower
cerulean

continues on next page

Table 2 – continued from previous page

dodger_blue
indigo
manganese_blue
midnight_blue
navy
peacock
powder_blue
royal_blue
slate_blue
slate_blue_dark
slate_blue_light
slate_blue_medium
sky_blue
sky_blue_deep
sky_blue_light
steel_blue
steel_blue_light
turquoise_blue
ultramarine
blue_violet
cobalt_violet_deep
magenta
orchid
orchid_dark
orchid_medium
permanent_red_violet
plum
purple
purple_medium
ultramarine_violet
violet
violet_dark
violet_red
violet_red_medium
violet_red_pale

API

`vtkmodules.util.colors.`**`antique_white`**

(0.9804, 0.9216, 0.8431)

`vtkmodules.util.colors.`**`azure`**

(0.9412, 1.0, 1.0)

`vtkmodules.util.colors.`**`bisque`**

(1.0, 0.8941, 0.7686)

`vtkmodules.util.colors.`**`blanched_almond`**

(1.0, 0.9216, 0.8039)

`vtkmodules.util.colors.`**`cornsilk`**

(1.0, 0.9725, 0.8627)

vtkmodules.util.colors.**eggshell**
(0.99, 0.9, 0.79)

vtkmodules.util.colors.**floral_white**
(1.0, 0.9804, 0.9412)

vtkmodules.util.colors.**gainsboro**
(0.8627, 0.8627, 0.8627)

vtkmodules.util.colors.**ghost_white**
(0.9725, 0.9725, 1.0)

vtkmodules.util.colors.**honeydew**
(0.9412, 1.0, 0.9412)

vtkmodules.util.colors.**ivory**
(1.0, 1.0, 0.9412)

vtkmodules.util.colors.**lavender**
(0.902, 0.902, 0.9804)

vtkmodules.util.colors.**lavender_blush**
(1.0, 0.9412, 0.9608)

vtkmodules.util.colors.**lemon_chiffon**
(1.0, 0.9804, 0.8039)

vtkmodules.util.colors.**linen**
(0.9804, 0.9412, 0.902)

vtkmodules.util.colors.**mint_cream**
(0.9608, 1.0, 0.9804)

vtkmodules.util.colors.**misty_rose**
(1.0, 0.8941, 0.8824)

vtkmodules.util.colors.**moccasin**
(1.0, 0.8941, 0.7098)

vtkmodules.util.colors.**navajo_white**
(1.0, 0.8706, 0.6784)

vtkmodules.util.colors.**old_lace**
(0.9922, 0.9608, 0.902)

vtkmodules.util.colors.**papaya_whip**
(1.0, 0.9373, 0.8353)

vtkmodules.util.colors.**peach_puff**
(1.0, 0.8549, 0.7255)

vtkmodules.util.colors.**seashell**
(1.0, 0.9608, 0.9333)

vtkmodules.util.colors.**snow**
(1.0, 0.9804, 0.9804)

vtkmodules.util.colors.**thistle**
    (0.8471, 0.749, 0.8471)

vtkmodules.util.colors.**titanium_white**
    (0.99, 1.0, 0.94)

vtkmodules.util.colors.**wheat**
    (0.9608, 0.8706, 0.702)

vtkmodules.util.colors.**white**
    (1.0, 1.0, 1.0)

vtkmodules.util.colors.**white_smoke**
    (0.9608, 0.9608, 0.9608)

vtkmodules.util.colors.**zinc_white**
    (0.99, 0.97, 1.0)

vtkmodules.util.colors.**cold_grey**
    (0.5, 0.54, 0.53)

vtkmodules.util.colors.**dim_grey**
    (0.4118, 0.4118, 0.4118)

vtkmodules.util.colors.**grey**
    (0.7529, 0.7529, 0.7529)

vtkmodules.util.colors.**light_grey**
    (0.8275, 0.8275, 0.8275)

vtkmodules.util.colors.**slate_grey**
    (0.4392, 0.502, 0.5647)

vtkmodules.util.colors.**slate_grey_dark**
    (0.1843, 0.3098, 0.3098)

vtkmodules.util.colors.**slate_grey_light**
    (0.4667, 0.5333, 0.6)

vtkmodules.util.colors.**warm_grey**
    (0.5, 0.5, 0.41)

vtkmodules.util.colors.**black**
    (0.0, 0.0, 0.0)

vtkmodules.util.colors.**ivory_black**
    (0.16, 0.14, 0.13)

vtkmodules.util.colors.**lamp_black**
    (0.18, 0.28, 0.23)

vtkmodules.util.colors.**alizarin_crimson**
    (0.89, 0.15, 0.21)

vtkmodules.util.colors.**brick**
    (0.61, 0.4, 0.12)

`vtkmodules.util.colors.`**`cadmium_red_deep`**
(0.89, 0.09, 0.05)

`vtkmodules.util.colors.`**`coral`**
(1.0, 0.498, 0.3137)

`vtkmodules.util.colors.`**`coral_light`**
(0.9412, 0.502, 0.502)

`vtkmodules.util.colors.`**`deep_pink`**
(1.0, 0.0784, 0.5765)

`vtkmodules.util.colors.`**`english_red`**
(0.83, 0.24, 0.1)

`vtkmodules.util.colors.`**`firebrick`**
(0.698, 0.1333, 0.1333)

`vtkmodules.util.colors.`**`geranium_lake`**
(0.89, 0.07, 0.19)

`vtkmodules.util.colors.`**`hot_pink`**
(1.0, 0.4118, 0.7059)

`vtkmodules.util.colors.`**`indian_red`**
(0.69, 0.09, 0.12)

`vtkmodules.util.colors.`**`light_salmon`**
(1.0, 0.6275, 0.4784)

`vtkmodules.util.colors.`**`madder_lake_deep`**
(0.89, 0.18, 0.19)

`vtkmodules.util.colors.`**`maroon`**
(0.6902, 0.1882, 0.3765)

`vtkmodules.util.colors.`**`pink`**
(1.0, 0.7529, 0.7961)

`vtkmodules.util.colors.`**`pink_light`**
(1.0, 0.7137, 0.7569)

`vtkmodules.util.colors.`**`raspberry`**
(0.53, 0.15, 0.34)

`vtkmodules.util.colors.`**`red`**
(1.0, 0.0, 0.0)

`vtkmodules.util.colors.`**`rose_madder`**
(0.89, 0.21, 0.22)

`vtkmodules.util.colors.`**`salmon`**
(0.9804, 0.502, 0.4471)

`vtkmodules.util.colors.`**`tomato`**
(1.0, 0.3882, 0.2784)

vtkmodules.util.colors.**venetian_red**
(0.83, 0.1, 0.12)

vtkmodules.util.colors.**beige**
(0.64, 0.58, 0.5)

vtkmodules.util.colors.**brown**
(0.5, 0.1647, 0.1647)

vtkmodules.util.colors.**brown_madder**
(0.86, 0.16, 0.16)

vtkmodules.util.colors.**brown_ochre**
(0.53, 0.26, 0.12)

vtkmodules.util.colors.**burlywood**
(0.8706, 0.7216, 0.5294)

vtkmodules.util.colors.**burnt_sienna**
(0.54, 0.21, 0.06)

vtkmodules.util.colors.**burnt_umber**
(0.54, 0.2, 0.14)

vtkmodules.util.colors.**chocolate**
(0.8235, 0.4118, 0.1176)

vtkmodules.util.colors.**deep_ochre**
(0.45, 0.24, 0.1)

vtkmodules.util.colors.**flesh**
(1.0, 0.49, 0.25)

vtkmodules.util.colors.**flesh_ochre**
(1.0, 0.34, 0.13)

vtkmodules.util.colors.**gold_ochre**
(0.78, 0.47, 0.15)

vtkmodules.util.colors.**greenish_umber**
(1.0, 0.24, 0.05)

vtkmodules.util.colors.**khaki**
(0.9412, 0.902, 0.549)

vtkmodules.util.colors.**khaki_dark**
(0.7412, 0.7176, 0.4196)

vtkmodules.util.colors.**light_beige**
(0.9608, 0.9608, 0.8627)

vtkmodules.util.colors.**peru**
(0.8039, 0.5216, 0.2471)

vtkmodules.util.colors.**rosy_brown**
(0.7373, 0.5608, 0.5608)

`vtkmodules.util.colors.`**`raw_sienna`**
(0.78, 0.38, 0.08)

`vtkmodules.util.colors.`**`raw_umber`**
(0.45, 0.29, 0.07)

`vtkmodules.util.colors.`**`sepia`**
(0.37, 0.15, 0.07)

`vtkmodules.util.colors.`**`sienna`**
(0.6275, 0.3216, 0.1765)

`vtkmodules.util.colors.`**`saddle_brown`**
(0.5451, 0.2706, 0.0745)

`vtkmodules.util.colors.`**`sandy_brown`**
(0.9569, 0.6431, 0.3765)

`vtkmodules.util.colors.`**`tan`**
(0.8235, 0.7059, 0.549)

`vtkmodules.util.colors.`**`van_dyke_brown`**
(0.37, 0.15, 0.02)

`vtkmodules.util.colors.`**`cadmium_orange`**
(1.0, 0.38, 0.01)

`vtkmodules.util.colors.`**`cadmium_red_light`**
(1.0, 0.01, 0.05)

`vtkmodules.util.colors.`**`carrot`**
(0.93, 0.57, 0.13)

`vtkmodules.util.colors.`**`dark_orange`**
(1.0, 0.549, 0.0)

`vtkmodules.util.colors.`**`mars_orange`**
(0.59, 0.27, 0.08)

`vtkmodules.util.colors.`**`mars_yellow`**
(0.89, 0.44, 0.1)

`vtkmodules.util.colors.`**`orange`**
(1.0, 0.5, 0.0)

`vtkmodules.util.colors.`**`orange_red`**
(1.0, 0.2706, 0.0)

`vtkmodules.util.colors.`**`yellow_ochre`**
(0.89, 0.51, 0.09)

`vtkmodules.util.colors.`**`aureoline_yellow`**
(1.0, 0.66, 0.14)

`vtkmodules.util.colors.`**`banana`**
(0.89, 0.81, 0.34)

vtkmodules.util.colors.**cadmium_lemon**
(1.0, 0.89, 0.01)

vtkmodules.util.colors.**cadmium_yellow**
(1.0, 0.6, 0.07)

vtkmodules.util.colors.**cadmium_yellow_light**
(1.0, 0.69, 0.06)

vtkmodules.util.colors.**gold**
(1.0, 0.8431, 0.0)

vtkmodules.util.colors.**goldenrod**
(0.8549, 0.6471, 0.1255)

vtkmodules.util.colors.**goldenrod_dark**
(0.7216, 0.5255, 0.0431)

vtkmodules.util.colors.**goldenrod_light**
(0.9804, 0.9804, 0.8235)

vtkmodules.util.colors.**goldenrod_pale**
(0.9333, 0.9098, 0.6667)

vtkmodules.util.colors.**light_goldenrod**
(0.9333, 0.8667, 0.5098)

vtkmodules.util.colors.**melon**
(0.89, 0.66, 0.41)

vtkmodules.util.colors.**naples_yellow_deep**
(1.0, 0.66, 0.07)

vtkmodules.util.colors.**yellow**
(1.0, 1.0, 0.0)

vtkmodules.util.colors.**yellow_light**
(1.0, 1.0, 0.8784)

vtkmodules.util.colors.**chartreuse**
(0.498, 1.0, 0.0)

vtkmodules.util.colors.**chrome_oxide_green**
(0.4, 0.5, 0.08)

vtkmodules.util.colors.**cinnabar_green**
(0.38, 0.7, 0.16)

vtkmodules.util.colors.**cobalt_green**
(0.24, 0.57, 0.25)

vtkmodules.util.colors.**emerald_green**
(0.0, 0.79, 0.34)

vtkmodules.util.colors.**forest_green**
(0.1333, 0.5451, 0.1333)

`vtkmodules.util.colors.`**`green`**
: (0.0, 1.0, 0.0)

`vtkmodules.util.colors.`**`green_dark`**
: (0.0, 0.3922, 0.0)

`vtkmodules.util.colors.`**`green_pale`**
: (0.5961, 0.9843, 0.5961)

`vtkmodules.util.colors.`**`green_yellow`**
: (0.6784, 1.0, 0.1843)

`vtkmodules.util.colors.`**`lawn_green`**
: (0.4863, 0.9882, 0.0)

`vtkmodules.util.colors.`**`lime_green`**
: (0.1961, 0.8039, 0.1961)

`vtkmodules.util.colors.`**`mint`**
: (0.74, 0.99, 0.79)

`vtkmodules.util.colors.`**`olive`**
: (0.23, 0.37, 0.17)

`vtkmodules.util.colors.`**`olive_drab`**
: (0.4196, 0.5569, 0.1373)

`vtkmodules.util.colors.`**`olive_green_dark`**
: (0.3333, 0.4196, 0.1843)

`vtkmodules.util.colors.`**`permanent_green`**
: (0.04, 0.79, 0.17)

`vtkmodules.util.colors.`**`sap_green`**
: (0.19, 0.5, 0.08)

`vtkmodules.util.colors.`**`sea_green`**
: (0.1804, 0.5451, 0.3412)

`vtkmodules.util.colors.`**`sea_green_dark`**
: (0.5608, 0.7373, 0.5608)

`vtkmodules.util.colors.`**`sea_green_medium`**
: (0.2353, 0.702, 0.4431)

`vtkmodules.util.colors.`**`sea_green_light`**
: (0.1255, 0.698, 0.6667)

`vtkmodules.util.colors.`**`spring_green`**
: (0.0, 1.0, 0.498)

`vtkmodules.util.colors.`**`spring_green_medium`**
: (0.0, 0.9804, 0.6039)

`vtkmodules.util.colors.`**`terre_verte`**
: (0.22, 0.37, 0.06)

vtkmodules.util.colors.**viridian_light**
: (0.43, 1.0, 0.44)

vtkmodules.util.colors.**yellow_green**
: (0.6039, 0.8039, 0.1961)

vtkmodules.util.colors.**aquamarine**
: (0.498, 1.0, 0.8314)

vtkmodules.util.colors.**aquamarine_medium**
: (0.4, 0.8039, 0.6667)

vtkmodules.util.colors.**cyan**
: (0.0, 1.0, 1.0)

vtkmodules.util.colors.**cyan_white**
: (0.8784, 1.0, 1.0)

vtkmodules.util.colors.**turquoise**
: (0.251, 0.8784, 0.8157)

vtkmodules.util.colors.**turquoise_dark**
: (0.0, 0.8078, 0.8196)

vtkmodules.util.colors.**turquoise_medium**
: (0.2824, 0.8196, 0.8)

vtkmodules.util.colors.**turquoise_pale**
: (0.6863, 0.9333, 0.9333)

vtkmodules.util.colors.**alice_blue**
: (0.9412, 0.9725, 1.0)

vtkmodules.util.colors.**blue**
: (0.0, 0.0, 1.0)

vtkmodules.util.colors.**blue_light**
: (0.6784, 0.8471, 0.902)

vtkmodules.util.colors.**blue_medium**
: (0.0, 0.0, 0.8039)

vtkmodules.util.colors.**cadet**
: (0.3725, 0.6196, 0.6275)

vtkmodules.util.colors.**cobalt**
: (0.24, 0.35, 0.67)

vtkmodules.util.colors.**cornflower**
: (0.3922, 0.5843, 0.9294)

vtkmodules.util.colors.**cerulean**
: (0.02, 0.72, 0.8)

vtkmodules.util.colors.**dodger_blue**
: (0.1176, 0.5647, 1.0)

`vtkmodules.util.colors.`**`indigo`**
(0.03, 0.18, 0.33)

`vtkmodules.util.colors.`**`manganese_blue`**
(0.01, 0.66, 0.62)

`vtkmodules.util.colors.`**`midnight_blue`**
(0.098, 0.098, 0.4392)

`vtkmodules.util.colors.`**`navy`**
(0.0, 0.0, 0.502)

`vtkmodules.util.colors.`**`peacock`**
(0.2, 0.63, 0.79)

`vtkmodules.util.colors.`**`powder_blue`**
(0.6902, 0.8784, 0.902)

`vtkmodules.util.colors.`**`royal_blue`**
(0.2549, 0.4118, 0.8824)

`vtkmodules.util.colors.`**`slate_blue`**
(0.4157, 0.3529, 0.8039)

`vtkmodules.util.colors.`**`slate_blue_dark`**
(0.2824, 0.2392, 0.5451)

`vtkmodules.util.colors.`**`slate_blue_light`**
(0.5176, 0.4392, 1.0)

`vtkmodules.util.colors.`**`slate_blue_medium`**
(0.4824, 0.4078, 0.9333)

`vtkmodules.util.colors.`**`sky_blue`**
(0.5294, 0.8078, 0.9216)

`vtkmodules.util.colors.`**`sky_blue_deep`**
(0.0, 0.749, 1.0)

`vtkmodules.util.colors.`**`sky_blue_light`**
(0.5294, 0.8078, 0.9804)

`vtkmodules.util.colors.`**`steel_blue`**
(0.2745, 0.5098, 0.7059)

`vtkmodules.util.colors.`**`steel_blue_light`**
(0.6902, 0.7686, 0.8706)

`vtkmodules.util.colors.`**`turquoise_blue`**
(0.0, 0.78, 0.55)

`vtkmodules.util.colors.`**`ultramarine`**
(0.07, 0.04, 0.56)

`vtkmodules.util.colors.`**`blue_violet`**
(0.5412, 0.1686, 0.8863)

`vtkmodules.util.colors.`**`cobalt_violet_deep`**
: (0.57, 0.13, 0.62)

`vtkmodules.util.colors.`**`magenta`**
: (1.0, 0.0, 1.0)

`vtkmodules.util.colors.`**`orchid`**
: (0.8549, 0.4392, 0.8392)

`vtkmodules.util.colors.`**`orchid_dark`**
: (0.6, 0.1961, 0.8)

`vtkmodules.util.colors.`**`orchid_medium`**
: (0.7294, 0.3333, 0.8275)

`vtkmodules.util.colors.`**`permanent_red_violet`**
: (0.86, 0.15, 0.27)

`vtkmodules.util.colors.`**`plum`**
: (0.8667, 0.6275, 0.8667)

`vtkmodules.util.colors.`**`purple`**
: (0.6275, 0.1255, 0.9412)

`vtkmodules.util.colors.`**`purple_medium`**
: (0.5765, 0.4392, 0.8588)

`vtkmodules.util.colors.`**`ultramarine_violet`**
: (0.36, 0.14, 0.43)

`vtkmodules.util.colors.`**`violet`**
: (0.56, 0.37, 0.6)

`vtkmodules.util.colors.`**`violet_dark`**
: (0.5804, 0.0, 0.8275)

`vtkmodules.util.colors.`**`violet_red`**
: (0.8157, 0.1255, 0.5647)

`vtkmodules.util.colors.`**`violet_red_medium`**
: (0.7804, 0.0824, 0.5216)

`vtkmodules.util.colors.`**`violet_red_pale`**
: (0.8588, 0.4392, 0.5765)

`vtkmodules.util.pickle_support`

This module generates support for pickling vtkDataObjects from python. It needs to be imported specifically in order to work:

> import vtkmodules.util.pickle_support

Once imported however, the pickling of data objects is very straightforward. Here is an example using poly data:

> sphereSrc = vtkSphereSource() sphereSrc.Update() pickled = pickle.dumps(sphereSrc.GetOutput()) unpickled = pickle.loads(pickled) print(unpickled) *description of sphere data set*

The underlying serialization of the vtkDatObjects is based on the marshaling capabilities found in vtkCommunicator. Importing this module adds entries for the most common data objects in the global dispatch table used by pickle. NumPy is required as well since the -serialized data object gets pickled as a numpy array.

Module Contents

Functions

unserialize_VTK_data_object	Takes a state dictionary with entries:
serialize_VTK_data_object	Returns a tuple with a reference to the unpickling function and a state dictionary with entries:

API

vtkmodules.util.pickle_support.**unserialize_VTK_data_object**(*state*)

Takes a state dictionary with entries:

- Type : a string with the class name for the data object
- Serialized : a numpy array with the serialized data object

and transforms it into a data object.

vtkmodules.util.pickle_support.**serialize_VTK_data_object**(*data_object*)

Returns a tuple with a reference to the unpickling function and a state dictionary with entries:

- Type : a string with the class name for the data object
- Serialized : a numpy array with the serialized data object

This is exactly the state dictionary that unserialize_VTK_data_object expects.

vtkmodules.util.vtkVariant

Utility functions to mimic the template support functions for vtkVariant

Module Contents

Classes

vtkVariantStrictWeakOrderKey	A key method (class, actually) for use with sort()

Functions

vtkVariantCreate	Create a vtkVariant of the specified type, where the type is in the following format: 'int', 'unsigned
vtkVariantExtract	Extract the specified value type from the vtkVariant, where the type is in the following format: 'int
vtkVariantCast	Cast the vtkVariant to the specified value type, where the type is in the following format: 'int', 'uns
vtkVariantStrictWeakOrder	Compare variants by type first, and then by value.
vtkVariantStrictEquality	Check two variants for strict equality of type and value.
vtkVariantLessThan	Return true if s1 < s2.
vtkVariantEqual	Return true if s1 == s2.

Data

_variant_type_map
_variant_method_map
_variant_check_map

API

`vtkmodules.util.vtkVariant.`**`_variant_type_map`**

None

`vtkmodules.util.vtkVariant.`**`_variant_method_map`**

None

`vtkmodules.util.vtkVariant.`**`_variant_check_map`**

None

`vtkmodules.util.vtkVariant.`**`vtkVariantCreate`**(*v*, *t*)

Create a vtkVariant of the specified type, where the type is in the following format: 'int', 'unsigned int', etc. for numeric types, and 'string' for strings. You can also use an integer VTK type constant for the type.

`vtkmodules.util.vtkVariant.`**`vtkVariantExtract`**(*v*, *t=None*)

Extract the specified value type from the vtkVariant, where the type is in the following format: 'int', 'unsigned int', etc. for numeric types, and 'string' for strings. You can also use an integer VTK type constant for the type. Set the type to 'None" to extract the value in its native type.

`vtkmodules.util.vtkVariant.`**`vtkVariantCast`**(*v*, *t*)

Cast the vtkVariant to the specified value type, where the type is in the following format: 'int', 'unsigned int', etc. for numeric types, and 'string' for strings. You can also use an integer VTK type constant for the type.

`vtkmodules.util.vtkVariant.`**`vtkVariantStrictWeakOrder`**(*s1*, *s2*)

Compare variants by type first, and then by value.

`class` `vtkmodules.util.vtkVariant.`**`vtkVariantStrictWeakOrderKey`**(*obj*, **args*)

A key method (class, actually) for use with sort()

Initialization

`__lt__`(*other*)

`vtkmodules.util.vtkVariant.`**`vtkVariantStrictEquality`**(*s1*, *s2*)

Check two variants for strict equality of type and value.

`vtkmodules.util.vtkVariant.`**`vtkVariantLessThan`**(*s1*, *s2*)

Return true if s1 < s2.

`vtkmodules.util.vtkVariant.`**`vtkVariantEqual`**(*s1*, *s2*)

Return true if s1 == s2.

`vtkmodules.util.vtkMethodParser`

This python module provides functionality to parse the methods of a VTK object.

Created by Prabhu Ramachandran. Committed in Apr, 2002.

Module Contents

Classes

`VtkDirMethodParser`	Parses the methods from dir(vtk_obj).
`VtkPrintMethodParser`	This class finds the methods for a given vtkObject. It uses the output from vtkObject->Print() (or in Pyth

Functions

`debug`

Data

`DEBUG`

API

`vtkmodules.util.vtkMethodParser.`**`DEBUG`**

0

`vtkmodules.util.vtkMethodParser.`**`debug`**(*msg*)

`class` `vtkmodules.util.vtkMethodParser.`**`VtkDirMethodParser`**

Parses the methods from dir(vtk_obj).

`initialize_methods`(*vtk_obj*)

parse_methods(*vtk_obj*)

clean_up_methods(*vtk_obj*)

clean_get_set(*vtk_obj*)

clean_state_methods(*vtk_obj*)

clean_get_methods(*vtk_obj*)

toggle_methods()

state_methods()

get_set_methods()

get_methods()

class vtkmodules.util.vtkMethodParser.**VtkPrintMethodParser**

This class finds the methods for a given vtkObject. It uses the output from vtkObject->Print() (or in Python str(vtkObject)) and output from the VtkDirMethodParser to obtain the methods.

parse_methods(*vtk_obj*)

Parse for the methods.

_get_str_obj(*vtk_obj*)

_initialize_methods(*vtk_obj*)

Do the basic parsing and setting up

_clean_up_methods(*vtk_obj*)

Merge dir and str methods. Finish up.

toggle_methods()

state_methods()

get_set_methods()

get_methods()

vtkmodules.util.vtkImageImportFromArray

vtkImageImportFromArray: a NumPy front-end to vtkImageImport

Load a python array into a vtk image. To use this class, you must have NumPy installed (http://numpy.scipy.org/)

Methods:

SetArray() – set the numpy array to load Update() – generate the output GetOutput() – get the image as vtkImageData GetOutputPort() – connect to VTK pipeline

Methods from vtkImageImport: (if you don't set these, sensible defaults will be used)

SetDataExtent() SetDataSpacing() SetDataOrigin()

Module Contents

Classes

vtkImageImportFromArray

API

class vtkmodules.util.vtkImageImportFromArray.**vtkImageImportFromArray**

Initialization

__typeDict

None

__sizeDict

None

SetConvertIntToUnsignedShort(*yesno*)

GetConvertIntToUnsignedShort()

ConvertIntToUnsignedShortOn()

ConvertIntToUnsignedShortOff()

Update()

GetOutputPort()

GetOutput()

SetArray(*imArray*)

GetArray()

SetDataExtent(*extent*)

GetDataExtent()

SetDataSpacing(*spacing*)

GetDataSpacing()

SetDataOrigin(*origin*)

GetDataOrigin()

vtkmodules.util.keys

Utility module to make it easier to create new keys.

Module Contents

Functions

MakeKey	Given a key type, make a new key of given name and location.

API

`vtkmodules.util.keys.`**`MakeKey`**`(`*key_type*, *name*, *location*, **args*`)`

Given a key type, make a new key of given name and location.

vtkmodules.util.vtkAlgorithm

Module Contents

Classes

VTKAlgorithm	This is a superclass which can be derived to implement Python classes that work with vtkPythonAlgor
VTKPythonAlgorithmBase	This is a superclass which can be derived to implement Python classes that act as VTK algorithms in a

API

`class` `vtkmodules.util.vtkAlgorithm.`**`VTKAlgorithm`**`(`*nInputPorts=1*, *inputType='vtkDataSet'*, *nOutputPorts=1*, *outputType='vtkPolyData'*`)`

Bases: `object`

This is a superclass which can be derived to implement Python classes that work with vtkPythonAlgorithm. It implements Initialize(), ProcessRequest(), FillInputPortInformation() and FillOutputPortInformation().

Initialize() sets the input and output ports based on data members.

ProcessRequest() calls RequestXXX() methods to implement various pipeline passes.

FillInputPortInformation() and FillOutputPortInformation() set the input and output types based on data members.

Initialization

Sets up default NumberOfInputPorts, NumberOfOutputPorts, InputType and OutputType that are used by various initialization methods.

`Initialize`(*vtkself*)

Sets up number of input and output ports based on NumberOfInputPorts and NumberOfOutputPorts.

`GetInputData`(*inInfo*, *i*, *j*)

Convenience method that returns an input data object given a vector of information objects and two indices.

`GetOutputData`(*outInfo*, *i*)

Convenience method that returns an output data object given an information object and an index.

`RequestDataObject`(*vtkself*, *request*, *inInfo*, *outInfo*)

Overwritten by subclass to manage data object creation. There is not need to overwrite this class if the output can be created based on the OutputType data member.

`RequestInformation`(*vtkself*, *request*, *inInfo*, *outInfo*)

Overwritten by subclass to provide meta-data to downstream pipeline.

`RequestUpdateExtent`(*vtkself*, *request*, *inInfo*, *outInfo*)

Overwritten by subclass to modify data request going to upstream pipeline.

`abstract RequestData`(*vtkself*, *request*, *inInfo*, *outInfo*)

Overwritten by subclass to execute the algorithm.

`ProcessRequest`(*vtkself*, *request*, *inInfo*, *outInfo*)

Splits a request to RequestXXX() methods.

`FillInputPortInformation`(*vtkself*, *port*, *info*)

Sets the required input type to InputType.

`FillOutputPortInformation`(*vtkself*, *port*, *info*)

Sets the default output type to OutputType.

`class vtkmodules.util.vtkAlgorithm.VTKPythonAlgorithmBase`(*nInputPorts=1*, *inputType='vtkDataSet'*, *nOutputPorts=1*, *outputType='vtkPolyData'*)

Bases: `vtkmodules.vtkFiltersPython.vtkPythonAlgorithm`

This is a superclass which can be derived to implement Python classes that act as VTK algorithms in a VTK pipeline. It implements ProcessRequest(), FillInputPortInformation() and FillOutputPortInformation().

ProcessRequest() calls RequestXXX() methods to implement various pipeline passes.

FillInputPortInformation() and FillOutputPortInformation() set the input and output types based on data members.

Common use is something like this:

class HDF5Source(VTKPythonAlgorithmBase): def **init**(self): VTKPythonAlgorithmBase.**init**(self, nInputPorts=0, nOutputPorts=1, outputType='vtkImageData')

```
def RequestInformation(self, request, inInfo, outInfo):
    f = h5py.File("foo.h5", 'r')
    dims = f['RTData'].shape[::-1]
    info = outInfo.GetInformationObject(0)
    info.Set(vtkmodules.vtkCommonExecutionModel.vtkStreamingDemandDrivenPipeline.
```

(continues on next page)

(continued from previous page)

```
↪WHOLE_EXTENT(),
        (0, dims[0]-1, 0, dims[1]-1, 0, dims[2]-1), 6)
    return 1

def RequestData(self, request, inInfo, outInfo):
    f = h5py.File("foo.h5", 'r')
    data = f['RTData'][:]
    output = dsa.WrapDataObject(vtkmodules.vtkCommonDataModel.vtkImageData.
↪GetData(outInfo))
    output.SetDimensions(data.shape)
    output.PointData.append(data.flatten(), 'RTData')
    output.PointData.SetActiveScalars('RTData')
    return 1
```

alg = HDF5Source()

cf = vtkmodules.vtkFiltersCore.vtkContourFilter() cf.SetInputConnection(alg.GetOutputPort()) cf.Update()

Initialization

Sets up default NumberOfInputPorts, NumberOfOutputPorts, InputType and OutputType that are used by various methods. Make sure to call this method from any subclass' **init**

`class InternalAlgorithm`

Bases: `object`

Internal class. Do not use.

`Initialize`(*vtkself*)

`FillInputPortInformation`(*vtkself*, *port*, *info*)

`FillOutputPortInformation`(*vtkself*, *port*, *info*)

`ProcessRequest`(*vtkself*, *request*, *inInfo*, *outInfo*)

`GetInputData`(*inInfo*, *i*, *j*)

Convenience method that returns an input data object given a vector of information objects and two indices.

`GetOutputData`(*outInfo*, *i*)

Convenience method that returns an output data object given an information object and an index.

`FillInputPortInformation`(*port*, *info*)

Sets the required input type to InputType.

`FillOutputPortInformation`(*port*, *info*)

Sets the default output type to OutputType.

`ProcessRequest`(*request*, *inInfo*, *outInfo*)

Splits a request to RequestXXX() methods.

`RequestDataObject`(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to manage data object creation. There is not need to overwrite this class if the output can be created based on the OutputType data member.

RequestInformation(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to provide meta-data to downstream pipeline.

RequestUpdateExtent(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to modify data request going to upstream pipeline.

abstract RequestData(*request*, *inInfo*, *outInfo*)

Overwritten by subclass to execute the algorithm.

vtkmodules.util.execution_model

Utility classes to help with the simpler Python interface for connecting and executing pipelines.

Module Contents

Classes

select_ports	Helper class for selecting input and output ports when connecting pipeline objects with the >> operator. Example u
Pipeline	Pipeline objects are created when 2 or more algorithms are connected with the >> operator. They store the first and
Output	Helper object to represent the output of an algorithms as returned by the update() method. Implements the output p

Functions

_call	Set the input of the first filter, update the pipeline and return the output.

Data

__all__

API

vtkmodules.util.execution_model.**__all__**

['select_ports', 'Pipeline', 'Output']

vtkmodules.util.execution_model.**_call**(*first*, *last*, *inp=None*, *port=0*)

Set the input of the first filter, update the pipeline and return the output.

class vtkmodules.util.execution_model.**select_ports**(**args*)

Bases: `object`

Helper class for selecting input and output ports when connecting pipeline objects with the >> operator. Example uses:

Connect a source to the second input of a filter.

source >> select_ports(1, filter)

Connect the second output of a source to a filter.

select_ports(source, 1) >> filter

Combination of both: Connect source to second

input of the filter, then connect the second

output of that filter to another one.

source >>> select_ports(1, filter, 1) >> filter2

Initialization

This constructor takes 2 or 3 arguments. The possibilities are: select_ports(input_port, algorithm) select_ports(algorithm, output_port) select_ports(input_port, algorithm, output_port)

`SetInputConnection`(*inp*)

Forwards to underlying algorithm and port.

`AddInputConnection`(*inp*)

Forwards to underlying algorithm and port.

`GetOutputPort`()

Returns the output port of the underlying algorithm.

`GetInputPortInformation`(*port*)

`update`()

Execute the algorithm and return the output from the selected output port.

`__rshift__`(*rhs*)

Creates a pipeline between the underlying port and an algorithm.

`__rrshift__`(*lhs*)

Creates a pipeline between the underlying port and an algorithm. This is to handle sequence >> select_ports where the port can accept multiple connections.

`__call__`(*inp=None*)

Executes the underlying algorithm by passing input data to the selected input port. Returns a single output or a tuple if there are multiple outputs.

`class vtkmodules.util.execution_model.Pipeline`(*lhs*, *rhs*)

Bases: `object`

Pipeline objects are created when 2 or more algorithms are connected with the >> operator. They store the first and last algorithms in the pipeline and enable connecting more algorithms and executing the pipeline. One should not have to create Pipeline objects directly. They are created by the use of the >> operator.

Initialization

Create a pipeline object that connects two objects of the following type: data object, pipeline object, algorithm object.

`PIPELINE`

0

`ALGORITHM`

1

`DATA`

2

`UNKNOWN`

3

`_connect`(*lhs*, *rhs*, *rhs_alg*, *connect_method*)

`_determine_type`(*arg*)

`update`(***kwargs*)

Update the pipeline and return the last algorithm's output.

`__call__`(*inp=None*)

Sets the input of the first filter, update the pipeline and returns the output. A single data object or a tuple of data objects (when there are multiple outputs) are returned.

`__rshift__`(*rhs*)

Used to connect two pipeline items. The left side can be a data object, an algorithm or a pipeline. The right side can be an algorithm or a pipeline.

`__rrshift__`(*lhs*)

Creates a pipeline between a sequence input and a pipeline.

`class` `vtkmodules.util.execution_model.`**`Output`**(*algorithm*, ***kwargs*)

Bases: `object`

Helper object to represent the output of an algorithms as returned by the update() method. Implements the output property enabling calling update().output.

Initialization

`property output`

Returns a single data object or a tuple of data objects if there are multiple outputs.

`vtkmodules.util.data_model`

This module provides classes that allow numpy style access to VTK datasets. See examples at bottom.

Module Contents

Classes

FieldDataBase	
vtkFieldData	
DataSetAttributesBase	
DataSetAttributes	
PointData	
CellData	
CompositeDataSetAttributesIterator	
CompositeDataSetAttributes	This is a python friendly wrapper for vtkDataSetAttributes for composite datasets. Sin
DataSet	
PointSet	
vtkUnstructuredGrid	
vtkImageData	
vtkPolyData	
CompositeDataIterator	Wrapper for a vtkCompositeDataIterator class to satisfy the python iterator protocol. T
CompositeDataSetBase	A wrapper for vtkCompositeData and subclasses that makes it easier to access Point/C
vtkPartitionedDataSet	

API

class vtkmodules.util.data_model.**FieldDataBase**

Bases: `object`

Initialization

`__getitem__`(*idx*)

Implements the [] operator. Accepts an array name or index.

`__setitem__`(*name*, *value*)

Implements the [] operator. Accepts an array name or index.

`get_array`(*idx*)

Given an index or name, returns a VTKArray.

`keys`()

Returns the names of the arrays as a list.

`values`()

Returns the arrays as a list.

`set_array`(*name*, *narray*)

Appends a new array to the dataset attributes.

class vtkmodules.util.data_model.**vtkFieldData**

Bases: *vtkmodules.util.data_model.FieldDataBase*, *vtkmodules.util.data_model.vtkFieldData*

Initialization

class vtkmodules.util.data_model.**DataSetAttributesBase**

Bases: *vtkmodules.util.data_model.FieldDataBase*

Initialization

class vtkmodules.util.data_model.**DataSetAttributes**

Bases: *vtkmodules.util.data_model.DataSetAttributesBase*, `vtkmodules.vtkCommonDataModel.vtkDataSetAttributes`

Initialization

class vtkmodules.util.data_model.**PointData**

Bases: *vtkmodules.util.data_model.DataSetAttributesBase*, `vtkmodules.vtkCommonDataModel.vtkPointData`

Initialization

class vtkmodules.util.data_model.**CellData**

Bases: *vtkmodules.util.data_model.DataSetAttributesBase*, `vtkmodules.vtkCommonDataModel.vtkCellData`

Initialization

class vtkmodules.util.data_model.**CompositeDataSetAttributesIterator**(*cdsa*)

Bases: `object`

Initialization

__iter__()

__next__()

next()

class vtkmodules.util.data_model.**CompositeDataSetAttributes**(*dataset*, *association*)

Bases: `object`

This is a python friendly wrapper for vtkDataSetAttributes for composite datasets. Since composite datasets themselves don't have attribute data, but the attribute data is associated with the leaf nodes in the composite dataset, this class simulates a DataSetAttributes interface by taking a union of DataSetAttributes associated with all leaf nodes.

Initialization

__determine_arraynames()

modified()

Rescans the contained dataset to update the internal list of arrays.

keys()

Returns the names of the arrays as a list.

__getitem__(*idx*)

Implements the [] operator. Accepts an array name.

__setitem__(*name*, *narray*)

Implements the [] operator. Accepts an array name.

set_array(*name*, *narray*)

Appends a new array to the composite dataset attributes.

get_array(*idx*)

Given a name, returns a VTKCompositeArray.

__iter__()

Creates an iterator for the contained arrays.

class `vtkmodules.util.data_model.`**DataSet**

Bases: `object`

property point_data

property cell_data

convert_to_unstructured_grid()

class `vtkmodules.util.data_model.`**PointSet**

Bases: *vtkmodules.util.data_model.DataSet*

property points

class `vtkmodules.util.data_model.`**vtkUnstructuredGrid**

Bases: *vtkmodules.util.data_model.PointSet*, *vtkmodules.util.data_model.vtkUnstructuredGrid*

property cells

class `vtkmodules.util.data_model.`**vtkImageData**

Bases: *vtkmodules.util.data_model.DataSet*, *vtkmodules.util.data_model.vtkImageData*

class `vtkmodules.util.data_model.`**vtkPolyData**

Bases: *vtkmodules.util.data_model.PointSet*, *vtkmodules.util.data_model.vtkPolyData*

property polygons

class `vtkmodules.util.data_model.`**CompositeDataIterator**(*cds*)

Bases: `object`

Wrapper for a vtkCompositeDataIterator class to satisfy the python iterator protocol. This iterator iterates over non-empty leaf nodes. To iterate over empty or non-leaf nodes, use the vtkCompositeDataIterator directly.

Initialization

`__iter__`()

`__next__`()

`next`()

`__getattr__`(*name*)

Returns attributes from the vtkCompositeDataIterator.

`class vtkmodules.util.data_model.`**`CompositeDataSetBase`**(***kwargs*)

Bases: `object`

A wrapper for vtkCompositeData and subclasses that makes it easier to access Point/Cell/Field data as VTK-CompositeDataArrays. It also provides a Python type iterator.

Initialization

`__iter__`()

Creates an iterator for the contained datasets.

`get_attributes`(*type*)

Returns the attributes specified by the type as a CompositeDataSetAttributes instance.

`property point_data`

Returns the point data as a DataSetAttributes instance.

`property cell_data`

Returns the cell data as a DataSetAttributes instance.

`property field_data`

Returns the field data as a DataSetAttributes instance.

`property points`

Returns the points as a VTKCompositeDataArray instance.

`class vtkmodules.util.data_model.`**`vtkPartitionedDataSet`**(***kwargs*)

Bases: *`vtkmodules.util.data_model.CompositeDataSetBase`*, *`vtkmodules.util.data_model.vtkPartitionedDataSet`*

`append`(*dataset*)

`vtkmodules.util.numpy_support`

Caveats:

- Bit arrays in general do not have a numpy equivalent and are not supported. Char arrays are also not easy to handle and might not work as you expect. Patches welcome.
- You need to make sure you hold a reference to a Numpy array you want to import into VTK. If not you'll get a segfault (in the best case). The same holds in reverse when you convert a VTK array to a numpy array – don't delete the VTK array.

Created by Prabhu Ramachandran in Feb. 2008.

This module adds support to easily import and export NumPy (http://numpy.scipy.org) arrays into/out of VTK arrays. The code is loosely based on TVTK (https://svn.enthought.com/enthought/wiki/TVTK).

This code depends on an addition to the VTK data arrays made by Berk Geveci to make it support Python's buffer protocol (on Feb. 15, 2008).

The main functionality of this module is provided by the two functions: numpy_to_vtk, vtk_to_numpy.

Module Contents

Functions

get_vtk_array_type	Returns a VTK typecode given a numpy array.
get_vtk_to_numpy_typemap	Returns the VTK array type to numpy array type mapping.
get_numpy_array_type	Returns a numpy array typecode given a VTK array type.
create_vtk_array	Internal function used to create a VTK data array from another VTK array given the VTK array typ
numpy_to_vtk	Converts a real numpy Array to a VTK array object.
numpy_to_vtkIdTypeArray	
vtk_to_numpy	Converts a VTK data array to a numpy array.

Data

VTK_ID_TYPE_SIZE
VTK_LONG_TYPE_SIZE

API

`vtkmodules.util.numpy_support.`**`VTK_ID_TYPE_SIZE`**

'GetDataTypeSize(…)'

`vtkmodules.util.numpy_support.`**`VTK_LONG_TYPE_SIZE`**

'GetDataTypeSize(…)'

`vtkmodules.util.numpy_support.`**`get_vtk_array_type`**(*numpy_array_type*)

Returns a VTK typecode given a numpy array.

`vtkmodules.util.numpy_support.`**`get_vtk_to_numpy_typemap`**()

Returns the VTK array type to numpy array type mapping.

`vtkmodules.util.numpy_support.`**`get_numpy_array_type`**(*vtk_array_type*)

Returns a numpy array typecode given a VTK array type.

`vtkmodules.util.numpy_support.`**`create_vtk_array`**(*vtk_arr_type*)

Internal function used to create a VTK data array from another VTK array given the VTK array type.

`vtkmodules.util.numpy_support.`**`numpy_to_vtk`**`(`*num_array*, *deep=0*, *array_type=None*`)`

Converts a real numpy Array to a VTK array object.

This function only works for real arrays. Complex arrays are NOT handled. It also works for multi-component arrays. However, only 1, and 2 dimensional arrays are supported. This function is very efficient, so large arrays should not be a problem.

If the second argument is set to 1, the array is deep-copied from from numpy. This is not as efficient as the default behavior (shallow copy) and uses more memory but detaches the two arrays such that the numpy array can be released.

WARNING: You must maintain a reference to the passed numpy array, if the numpy data is gc'd and VTK will point to garbage which will in the best case give you a segfault.

Parameters:

num_array a 1D or 2D, real numpy array.

`vtkmodules.util.numpy_support.`**`numpy_to_vtkIdTypeArray`**`(`*num_array*, *deep=0*`)`

`vtkmodules.util.numpy_support.`**`vtk_to_numpy`**`(`*vtk_array*`)`

Converts a VTK data array to a numpy array.

Given a subclass of vtkDataArray, this function returns an appropriate numpy array containing the same data – it actually points to the same data.

WARNING: This does not work for bit arrays.

Parameters

vtk_array The VTK data array to be converted.

`vtkmodules.util.misc`

Miscellaneous functions and classes that don't fit into specific categories.

Module Contents

Functions

calldata_type	set_call_data_type(type) – convenience decorator to easily set the CallDataType attribute for python fu
vtkGetDataRoot	vtkGetDataRoot() – return vtk example data directory
vtkGetTempDir	vtkGetTempDir() – return vtk testing temp dir
vtkRegressionTestImage	vtkRegressionTestImage(renWin) – produce regression image for window

API

`vtkmodules.util.misc.`**`calldata_type`**`(`*type*`)`

set_call_data_type(type) – convenience decorator to easily set the CallDataType attribute for python function used as observer callback. For example:

import vtkmodules.util.calldata_type import vtkmodules.util.vtkConstants import vtkmodules.vtkCommonCore import vtkCommand, vtkLookupTable

@calldata_type(vtkConstants.VTK_STRING) def onError(caller, event, calldata): print("caller: %s - event: %s - msg: %s" % (caller.GetClassName(), event, calldata))

lt = vtkLookupTable() lt.AddObserver(vtkCommand.ErrorEvent, onError) lt.SetTableRange(2,1)

`vtkmodules.util.misc.`**`vtkGetDataRoot`**`()`

vtkGetDataRoot() – return vtk example data directory

`vtkmodules.util.misc.`**`vtkGetTempDir`**`()`

vtkGetTempDir() – return vtk testing temp dir

`vtkmodules.util.misc.`**`vtkRegressionTestImage`**`(`*renWin*`)`

vtkRegressionTestImage(renWin) – produce regression image for window

This function writes out a regression .png file for a vtkWindow. Does anyone involved in testing care to elaborate?

`vtkmodules.util.vtkConstants`

This file is obsolete. All the constants are part of the base vtk module.

Module Contents

Functions

vtkImageScalarTypeNameMacro

Data

_VTK_FLOAT_MAX
_VTK_INT_MAX
VTK_VOID
VTK_BIT
VTK_CHAR
VTK_SIGNED_CHAR
VTK_UNSIGNED_CHAR
VTK_SHORT
VTK_UNSIGNED_SHORT

continues on next page

Table 21 – continued from previous page

VTK_INT
VTK_UNSIGNED_INT
VTK_LONG
VTK_UNSIGNED_LONG
VTK_FLOAT
VTK_DOUBLE
VTK_ID_TYPE
VTK_STRING
VTK_OPAQUE
VTK_LONG_LONG
VTK_UNSIGNED_LONG_LONG
VTK_VARIANT
VTK_OBJECT
VTK_BIT_MIN
VTK_BIT_MAX
VTK_CHAR_MIN
VTK_CHAR_MAX
VTK_UNSIGNED_CHAR_MIN
VTK_UNSIGNED_CHAR_MAX
VTK_SHORT_MIN
VTK_SHORT_MAX
VTK_UNSIGNED_SHORT_MIN
VTK_UNSIGNED_SHORT_MAX
VTK_INT_MIN
VTK_INT_MAX
VTK_LONG_MIN
VTK_LONG_MAX
VTK_FLOAT_MIN
VTK_FLOAT_MAX
VTK_DOUBLE_MIN
VTK_DOUBLE_MAX
VTK_POLY_DATA
VTK_STRUCTURED_POINTS
VTK_STRUCTURED_GRID
VTK_RECTILINEAR_GRID
VTK_UNSTRUCTURED_GRID
VTK_PIECEWISE_FUNCTION
VTK_IMAGE_DATA
VTK_DATA_OBJECT
VTK_DATA_SET
VTK_POINT_SET
VTK_UNIFORM_GRID
VTK_COMPOSITE_DATA_SET
VTK_MULTIGROUP_DATA_SET
VTK_MULTIBLOCK_DATA_SET
VTK_HIERARCHICAL_DATA_SET
VTK_HIERARCHICAL_BOX_DATA_SET
VTK_GENERIC_DATA_SET
VTK_HYPER_OCTREE
VTK_TEMPORAL_DATA_SET
VTK_TABLE
VTK_GRAPH

continues on next page

Table 21 – continued from previous page

VTK_TREE
VTK_SELECTION
VTK_OK
VTK_ERROR
VTK_ARIAL
VTK_COURIER
VTK_TIMES
VTK_UNKNOWN_FONT
VTK_TEXT_LEFT
VTK_TEXT_CENTERED
VTK_TEXT_RIGHT
VTK_TEXT_BOTTOM
VTK_TEXT_TOP
VTK_TEXT_GLOBAL_ANTIALIASING_SOME
VTK_TEXT_GLOBAL_ANTIALIASING_NONE
VTK_TEXT_GLOBAL_ANTIALIASING_ALL
VTK_LUMINANCE
VTK_LUMINANCE_ALPHA
VTK_RGB
VTK_RGBA
VTK_COLOR_MODE_DEFAULT
VTK_COLOR_MODE_MAP_SCALARS
VTK_NEAREST_INTERPOLATION
VTK_LINEAR_INTERPOLATION
VTK_MAX_VRCOMP
VTK_EMPTY_CELL
VTK_VERTEX
VTK_POLY_VERTEX
VTK_LINE
VTK_POLY_LINE
VTK_TRIANGLE
VTK_TRIANGLE_STRIP
VTK_POLYGON
VTK_PIXEL
VTK_QUAD
VTK_TETRA
VTK_VOXEL
VTK_HEXAHEDRON
VTK_WEDGE
VTK_PYRAMID
VTK_PENTAGONAL_PRISM
VTK_HEXAGONAL_PRISM
VTK_QUADRATIC_EDGE
VTK_QUADRATIC_TRIANGLE
VTK_QUADRATIC_QUAD
VTK_QUADRATIC_TETRA
VTK_QUADRATIC_HEXAHEDRON
VTK_QUADRATIC_WEDGE
VTK_QUADRATIC_PYRAMID
VTK_BIQUADRATIC_QUAD
VTK_TRIQUADRATIC_HEXAHEDRON
VTK_QUADRATIC_LINEAR_QUAD

continues on next page

Table 21 – continued from previous page

VTK_QUADRATIC_LINEAR_WEDGE
VTK_BIQUADRATIC_QUADRATIC_WEDGE
VTK_BIQUADRATIC_QUADRATIC_HEXAHEDRON
VTK_CONVEX_POINT_SET
VTK_PARAMETRIC_CURVE
VTK_PARAMETRIC_SURFACE
VTK_PARAMETRIC_TRI_SURFACE
VTK_PARAMETRIC_QUAD_SURFACE
VTK_PARAMETRIC_TETRA_REGION
VTK_PARAMETRIC_HEX_REGION
VTK_HIGHER_ORDER_EDGE
VTK_HIGHER_ORDER_TRIANGLE
VTK_HIGHER_ORDER_QUAD
VTK_HIGHER_ORDER_POLYGON
VTK_HIGHER_ORDER_TETRAHEDRON
VTK_HIGHER_ORDER_WEDGE
VTK_HIGHER_ORDER_PYRAMID
VTK_HIGHER_ORDER_HEXAHEDRON
__vtkTypeNameDict

API

vtkmodules.util.vtkConstants.**_VTK_FLOAT_MAX**

1e+38

vtkmodules.util.vtkConstants.**_VTK_INT_MAX**

2147483647

vtkmodules.util.vtkConstants.**VTK_VOID**

0

vtkmodules.util.vtkConstants.**VTK_BIT**

1

vtkmodules.util.vtkConstants.**VTK_CHAR**

2

vtkmodules.util.vtkConstants.**VTK_SIGNED_CHAR**

15

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_CHAR**

3

vtkmodules.util.vtkConstants.**VTK_SHORT**

4

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_SHORT**

5

vtkmodules.util.vtkConstants.**VTK_INT**

6

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_INT**

7

vtkmodules.util.vtkConstants.**VTK_LONG**
    8

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_LONG**
    9

vtkmodules.util.vtkConstants.**VTK_FLOAT**
    10

vtkmodules.util.vtkConstants.**VTK_DOUBLE**
    11

vtkmodules.util.vtkConstants.**VTK_ID_TYPE**
    12

vtkmodules.util.vtkConstants.**VTK_STRING**
    13

vtkmodules.util.vtkConstants.**VTK_OPAQUE**
    14

vtkmodules.util.vtkConstants.**VTK_LONG_LONG**
    16

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_LONG_LONG**
    17

vtkmodules.util.vtkConstants.**VTK_VARIANT**
    20

vtkmodules.util.vtkConstants.**VTK_OBJECT**
    21

vtkmodules.util.vtkConstants.**VTK_BIT_MIN**
    0

vtkmodules.util.vtkConstants.**VTK_BIT_MAX**
    1

vtkmodules.util.vtkConstants.**VTK_CHAR_MIN**
    None

vtkmodules.util.vtkConstants.**VTK_CHAR_MAX**
    127

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_CHAR_MIN**
    0

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_CHAR_MAX**
    255

vtkmodules.util.vtkConstants.**VTK_SHORT_MIN**
    None

vtkmodules.util.vtkConstants.**VTK_SHORT_MAX**
    32767

vtkmodules.util.vtkConstants.**VTK_UNSIGNED_SHORT_MIN**
    0

`vtkmodules.util.vtkConstants.`**`VTK_UNSIGNED_SHORT_MAX`**
65535

`vtkmodules.util.vtkConstants.`**`VTK_INT_MIN`**
None

`vtkmodules.util.vtkConstants.`**`VTK_INT_MAX`**
None

`vtkmodules.util.vtkConstants.`**`VTK_LONG_MIN`**
None

`vtkmodules.util.vtkConstants.`**`VTK_LONG_MAX`**
None

`vtkmodules.util.vtkConstants.`**`VTK_FLOAT_MIN`**
None

`vtkmodules.util.vtkConstants.`**`VTK_FLOAT_MAX`**
None

`vtkmodules.util.vtkConstants.`**`VTK_DOUBLE_MIN`**
None

`vtkmodules.util.vtkConstants.`**`VTK_DOUBLE_MAX`**
1e+99

`vtkmodules.util.vtkConstants.`**`VTK_POLY_DATA`**
0

`vtkmodules.util.vtkConstants.`**`VTK_STRUCTURED_POINTS`**
1

`vtkmodules.util.vtkConstants.`**`VTK_STRUCTURED_GRID`**
2

`vtkmodules.util.vtkConstants.`**`VTK_RECTILINEAR_GRID`**
3

`vtkmodules.util.vtkConstants.`**`VTK_UNSTRUCTURED_GRID`**
4

`vtkmodules.util.vtkConstants.`**`VTK_PIECEWISE_FUNCTION`**
5

`vtkmodules.util.vtkConstants.`**`VTK_IMAGE_DATA`**
6

`vtkmodules.util.vtkConstants.`**`VTK_DATA_OBJECT`**
7

`vtkmodules.util.vtkConstants.`**`VTK_DATA_SET`**
8

`vtkmodules.util.vtkConstants.`**`VTK_POINT_SET`**
9

`vtkmodules.util.vtkConstants.`**`VTK_UNIFORM_GRID`**
10

vtkmodules.util.vtkConstants.**VTK_COMPOSITE_DATA_SET**
: 11

vtkmodules.util.vtkConstants.**VTK_MULTIGROUP_DATA_SET**
: 12

vtkmodules.util.vtkConstants.**VTK_MULTIBLOCK_DATA_SET**
: 13

vtkmodules.util.vtkConstants.**VTK_HIERARCHICAL_DATA_SET**
: 14

vtkmodules.util.vtkConstants.**VTK_HIERARCHICAL_BOX_DATA_SET**
: 15

vtkmodules.util.vtkConstants.**VTK_GENERIC_DATA_SET**
: 16

vtkmodules.util.vtkConstants.**VTK_HYPER_OCTREE**
: 17

vtkmodules.util.vtkConstants.**VTK_TEMPORAL_DATA_SET**
: 18

vtkmodules.util.vtkConstants.**VTK_TABLE**
: 19

vtkmodules.util.vtkConstants.**VTK_GRAPH**
: 20

vtkmodules.util.vtkConstants.**VTK_TREE**
: 21

vtkmodules.util.vtkConstants.**VTK_SELECTION**
: 22

vtkmodules.util.vtkConstants.**VTK_OK**
: 1

vtkmodules.util.vtkConstants.**VTK_ERROR**
: 2

vtkmodules.util.vtkConstants.**VTK_ARIAL**
: 0

vtkmodules.util.vtkConstants.**VTK_COURIER**
: 1

vtkmodules.util.vtkConstants.**VTK_TIMES**
: 2

vtkmodules.util.vtkConstants.**VTK_UNKNOWN_FONT**
: 3

vtkmodules.util.vtkConstants.**VTK_TEXT_LEFT**
: 0

vtkmodules.util.vtkConstants.**VTK_TEXT_CENTERED**
: 1

vtkmodules.util.vtkConstants.**VTK_TEXT_RIGHT**
    2

vtkmodules.util.vtkConstants.**VTK_TEXT_BOTTOM**
    0

vtkmodules.util.vtkConstants.**VTK_TEXT_TOP**
    2

vtkmodules.util.vtkConstants.**VTK_TEXT_GLOBAL_ANTIALIASING_SOME**
    0

vtkmodules.util.vtkConstants.**VTK_TEXT_GLOBAL_ANTIALIASING_NONE**
    1

vtkmodules.util.vtkConstants.**VTK_TEXT_GLOBAL_ANTIALIASING_ALL**
    2

vtkmodules.util.vtkConstants.**VTK_LUMINANCE**
    1

vtkmodules.util.vtkConstants.**VTK_LUMINANCE_ALPHA**
    2

vtkmodules.util.vtkConstants.**VTK_RGB**
    3

vtkmodules.util.vtkConstants.**VTK_RGBA**
    4

vtkmodules.util.vtkConstants.**VTK_COLOR_MODE_DEFAULT**
    0

vtkmodules.util.vtkConstants.**VTK_COLOR_MODE_MAP_SCALARS**
    1

vtkmodules.util.vtkConstants.**VTK_NEAREST_INTERPOLATION**
    0

vtkmodules.util.vtkConstants.**VTK_LINEAR_INTERPOLATION**
    1

vtkmodules.util.vtkConstants.**VTK_MAX_VRCOMP**
    4

vtkmodules.util.vtkConstants.**VTK_EMPTY_CELL**
    0

vtkmodules.util.vtkConstants.**VTK_VERTEX**
    1

vtkmodules.util.vtkConstants.**VTK_POLY_VERTEX**
    2

vtkmodules.util.vtkConstants.**VTK_LINE**
    3

vtkmodules.util.vtkConstants.**VTK_POLY_LINE**
    4

vtkmodules.util.vtkConstants.**VTK_TRIANGLE**
    5

vtkmodules.util.vtkConstants.**VTK_TRIANGLE_STRIP**
    6

vtkmodules.util.vtkConstants.**VTK_POLYGON**
    7

vtkmodules.util.vtkConstants.**VTK_PIXEL**
    8

vtkmodules.util.vtkConstants.**VTK_QUAD**
    9

vtkmodules.util.vtkConstants.**VTK_TETRA**
    10

vtkmodules.util.vtkConstants.**VTK_VOXEL**
    11

vtkmodules.util.vtkConstants.**VTK_HEXAHEDRON**
    12

vtkmodules.util.vtkConstants.**VTK_WEDGE**
    13

vtkmodules.util.vtkConstants.**VTK_PYRAMID**
    14

vtkmodules.util.vtkConstants.**VTK_PENTAGONAL_PRISM**
    15

vtkmodules.util.vtkConstants.**VTK_HEXAGONAL_PRISM**
    16

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_EDGE**
    21

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_TRIANGLE**
    22

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_QUAD**
    23

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_TETRA**
    24

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_HEXAHEDRON**
    25

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_WEDGE**
    26

vtkmodules.util.vtkConstants.**VTK_QUADRATIC_PYRAMID**
    27

vtkmodules.util.vtkConstants.**VTK_BIQUADRATIC_QUAD**
    28

`vtkmodules.util.vtkConstants.`**`VTK_TRIQUADRATIC_HEXAHEDRON`**
29

`vtkmodules.util.vtkConstants.`**`VTK_QUADRATIC_LINEAR_QUAD`**
30

`vtkmodules.util.vtkConstants.`**`VTK_QUADRATIC_LINEAR_WEDGE`**
31

`vtkmodules.util.vtkConstants.`**`VTK_BIQUADRATIC_QUADRATIC_WEDGE`**
32

`vtkmodules.util.vtkConstants.`**`VTK_BIQUADRATIC_QUADRATIC_HEXAHEDRON`**
33

`vtkmodules.util.vtkConstants.`**`VTK_CONVEX_POINT_SET`**
41

`vtkmodules.util.vtkConstants.`**`VTK_PARAMETRIC_CURVE`**
51

`vtkmodules.util.vtkConstants.`**`VTK_PARAMETRIC_SURFACE`**
52

`vtkmodules.util.vtkConstants.`**`VTK_PARAMETRIC_TRI_SURFACE`**
53

`vtkmodules.util.vtkConstants.`**`VTK_PARAMETRIC_QUAD_SURFACE`**
54

`vtkmodules.util.vtkConstants.`**`VTK_PARAMETRIC_TETRA_REGION`**
55

`vtkmodules.util.vtkConstants.`**`VTK_PARAMETRIC_HEX_REGION`**
56

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_EDGE`**
60

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_TRIANGLE`**
61

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_QUAD`**
62

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_POLYGON`**
63

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_TETRAHEDRON`**
64

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_WEDGE`**
65

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_PYRAMID`**
66

`vtkmodules.util.vtkConstants.`**`VTK_HIGHER_ORDER_HEXAHEDRON`**
67

`vtkmodules.util.vtkConstants.__vtkTypeNameDict`

None

`vtkmodules.util.vtkConstants.vtkImageScalarTypeNameMacro`(*type*)

Package Contents

Data

__all__

API

`vtkmodules.util.__all__`

['colors', 'misc', 'vtkConstants', 'vtkImageExportToArray', 'vtkImageImportFromArray', 'vtkMethodPar...

`vtkmodules.qt`

Qt module for VTK/Python.

Example usage:

```
import sys
import PyQt5
from PyQt5.QtWidgets import QApplication
from vtkmodules.qt.QVTKRenderWindowInteractor import QVTKRenderWindowInteractor

app = QApplication(sys.argv)

widget = QVTKRenderWindowInteractor()
widget.Initialize()
widget.Start()

renwin = widget.GetRenderWindow()
```

For more information, see QVTKRenderWidgetConeExample() in the file QVTKRenderWindowInteractor.py.

Submodules

`vtkmodules.qt.QVTKRenderWindowInteractor`

A simple VTK widget for PyQt or PySide. See http://www.trolltech.com for Qt documentation, http://www.riverbankcomputing.co.uk for PyQt, and http://pyside.github.io for PySide.

This class is based on the vtkGenericRenderWindowInteractor and is therefore fairly powerful. It should also play nicely with the vtk3DWidget code.

Created by Prabhu Ramachandran, May 2002 Based on David Gobbi's QVTKRenderWidget.py

Changes by Gerard Vermeulen Feb. 2003 Win32 support.

Changes by Gerard Vermeulen, May 2003 Bug fixes and better integration with the Qt framework.

Changes by Phil Thompson, Nov. 2006 Ported to PyQt v4. Added support for wheel events.

Changes by Phil Thompson, Oct. 2007 Bug fixes.

Changes by Phil Thompson, Mar. 2008 Added cursor support.

Changes by Rodrigo Mologni, Sep. 2013 (Credit to Daniele Esposti) Bug fix to PySide: Converts PyCObject to void pointer.

Changes by Greg Schussman, Aug. 2014 The keyPressEvent function now passes keysym instead of None.

Changes by Alex Tsui, Apr. 2015 Port from PyQt4 to PyQt5.

Changes by Fabian Wenzel, Jan. 2016 Support for Python3

Changes by Tobias Hänel, Sep. 2018 Support for PySide2

Changes by Ruben de Bruin, Aug. 2019 Fixes to the keyPressEvent function

Changes by Chen Jintao, Aug. 2021 Support for PySide6

Changes by Eric Larson and Guillaume Favelier, Apr. 2022 Support for PyQt6

Module Contents

Classes

QVTKRenderWindowInteractor	A QVTKRenderWindowInteractor for Python and Qt. Uses a vtkGenericRenderWindowInteracto

Functions

_get_event_pos	
QVTKRenderWidgetConeExample	A simple example that uses the QVTKRenderWindowInteractor class.

Data

QVTKRWIBase
_keysyms_for_ascii
_keysyms

API

`vtkmodules.qt.QVTKRenderWindowInteractor.`**`QVTKRWIBase`**

'QWidget'

`vtkmodules.qt.QVTKRenderWindowInteractor.`**`_get_event_pos`**(*ev*)

`class vtkmodules.qt.QVTKRenderWindowInteractor.`**`QVTKRenderWindowInteractor`**(*parent=None*, ***kw*)

Bases: `vtkmodules.qt.QVTKRenderWindowInteractor.QVTKRWIBaseClass`

A QVTKRenderWindowInteractor for Python and Qt. Uses a vtkGenericRenderWindowInteractor to handle the interactions. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

The user interface is summarized in vtkInteractorStyle.h:

- Keypress j / Keypress t: toggle between joystick (position sensitive) and trackball (motion sensitive) styles. In joystick style, motion occurs continuously as long as a mouse button is pressed. In trackball style, motion occurs when the mouse button is pressed and the mouse pointer moves.
- Keypress c / Keypress o: toggle between camera and object (actor) modes. In camera mode, mouse events affect the camera position and focal point. In object mode, mouse events affect the actor that is under the mouse pointer.
- Button 1: rotate the camera around its focal point (if camera mode) or rotate the actor around its origin (if actor mode). The rotation is in the direction defined from the center of the renderer's viewport towards the mouse position. In joystick mode, the magnitude of the rotation is determined by the distance the mouse is from the center of the render window.
- Button 2: pan the camera (if camera mode) or translate the actor (if object mode). In joystick mode, the direction of pan or translation is from the center of the viewport towards the mouse position. In trackball mode, the direction of motion is the direction the mouse moves. (Note: with 2-button mice, pan is defined as -Button 1.)
- Button 3: zoom the camera (if camera mode) or scale the actor (if object mode). Zoom in/increase scale if the mouse position is in the top half of the viewport; zoom out/decrease scale if the mouse position is in the bottom half. In joystick mode, the amount of zoom is controlled by the distance of the mouse pointer from the horizontal centerline of the window.
- Keypress 3: toggle the render window into and out of stereo mode. By default, red-blue stereo pairs are created. Some systems support Crystal Eyes LCD stereo glasses; you have to invoke SetStereoTypeToCrystalEyes() on the rendering window. Note: to use stereo you also need to pass a stereo=1 keyword argument to the constructor.
- Keypress e: exit the application.
- Keypress f: fly to the picked point
- Keypress p: perform a pick operation. The render window interactor has an internal instance of vtkCellPicker that it uses to pick.
- Keypress r: reset the camera view along the current view direction. Centers the actors and moves the camera so that all actors are visible.
- Keypress s: modify the representation of all actors so that they are surfaces.

- Keypress u: invoke the user-defined function. Typically, this keypress will bring up an interactor that you can type commands in.
- Keypress w: modify the representation of all actors so that they are wireframe.

Initialization

`_CURSOR_MAP`
None

`__getattr__`(*attr*)
Makes the object behave like a vtkGenericRenderWindowInteractor

`Finalize`()
Call internal cleanup method on VTK objects

`CreateTimer`(*obj*, *evt*)

`DestroyTimer`(*obj*, *evt*)

`TimerEvent`()

`CursorChangedEvent`(*obj*, *evt*)
Called when the CursorChangedEvent fires on the render window.

`HideCursor`()
Hides the cursor.

`ShowCursor`()
Shows the cursor.

`closeEvent`(*evt*)

`sizeHint`()

`paintEngine`()

`paintEvent`(*ev*)

`resizeEvent`(*ev*)

`_GetKeyCharAndKeySym`(*ev*)
Convert a Qt key into a char and a vtk keysym.

This is essentially copied from the c++ implementation in GUISupport/Qt/QVTKInteractorAdapter.cxx.

`_GetCtrlShift`(*ev*)

`static _getPixelRatio`()

`_setEventInformation`(*x*, *y*, *ctrl*, *shift*, *key*, *repeat=0*, *keysum=None*)

`enterEvent`(*ev*)

`leaveEvent`(*ev*)

`mousePressEvent`(*ev*)

`mouseReleaseEvent`(*ev*)

mouseMoveEvent(*ev*)

keyPressEvent(*ev*)

keyReleaseEvent(*ev*)

wheelEvent(*ev*)

GetRenderWindow()

Render()

`vtkmodules.qt.QVTKRenderWindowInteractor.`**QVTKRenderWidgetConeExample**()

A simple example that uses the QVTKRenderWindowInteractor class.

`vtkmodules.qt.QVTKRenderWindowInteractor.`**_keysyms_for_ascii**

(None, None, None, None, None, None, None, None, None, 'Tab', None, None, None, None, None, None, No…

`vtkmodules.qt.QVTKRenderWindowInteractor.`**_keysyms**

None

Package Contents

Data

PyQtImpl
QVTKRWIBase
__all__

API

`vtkmodules.qt.`**PyQtImpl**

None

`vtkmodules.qt.`**QVTKRWIBase**

'QWidget'

`vtkmodules.qt.`**__all__**

['QVTKRenderWindowInteractor']

vtkmodules.wx

wxPython widgets for VTK.

Submodules

vtkmodules.wx.wxVTKRenderWindow

A simple VTK widget for wxPython.

Find wxPython info at http://wxPython.org

Created by David Gobbi, December 2001 Based on vtkTkRenderWindget.py

Updated to new wx namespace and some cleaning by Andrea Gavana, December 2006

Module Contents

Classes

wxVTKRenderWindow	A wxRenderWindow for wxPython. Use GetRenderWindow() to get the vtkRenderWindow. Create with the

Functions

wxVTKRenderWindowConeExample	Like it says, just a simple example.

Data

baseClass
_useCapture

API

`vtkmodules.wx.wxVTKRenderWindow.`**`baseClass`**

None

`vtkmodules.wx.wxVTKRenderWindow.`**`_useCapture`**

None

`class` `vtkmodules.wx.wxVTKRenderWindow.`**`wxVTKRenderWindow`**(*parent*, *ID*, **args*, ***kw*)

Bases: *vtkmodules.wx.wxVTKRenderWindow.baseClass*

A wxRenderWindow for wxPython. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

Initialization

Default class constructor. @param parent: parent window @param ID: window id @param **kw: wxPython keywords (position, size, style) plus the 'stereo' keyword

`SetDesiredUpdateRate`(*rate*)
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetDesiredUpdateRate`()
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`SetStillUpdateRate`(*rate*)
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetStillUpdateRate`()
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`OnPaint`(*event*)
: Handles the wx.EVT_PAINT event for wxVTKRenderWindow.

`_OnSize`(*event*)
: Handles the wx.EVT_SIZE event for wxVTKRenderWindow.

`OnSize`(*event*)
: Overridable event.

`OnMove`(*event*)
: Overridable event.

`_OnEnterWindow`(*event*)
: Handles the wx.EVT_ENTER_WINDOW event for wxVTKRenderWindow.

`OnEnterWindow`(*event*)
: Overridable event.

`_OnLeaveWindow`(*event*)
: Handles the wx.EVT_LEAVE_WINDOW event for wxVTKRenderWindow.

`OnLeaveWindow`(*event*)
: Overridable event.

`OnSetFocus`(*event*)
: Overridable event.

`OnKillFocus`(*event*)
: Overridable event.

`_OnButtonDown`(*event*)
: Handles the wx.EVT_LEFT/RIGHT/MIDDLE_DOWN events for wxVTKRenderWindow.

`OnButtonDown`(*event*)
: Overridable event.

`OnLeftDown`(*event*)
: Overridable event.

`OnRightDown`(*event*)
: Overridable event.

`OnMiddleDown`(*event*)
: Overridable event.

`_OnButtonUp`(*event*)
: Handles the wx.EVT_LEFT/RIGHT/MIDDLE_UP events for wxVTKRenderWindow.

`OnButtonUp`(*event*)
: Overridable event.

`OnLeftUp`(*event*)
: Overridable event.

`OnRightUp`(*event*)
: Overridable event.

`OnMiddleUp`(*event*)
: Overridable event.

`OnMotion`(*event*)
: Overridable event.

`OnChar`(*event*)
: Overridable event.

`OnKeyDown`(*event*)
: Handles the wx.EVT_KEY_DOWN events for wxVTKRenderWindow.

`OnKeyUp`(*event*)
: Overridable event.

`GetZoomFactor`()
: Returns the current zoom factor.

`GetRenderWindow`()
: Returns the render window (vtkRenderWindow).

`GetPicker`()
: Returns the current picker (vtkCellPicker).

`Render`()
: Actually renders the VTK scene on screen.

`UpdateRenderer`(*event*)
: UpdateRenderer will identify the renderer under the mouse and set up _CurrentRenderer, _CurrentCamera, and _CurrentLight.

`GetCurrentRenderer`()
: Returns the current renderer.

`Rotate`(*event*)
: Rotates the scene (camera).

`Pan`(*event*)
: Pans the scene (camera).

`Zoom`(*event*)
: Zooms the scene (camera).

Reset(*event=None*)
Resets the camera.

Wireframe()
Sets the current actor representation as wireframe.

Surface()
Sets the current actor representation as surface.

PickActor(*event*)
Picks an actor.

`vtkmodules.wx.wxVTKRenderWindow.`**wxVTKRenderWindowConeExample**()
Like it says, just a simple example.

vtkmodules.wx.wxVTKRenderWindowInteractor

A VTK RenderWindowInteractor widget for wxPython.

Find wxPython info at http://wxPython.org

Created by Prabhu Ramachandran, April 2002 Based on wxVTKRenderWindow.py

Fixes and updates by Charl P. Botha 2003-2008

Updated to new wx namespace and some cleaning up by Andrea Gavana, December 2006

Module Contents

Classes

EventTimer	Simple wx.Timer class.
wxVTKRenderWindowInteractor	A wxRenderWindow for wxPython. Use GetRenderWindow() to get the vtkRenderWindow. Cr

Functions

wxVTKRenderWindowInteractorConeExample	Like it says, just a simple example

Data

baseClass
_useCapture

API

`vtkmodules.wx.wxVTKRenderWindowInteractor.`**`baseClass`**

None

`vtkmodules.wx.wxVTKRenderWindowInteractor.`**`_useCapture`**

None

`class vtkmodules.wx.wxVTKRenderWindowInteractor.`**`EventTimer`**`(`*iren*`)`

Bases: `wx.Timer`

Simple wx.Timer class.

Initialization

Default class constructor. @param iren: current render window

`Notify``()`

The timer has expired.

`class vtkmodules.wx.wxVTKRenderWindowInteractor.`**`wxVTKRenderWindowInteractor`**`(`*parent*, *ID*, **args*, ***kw*`)`

Bases: *`vtkmodules.wx.wxVTKRenderWindowInteractor.baseClass`*

A wxRenderWindow for wxPython. Use GetRenderWindow() to get the vtkRenderWindow. Create with the keyword stereo=1 in order to generate a stereo-capable window.

Initialization

Default class constructor. @param parent: parent window @param ID: window id @param **kw: wxPython keywords (position, size, style) plus the 'stereo' keyword

`USE_STEREO`

False

`BindEvents``()`

Binds all the necessary events for navigation, sizing, drawing.

`__getattr__``(`*attr*`)`

Makes the object behave like a vtkGenericRenderWindowInteractor.

`CreateTimer``(`*obj*, *evt*`)`

Creates a timer.

`DestroyTimer``(`*obj*, *evt*`)`

The timer is a one shot timer so will expire automatically.

`_CursorChangedEvent``(`*obj*, *evt*`)`

Change the wx cursor if the renderwindow's cursor was changed.

`CursorChangedEvent``(`*obj*, *evt*`)`

Called when the CursorChangedEvent fires on the render window.

`HideCursor``()`

Hides the cursor.

`ShowCursor()`

Shows the cursor.

`GetDisplayId()`

Function to get X11 Display ID from WX and return it in a format that can be used by VTK Python.

We query the X11 Display with a new call that was added in wxPython 2.6.0.1. The call returns a SWIG object which we can query for the address and subsequently turn into an old-style SWIG-mangled string representation to pass to VTK.

`OnMouseCaptureLost`(*event*)

This is signalled when we lose mouse capture due to an external event, such as when a dialog box is shown. See the wx documentation.

`OnPaint`(*event*)

Handles the wx.EVT_PAINT event for wxVTKRenderWindowInteractor.

`OnSize`(*event*)

Handles the wx.EVT_SIZE event for wxVTKRenderWindowInteractor.

`OnMotion`(*event*)

Handles the wx.EVT_MOTION event for wxVTKRenderWindowInteractor.

`OnEnter`(*event*)

Handles the wx.EVT_ENTER_WINDOW event for wxVTKRenderWindowInteractor.

`OnLeave`(*event*)

Handles the wx.EVT_LEAVE_WINDOW event for wxVTKRenderWindowInteractor.

`OnButtonDown`(*event*)

Handles the wx.EVT_LEFT/RIGHT/MIDDLE_DOWN events for wxVTKRenderWindowInteractor.

`OnButtonUp`(*event*)

Handles the wx.EVT_LEFT/RIGHT/MIDDLE_UP events for wxVTKRenderWindowInteractor.

`OnMouseWheel`(*event*)

Handles the wx.EVT_MOUSEWHEEL event for wxVTKRenderWindowInteractor.

`OnKeyDown`(*event*)

Handles the wx.EVT_KEY_DOWN event for wxVTKRenderWindowInteractor.

`OnKeyUp`(*event*)

Handles the wx.EVT_KEY_UP event for wxVTKRenderWindowInteractor.

`GetRenderWindow()`

Returns the render window (vtkRenderWindow).

`Render()`

Actually renders the VTK scene on screen.

`SetRenderWhenDisabled`(*newValue*)

Change value of __RenderWhenDisabled ivar.

If __RenderWhenDisabled is false (the default), this widget will not call Render() on the RenderWindow if the top level frame (i.e. the containing frame) has been disabled.

This prevents recursive rendering during wx.SafeYield() calls. wx.SafeYield() can be called during the ProgressMethod() callback of a VTK object to have progress bars and other GUI elements updated - it does this by disabling all windows (disallowing user-input to prevent re-entrancy of code) and then handling all outstanding GUI events.

However, this often triggers an OnPaint() method for wxVTKRWIs, resulting in a Render(), resulting in Update() being called whilst still in progress.

`vtkmodules.wx.wxVTKRenderWindowInteractor.`**`wxVTKRenderWindowInteractorConeExample`**`()`

Like it says, just a simple example

Package Contents

Data

`__all__`

API

`vtkmodules.wx.`**`__all__`**

[‘wxVTKRenderWindow’, ‘wxVTKRenderWindowInteractor’]

`vtkmodules.numpy_interface`

Utility modules for the VTK-Python wrappers.

Submodules

`vtkmodules.numpy_interface.internal_algorithms`

Module Contents

Functions

`_cell_derivatives`	
`_cell_quality`	
`_matrix_math_filter`	
`abs`	Returns the absolute values of an array of scalars/vectors/tensors.
`all`	Returns the min value of an array of scalars/vectors/tensors.
`area`	Returns the surface area of each cell in a mesh.
`aspect`	Returns the aspect ratio of each cell in a mesh.
`aspect_gamma`	Returns the aspect ratio gamma of each cell in a mesh.
`condition`	Returns the condition number of each cell in a mesh.
`cross`	Return the cross product for two 3D vectors from two arrays of 3D vectors.
`curl`	Returns the curl of an array of 3D vectors.
`divergence`	Returns the divergence of an array of 3D vectors.
`det`	Returns the determinant of an array of 2D square matrices.

continues on next page

Table 34 – continued from previous page

determinant	Returns the determinant of an array of 2D square matrices.
diagonal	Returns the diagonal length of each cell in a dataset.
dot	Returns the dot product of two scalars/vectors of two array of scalars/vectors.
eigenvalue	Returns the eigenvalue of an array of 2D square matrices.
eigenvector	Returns the eigenvector of an array of 2D square matrices.
gradient	Returns the gradient of an array of scalars/vectors.
inv	Returns the inverse an array of 2D square matrices.
inverse	Returns the inverse of an array of 2D square matrices.
jacobian	Returns the jacobian of an array of 2D square matrices.
laplacian	Returns the jacobian of an array of scalars.
ln	Returns the natural logarithm of an array of scalars/vectors/tensors.
log	Returns the natural logarithm of an array of scalars/vectors/tensors.
log10	Returns the base 10 logarithm of an array of scalars/vectors/tensors.
max	Returns the maximum value of an array of scalars/vectors/tensors.
max_angle	Returns the maximum angle of each cell in a dataset.
mag	Returns the magnigude of an array of scalars/vectors.
matmul	Return the product of the inputs. Inputs can be vectors/tensors.
mean	Returns the mean value of an array of scalars/vectors/tensors.
min	Returns the min value of an array of scalars/vectors/tensors.
min_angle	Returns the minimum angle of each cell in a dataset.
norm	Returns the normalized values of an array of scalars/vectors.
shear	Returns the shear of each cell in a dataset.
skew	Returns the skew of each cell in a dataset.
strain	Returns the strain of an array of 3D vectors.
sum	Returns the min value of an array of scalars/vectors/tensors.
surface_normal	Returns the surface normal of each cell in a dataset.
trace	Returns the trace of an array of 2D square matrices.
var	Returns the mean value of an array of scalars/vectors/tensors.
volume	Returns the volume of each cell in a dataset.
vorticity	Returns the vorticity/curl of an array of 3D vectors.
vertex_normal	Returns the vertex normal of each point in a dataset.
make_vector	

API

`vtkmodules.numpy_interface.internal_algorithms.`**`_cell_derivatives`**`(`*narray*, *dataset*, *attribute_type*, *filter*`)`

`vtkmodules.numpy_interface.internal_algorithms.`**`_cell_quality`**`(`*dataset*, *quality*`)`

`vtkmodules.numpy_interface.internal_algorithms.`**`_matrix_math_filter`**`(`*narray*, *operation*`)`

`vtkmodules.numpy_interface.internal_algorithms.`**`abs`**`(`*narray*`)`

Returns the absolute values of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`all`**`(`*narray*, *axis=None*`)`

Returns the min value of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`area`**`(`*dataset*`)`

Returns the surface area of each cell in a mesh.

`vtkmodules.numpy_interface.internal_algorithms.`**`aspect`**`(`*dataset*`)`

Returns the aspect ratio of each cell in a mesh.

`vtkmodules.numpy_interface.internal_algorithms.`**`aspect_gamma`**`(`*dataset*`)`

Returns the aspect ratio gamma of each cell in a mesh.

`vtkmodules.numpy_interface.internal_algorithms.`**`condition`**`(`*dataset*`)`

Returns the condition number of each cell in a mesh.

`vtkmodules.numpy_interface.internal_algorithms.`**`cross`**`(`*x*, *y*`)`

Return the cross product for two 3D vectors from two arrays of 3D vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`curl`**`(`*narray*, *dataset=None*`)`

Returns the curl of an array of 3D vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`divergence`**`(`*narray*, *dataset=None*`)`

Returns the divergence of an array of 3D vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`det`**`(`*narray*`)`

Returns the determinant of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`determinant`**`(`*narray*`)`

Returns the determinant of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`diagonal`**`(`*dataset*`)`

Returns the diagonal length of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`dot`**`(`*a1*, *a2*`)`

Returns the dot product of two scalars/vectors of two array of scalars/vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`eigenvalue`**`(`*narray*`)`

Returns the eigenvalue of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`eigenvector`**`(`*narray*`)`

Returns the eigenvector of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`gradient`**`(`*narray*, *dataset=None*`)`

Returns the gradient of an array of scalars/vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`inv`**`(`*narray*`)`

Returns the inverse an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`inverse`**`(`*narray*`)`

Returns the inverse of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`jacobian`**`(`*dataset*`)`

Returns the jacobian of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`laplacian`**`(`*narray*, *dataset=None*`)`

Returns the jacobian of an array of scalars.

`vtkmodules.numpy_interface.internal_algorithms.`**`ln`**`(`*narray*`)`

Returns the natural logarithm of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`log`**`(`*narray*`)`

Returns the natural logarithm of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`log10`**`(`*narray*`)`

Returns the base 10 logarithm of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`max`**`(`*narray*, *axis=None*`)`

Returns the maximum value of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`max_angle`**`(`*dataset*`)`

Returns the maximum angle of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`mag`**`(`*a*`)`

Returns the magnigude of an array of scalars/vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`matmul`**`(`*a*, *b*`)`

Return the product of the inputs. Inputs can be vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`mean`**`(`*narray*, *axis=None*`)`

Returns the mean value of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`min`**`(`*narray*, *axis=None*`)`

Returns the min value of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`min_angle`**`(`*dataset*`)`

Returns the minimum angle of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`norm`**`(`*a*`)`

Returns the normalized values of an array of scalars/vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`shear`**`(`*dataset*`)`

Returns the shear of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`skew`**`(`*dataset*`)`

Returns the skew of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`strain`**`(`*narray*, *dataset=None*`)`

Returns the strain of an array of 3D vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`sum`**`(`*narray*, *axis=None*`)`

Returns the min value of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`surface_normal`**`(`*dataset*`)`

Returns the surface normal of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`trace`**`(`*narray*`)`

Returns the trace of an array of 2D square matrices.

`vtkmodules.numpy_interface.internal_algorithms.`**`var`**`(`*narray*, *axis=None*`)`

Returns the mean value of an array of scalars/vectors/tensors.

`vtkmodules.numpy_interface.internal_algorithms.`**`volume`**`(`*dataset*`)`

Returns the volume of each cell in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`vorticity`**`(`*narray*, *dataset=None*`)`

Returns the vorticity/curl of an array of 3D vectors.

`vtkmodules.numpy_interface.internal_algorithms.`**`vertex_normal`**`(`*dataset*`)`

Returns the vertex normal of each point in a dataset.

`vtkmodules.numpy_interface.internal_algorithms.`**`make_vector`**`(`*ax*, *ay*, *az=None*`)`

`vtkmodules.numpy_interface.dataset_adapter`

This module provides classes that allow Numpy-type access to VTK datasets and arrays. This is best described with some examples.

To normalize a VTK array:

from vtkmodules.vtkImagingCore vtkRTAnalyticSource import vtkmodules.numpy_interface.dataset_adapter as dsa import vtkmodules.numpy_interface.algorithms as algs

rt = vtkRTAnalyticSource() rt.Update() image = dsa.WrapDataObject(rt.GetOutput()) rtdata = image.PointData['RTData'] rtmin = algs.min(rtdata) rtmax = algs.max(rtdata) rtnorm = (rtdata - rtmin) / (rtmax - rtmin) image.PointData.append(rtnorm, 'RTData - normalized') print image.GetPointData().GetArray('RTData - normalized').GetRange()

To calculate gradient:

grad= algs.gradient(rtnorm)

To access subsets:

> grad[0:10] VTKArray([[0.10729134, 0.03763443, 0.03136338], [0.02754352, 0.03886006, 0.032589], [0.02248248, 0.04127144, 0.03500038], [0.02678365, 0.04357527, 0.03730421], [0.01765099, 0.04571581, 0.03944477], [0.02344007, 0.04763837, 0.04136734], [0.01089381, 0.04929155, 0.04302051], [0.01769151, 0.05062952, 0.04435848], [0.002764 , 0.05161414, 0.04534309], [0.01010841, 0.05221677, 0.04594573]])
>
> grad[:, 0] VTKArray([0.10729134, 0.02754352, 0.02248248, ..., -0.02748174, -0.02410045, 0.05509736])

All of this functionality is also supported for composite datasets even though their data arrays may be spread across multiple datasets. We have implemented a VTKCompositeDataArray class that handles many Numpy style operators and is supported by all algorithms in the algorithms module.

This module also provides an API to access composite datasets. For example:

from vtkmodules.vtkCommonDataModel import vtkMultiBlockDataSet mb = vtkMultiBlockDataSet() mb.SetBlock(0, image.VTKObject) mb.SetBlock(1e, image.VTKObject) cds = dsa.WrapDataObject(mb) for block in cds: print block

Note that this module implements only the wrappers for datasets and arrays. The classes implement many useful operators. However, to make best use of these classes, take a look at the algorithms module.

Module Contents

Classes

ArrayAssociation	Easy access to vtkDataObject.AttributeTypes
VTKObjectWrapper	Superclass for classes that wrap VTK objects with Python objects. This class holds a refere
VTKArrayMetaClass	
VTKArray	This is a sub-class of numpy ndarray that stores a reference to a vtk array as well as the own
VTKNoneArrayMetaClass	
VTKNoneArray	VTKNoneArray is used to represent a “void” array. An instance of this class (NoneArray) is
VTKCompositeDataArrayMetaClass	
VTKCompositeDataArray	This class manages a set of arrays of the same name contained within a composite dataset. I

DataSetAttributes	This is a python friendly wrapper of vtkDataSetAttributes. It returns VTKArrays. It also pro
CompositeDataSetAttributes	This is a python friendly wrapper for vtkDataSetAttributes for composite datasets. Since co
CompositeDataIterator	Wrapper for a vtkCompositeDataIterator class to satisfy the python iterator protocol. This it
MultiCompositeDataIterator	Iterator that can be used to iterate over multiple composite datasets together. This iterator w
DataObject	A wrapper for vtkDataObject that makes it easier to access FielData arrays as VTKArrays
Table	A wrapper for vtkTable that makes it easier to access RowData array as VTKArrays
HyperTreeGrid	A wrapper for vtkHyperTreeGrid that makes it easier to access CellData arrays as VTKArra
CompositeDataSet	A wrapper for vtkCompositeData and subclasses that makes it easier to access Point/Cell/Fi
DataSet	This is a python friendly wrapper of a vtkDataSet that defines a few useful properties.
PointSet	This is a python friendly wrapper of a vtkPointSet that defines a few useful properties.
PolyData	This is a python friendly wrapper of a vtkPolyData that defines a few useful properties.
UnstructuredGrid	This is a python friendly wrapper of a vtkUnstructuredGrid that defines a few useful propert
Graph	This is a python friendly wrapper of a vtkGraph that defines a few useful properties.
Molecule	This is a python friendly wrapper of a vtkMolecule that defines a few useful properties.

Functions

reshape_append_ones	Returns a list with the two arguments, any of them may be processed. If the arguments are nu
vtkDataArrayToVTKArray	Given a vtkDataArray and a dataset owning it, returns a VTKArray.
numpyTovtkDataArray	Given a numpy array or a VTKArray and a name, returns a vtkDataArray. The resulting vtkD
_make_tensor_array_contiguous	
_metaclass	For compatibility between python 2 and python 3.
WrapDataObject	Returns a Numpy friendly wrapper of a vtkDataObject.

Data

NoneArray

API

`vtkmodules.numpy_interface.dataset_adapter.`**`reshape_append_ones`**`(`*a1*, *a2*`)`

Returns a list with the two arguments, any of them may be processed. If the arguments are numpy.ndarrays, append 1s to the shape of the array with the smallest number of dimensions until the arrays have the same number of dimensions. Does nothing if the arguments are not ndarrays or the arrays have the same number of dimensions.

`class vtkmodules.numpy_interface.dataset_adapter.`**`ArrayAssociation`**

Easy access to vtkDataObject.AttributeTypes

`POINT`

None

`CELL`

None

`FIELD`

None

`ROW`

None

`class vtkmodules.numpy_interface.dataset_adapter.`**`VTKObjectWrapper`**`(`*vtkobject*`)`

Bases: `object`

Superclass for classes that wrap VTK objects with Python objects. This class holds a reference to the wrapped VTK object. It also forwards unresolved methods to the underlying object by overloading __get__attr.

Initialization

`__getattr__``(`*name*`)`

Forwards unknown attribute requests to VTK object.

`vtkmodules.numpy_interface.dataset_adapter.`**`vtkDataArrayToVTKArray`**`(`*array*, *dataset=None*`)`

Given a vtkDataArray and a dataset owning it, returns a VTKArray.

`vtkmodules.numpy_interface.dataset_adapter.`**`numpyTovtkDataArray`**`(`*array*, *name='numpy_array'*, *array_type=None*`)`

Given a numpy array or a VTKArray and a name, returns a vtkDataArray. The resulting vtkDataArray will store a reference to the numpy array: the numpy array is released only when the vtkDataArray is destroyed.

`vtkmodules.numpy_interface.dataset_adapter.`**`_make_tensor_array_contiguous`**`(`*array*`)`

`vtkmodules.numpy_interface.dataset_adapter.`**`_metaclass`**`(`*mcs*`)`

For compatibility between python 2 and python 3.

`class vtkmodules.numpy_interface.dataset_adapter.`**`VTKArrayMetaClass`**

Bases: `type`

`__new__``(`*name*, *parent*, *attr*`)`

We overwrite numerical/comparison operators because we might need to reshape one of the arrays to perform the operation without broadcast errors. For instance:

An array G of shape (n,3) resulted from computing the gradient on a scalar array S of shape (n,) cannot be added together without reshaping. G + expand_dims(S,1) works, G + S gives an error: ValueError: operands could not be broadcast together with shapes (n,3) (n,)

This metaclass overwrites operators such that it computes this reshape operation automatically by appending 1s to the dimensions of the array with fewer dimensions.

`class vtkmodules.numpy_interface.dataset_adapter.`**`VTKArray`**`(`*shape*, *dtype=float*, *buffer=None*, *offset=0*, *strides=None*, *order=None*`)`

Bases: `numpy.ndarray`

This is a sub-class of numpy ndarray that stores a reference to a vtk array as well as the owning dataset. The numpy array and vtk array should point to the same memory location.

Initialization

_numeric_op(*other*, *attr_name*)

Used to implement numpy-style numerical operations such as **add**, **mul**, etc.

_reverse_numeric_op(*other*, *attr_name*)

Used to implement numpy-style numerical operations such as **add**, **mul**, etc.

__new__(*input_array*, *array=None*, *dataset=None*)

__array_finalize__(*obj*)

__getattr__(*name*)

Forwards unknown attribute requests to VTK array.

__array_wrap__(*out_arr*, *context=None*)

property **DataSet**

Get the dataset this array is associated with. The reference to the dataset is held through a vtkWeakReference to ensure it doesn't prevent the dataset from being collected if necessary.

class vtkmodules.numpy_interface.dataset_adapter.**VTKNoneArrayMetaClass**

Bases: `type`

__new__(*name*, *parent*, *attr*)

Simplify the implementation of the numeric/logical sequence API.

class vtkmodules.numpy_interface.dataset_adapter.**VTKNoneArray**

Bases: `object`

VTKNoneArray is used to represent a "void" array. An instance of this class (NoneArray) is returned instead of None when an array that doesn't exist in a DataSetAttributes is requested. All operations on the NoneArray return NoneArray. The main reason for this is to support operations in parallel where one of the processes may be working on an empty dataset. In such cases, the process is still expected to evaluate a whole expression because some of the functions may perform bulk MPI communication. None cannot be used in these instances because it cannot properly override operators such as **add**, **sub** etc. This is the main raison d'etre for VTKNoneArray.

__getitem__(*index*)

_op(*other*, *op*)

Used to implement numpy-style numerical operations such as **add**, **mul**, etc.

astype(*dtype*)

Implements numpy array's astype method.

vtkmodules.numpy_interface.dataset_adapter.**NoneArray**

'VTKNoneArray(...)'

class vtkmodules.numpy_interface.dataset_adapter.**VTKCompositeDataArrayMetaClass**

Bases: `type`

__new__(*name*, *parent*, *attr*)

Simplify the implementation of the numeric/logical sequence API.

class vtkmodules.numpy_interface.dataset_adapter.**VTKCompositeDataArray**(*arrays=[]*, *dataset=None*, *name=None*, *association=None*)

Bases: `object`

This class manages a set of arrays of the same name contained within a composite dataset. Its main purpose is to provide a Numpy-type interface to composite data arrays which are naturally nothing but a collection of vtkDataArrays. A VTKCompositeDataArray makes such a collection appear as a single Numpy array and support all array operations that this module and the associated algorithm module support. Note that this is not a subclass of a Numpy array and as such cannot be passed to native Numpy functions. Instead VTK modules should be used to process composite arrays.

Initialization

Construct a composite array given a container of arrays, a dataset, name and association. It is sufficient to define a container of arrays to define a composite array. It is also possible to initialize an array by defining the dataset, name and array association. In that case, the underlying arrays will be created lazily when they are needed. It is recommended to use the latter method when initializing from an existing composite dataset.

`__init_from_composite``()`

`GetSize``()`

Returns the number of elements in the array.

`size`

'property(. . .)'

`GetArrays``()`

Returns the internal container of VTKArrays. If necessary, this will populate the array list from a composite dataset.

`Arrays`

'property(. . .)'

`__getitem__``(`*index*`)`

Overwritten to refer indexing to underlying VTKArrays. For the most part, this will behave like Numpy. Note that indexing is done per array - arrays are never treated as forming a bigger array. If the index is another composite array, a one-to-one mapping between arrays is assumed.

`_numeric_op``(`*other*, *op*`)`

Used to implement numpy-style numerical operations such as **add**, **mul**, etc.

`_reverse_numeric_op``(`*other*, *op*`)`

Used to implement numpy-style numerical operations such as **add**, **mul**, etc.

`__str__``()`

`astype``(`*dtype*`)`

Implements numpy array's as array method.

`class vtkmodules.numpy_interface.dataset_adapter.`**`DataSetAttributes`**`(`*vtkobject*, *dataset*, *association*`)`

Bases: `vtkmodules.numpy_interface.dataset_adapter.VTKObjectWrapper`

This is a python friendly wrapper of vtkDataSetAttributes. It returns VTKArrays. It also provides the dictionary interface. Note that the stored array should have a shape that matches the number of elements. E.g. for a PointData, narray.shape[0] should be equal to dataset.GetNumberOfPoints()

Initialization

`__getitem__`(*idx*)

Implements the [] operator. Accepts an array name or index.

`GetArray`(*idx*)

Given an index or name, returns a VTKArray.

`keys`()

Returns the names of the arrays as a list.

`values`()

Returns the arrays as a list.

`PassData`(*other*)

A wrapper for vtkDataSet.PassData.

`append`(*narray*, *name*)

Appends narray to the dataset attributes.

If narray is a scalar, create an array with this scalar for each element. If narray is an array with a size not matching the array association (e.g. size should be equal to GetNumberOfPoints() for PointData), copy the input narray for each element. This is intended to ease initialization, typically using same 3d vector for each element. In any case, be careful about memory explosion.

`class vtkmodules.numpy_interface.dataset_adapter.`**`CompositeDataSetAttributes`**(*dataset*, *association*)

This is a python friendly wrapper for vtkDataSetAttributes for composite datasets. Since composite datasets themselves don't have attribute data, but the attribute data is associated with the leaf nodes in the composite dataset, this class simulates a DataSetAttributes interface by taking a union of DataSetAttributes associated with all leaf nodes.

Initialization

`__determine_arraynames`()

`keys`()

Returns the names of the arrays as a list.

`__getitem__`(*idx*)

Implements the [] operator. Accepts an array name.

`append`(*narray*, *name*)

Appends a new array to the composite dataset attributes.

`GetArray`(*idx*)

Given a name, returns a VTKCompositeArray.

`PassData`(*other*)

Emulate PassData for composite datasets.

`class vtkmodules.numpy_interface.dataset_adapter.`**`CompositeDataIterator`**(*cds*)

Bases: `object`

Wrapper for a vtkCompositeDataIterator class to satisfy the python iterator protocol. This iterator iterates over non-empty leaf nodes. To iterate over empty or non-leaf nodes, use the vtkCompositeDataIterator directly.

Initialization

`__iter__`()

`__next__`()

`next`()

`__getattr__`(*name*)

Returns attributes from the vtkCompositeDataIterator.

class `vtkmodules.numpy_interface.dataset_adapter.`**`MultiCompositeDataIterator`**(*cds*)

Bases: *`vtkmodules.numpy_interface.dataset_adapter.CompositeDataIterator`*

Iterator that can be used to iterate over multiple composite datasets together. This iterator works only with arrays that were copied from an original using CopyStructured. The most common use case is to use CopyStructure, then iterate over input and output together while creating output datasets from corresponding input datasets.

Initialization

`__next__`()

`next`()

class `vtkmodules.numpy_interface.dataset_adapter.`**`DataObject`**(*vtkobject*)

Bases: *`vtkmodules.numpy_interface.dataset_adapter.VTKObjectWrapper`*

A wrapper for vtkDataObject that makes it easier to access FielData arrays as VTKArrays

Initialization

`GetAttributes`(*type*)

Returns the attributes specified by the type as a DataSetAttributes instance.

`HasAttributes`(*type*)

Returns if current object support this attributes type

`GetFieldData`()

Returns the field data as a DataSetAttributes instance.

`FieldData`

'property(...)'

class `vtkmodules.numpy_interface.dataset_adapter.`**`Table`**(*vtkobject*)

Bases: *`vtkmodules.numpy_interface.dataset_adapter.DataObject`*

A wrapper for vtkTable that makes it easier to access RowData array as VTKArrays

Initialization

`GetRowData()`

Returns the row data as a DataSetAttributes instance.

`HasAttributes(`*type*`)`

Returns if current object support this attributes type

`RowData`

'property(. . .)'

`class vtkmodules.numpy_interface.dataset_adapter.`**`HyperTreeGrid`**`(`*vtkobject*`)`

Bases: *`vtkmodules.numpy_interface.dataset_adapter.DataObject`*

A wrapper for vtkHyperTreeGrid that makes it easier to access CellData arrays as VTKArrays.

Initialization

`GetCellData()`

Returns the cell data as DataSetAttributes instance.

`HasAttributes(`*type*`)`

Returns if current object support this attributes type

`CellData`

'property(. . .)'

`class vtkmodules.numpy_interface.dataset_adapter.`**`CompositeDataSet`**`(`*vtkobject*`)`

Bases: *`vtkmodules.numpy_interface.dataset_adapter.DataObject`*

A wrapper for vtkCompositeData and subclasses that makes it easier to access Point/Cell/Field data as VTK-CompositeDataArrays. It also provides a Python type iterator.

Initialization

`__iter__()`

Creates an iterator for the contained datasets.

`GetNumberOfElements(`*assoc*`)`

Returns the total number of cells or points depending on the value of assoc which can be ArrayAssociation.POINT or ArrayAssociation.CELL.

`GetNumberOfPoints()`

Returns the total number of points of all datasets in the composite dataset. Note that this traverses the whole composite dataset every time and should not be called repeatedly for large composite datasets.

`GetNumberOfCells()`

Returns the total number of cells of all datasets in the composite dataset. Note that this traverses the whole composite dataset every time and should not be called repeatedly for large composite datasets.

`GetAttributes(`*type*`)`

Returns the attributes specified by the type as a CompositeDataSetAttributes instance.

`HasAttributes(`*type*`)`

Returns true if every leaves of current composite object support this attributes type

GetPointData()

Returns the point data as a DataSetAttributes instance.

GetCellData()

Returns the cell data as a DataSetAttributes instance.

GetFieldData()

Returns the field data as a DataSetAttributes instance.

GetPoints()

Returns the points as a VTKCompositeDataArray instance.

PointData

'property(...)'

CellData

'property(...)'

FieldData

'property(...)'

Points

'property(...)'

class vtkmodules.numpy_interface.dataset_adapter.**DataSet**(*vtkobject*)

Bases: *vtkmodules.numpy_interface.dataset_adapter.DataObject*

This is a python friendly wrapper of a vtkDataSet that defines a few useful properties.

Initialization

GetPointData()

Returns the point data as a DataSetAttributes instance.

GetCellData()

Returns the cell data as a DataSetAttributes instance.

HasAttributes(*type*)

Returns if current object support this attributes type

PointData

'property(...)'

CellData

'property(...)'

class vtkmodules.numpy_interface.dataset_adapter.**PointSet**(*vtkobject*)

Bases: *vtkmodules.numpy_interface.dataset_adapter.DataSet*

This is a python friendly wrapper of a vtkPointSet that defines a few useful properties.

Initialization

GetPoints()

Returns the points as a VTKArray instance. Returns None if the dataset has implicit points.

SetPoints(*pts*)

Given a VTKArray instance, sets the points of the dataset.

Points

'property(. . .)'

class vtkmodules.numpy_interface.dataset_adapter.**PolyData**(*vtkobject*)

Bases: *vtkmodules.numpy_interface.dataset_adapter.PointSet*

This is a python friendly wrapper of a vtkPolyData that defines a few useful properties.

Initialization

GetPolygons()

Returns the polys as a VTKArray instance.

Polygons

'property(. . .)'

class vtkmodules.numpy_interface.dataset_adapter.**UnstructuredGrid**(*vtkobject*)

Bases: *vtkmodules.numpy_interface.dataset_adapter.PointSet*

This is a python friendly wrapper of a vtkUnstructuredGrid that defines a few useful properties.

Initialization

GetCellTypes()

Returns the cell types as a VTKArray instance.

GetCellLocations()

Returns the cell locations as a VTKArray instance.

GetCells()

Returns the cells as a VTKArray instance.

SetCells(*cellTypes*, *cellLocations*, *cells*)

Given cellTypes, cellLocations, cells as VTKArrays, populates the unstructured grid data structures.

CellTypes

'property(. . .)'

CellLocations

'property(. . .)'

Cells

'property(. . .)'

class vtkmodules.numpy_interface.dataset_adapter.**Graph**(*vtkobject*)

Bases: *vtkmodules.numpy_interface.dataset_adapter.DataObject*

This is a python friendly wrapper of a vtkGraph that defines a few useful properties.

Initialization

GetVertexData()

Returns the vertex data as a DataSetAttributes instance.

GetEdgeData()

Returns the edge data as a DataSetAttributes instance.

VertexData

'property(. . .)'

EdgeData

'property(. . .)'

class vtkmodules.numpy_interface.dataset_adapter.**Molecule**(*vtkobject*)

Bases: *vtkmodules.numpy_interface.dataset_adapter.DataObject*

This is a python friendly wrapper of a vtkMolecule that defines a few useful properties.

Initialization

GetAtomData()

Returns the atom data as a DataSetAttributes instance.

GetBondData()

Returns the bond data as a DataSetAttributes instance.

AtomData

'property(. . .)'

BondData

'property(. . .)'

vtkmodules.numpy_interface.dataset_adapter.**WrapDataObject**(*ds*)

Returns a Numpy friendly wrapper of a vtkDataObject.

vtkmodules.numpy_interface.algorithms

This module provides a number of algorithms that can be used with the dataset classes defined in the dataset_adapter module. See the documentation of the dataset_adapter for some examples. These algorithms work in serial and in parallel as long as the data is partitioned according to VTK data parallel execution guidelines. For details, see the documentation of individual algorithms.

Module Contents

Functions

_apply_func2	Apply a function to each member of a VTKCompositeDataArray. Returns a list of arra
apply_ufunc	Apply a function to each member of a VTKCompositeDataArray. VTKArray and num
_make_ufunc	Given a ufunc, creates a closure that applies it to each member of a VTKCompositeDa
apply_dfunc	Apply a two argument function to each member of a VTKCompositeDataArray and an

_make_dfunc	Given a function that requires two arguments, creates a closure that applies it to each m
_make_dsfunc	Given a function that requires two arguments (one array, one dataset), creates a closure
_make_dsfunc2	Given a function that requires a dataset, creates a closure that applies it to each membe
_lookup_mpi_type	
_reduce_dims	
_global_func	
bitwise_or	Implements element by element or (bitwise, | in C/C++) operation. If one of the arrays
make_point_mask_from_NaNs	This method will create a ghost array corresponding to an input with NaN values. For
make_cell_mask_from_NaNs	This method will create a ghost array corresponding to an input with NaN values. For
make_mask_from_NaNs	This method will create a ghost array corresponding to an input with NaN values. For
sum	Returns the sum of all values along a particular axis (dimension). Given an array of m
max	Returns the max of all values along a particular axis (dimension). Given an array of m
min	Returns the min of all values along a particular axis (dimension). Given an array of m
_global_per_block	
sum_per_block	Returns the sum of all values along a particular axis (dimension) for each block of an V
count_per_block	Return the number of elements of each block in a VTKCompositeDataArray along an a
mean_per_block	Returns the mean of all values along a particular axis (dimension) for each block of a V
max_per_block	Returns the max of all values along a particular axis (dimension) for each block of a V
min_per_block	Returns the min of all values along a particular axis (dimension) for each block of a V
all	Returns True if all values of an array evaluate to True, returns False otherwise. This is
_local_array_count	
_array_count	
mean	Returns the mean of all values along a particular axis (dimension). Given an array of n
var	Returns the variance of all values along a particular axis (dimension). Given an array o
std	Returns the standard deviation of all values along a particular axis (dimension). Given
shape	Returns the shape (dimensions) of an array.
make_vector	Given 2 or 3 scalar arrays, returns a vector array. If only 2 scalars are provided, the thi
unstructured_from_composite_arrays	Given a set of VTKCompositeDataArrays, creates a vtkUnstructuredGrid. The main g

Data

in1d
isnan
sqrt
negative
reciprocal
square
exp
floor
ceil
rint
sin
cos
tan
arcsin
arccos
arctan
arctan2
sinh

continues on next page

Table 39 – continued from previous page

cosh
tanh
arcsinh
arccosh
arctanh
where
flatnonzero
nonzero
expand_dims
abs
area
aspect
aspect_gamma
condition
cross
curl
divergence
det
determinant
diagonal
dot
eigenvalue
eigenvector
gradient
inv
inverse
jacobian
laplacian
ln
log
log10
max_angle
mag
matmul
min_angle
norm
shear
skew
strain
surface_normal
trace
volume
vorticity
vertex_normal
logical_not
divide
multiply
add
subtract
mod
remainder
power

continues on next page

Table 39 – continued from previous page

hypot

API

`vtkmodules.numpy_interface.algorithms._apply_func2(func, array, args)`

Apply a function to each member of a VTKCompositeDataArray. Returns a list of arrays.

Note that this function is mainly for internal use by this module.

`vtkmodules.numpy_interface.algorithms.apply_ufunc(func, array, args=())`

Apply a function to each member of a VTKCompositeDataArray. VTKArray and numpy arrays are also supported.

`vtkmodules.numpy_interface.algorithms._make_ufunc(ufunc)`

Given a ufunc, creates a closure that applies it to each member of a VTKCompositeDataArray.

Note that this function is mainly for internal use by this module.

`vtkmodules.numpy_interface.algorithms.apply_dfunc(dfunc, array1, val2)`

Apply a two argument function to each member of a VTKCompositeDataArray and another argument The second argument can be a VTKCompositeDataArray, in which case a one-to-one match between arrays is assumed. Otherwise, the function is applied to the composite array with the second argument repeated. VTKArray and numpy arrays are also supported.

`vtkmodules.numpy_interface.algorithms._make_dfunc(dfunc)`

Given a function that requires two arguments, creates a closure that applies it to each member of a VTKCompositeDataArray.

Note that this function is mainly for internal use by this module.

`vtkmodules.numpy_interface.algorithms._make_dsfunc(dsfunc)`

Given a function that requires two arguments (one array, one dataset), creates a closure that applies it to each member of a VTKCompositeDataArray. Note that this function is mainly for internal use by this module.

`vtkmodules.numpy_interface.algorithms._make_dsfunc2(dsfunc)`

Given a function that requires a dataset, creates a closure that applies it to each member of a VTKCompositeDataArray.

Note that this function is mainly for internal use by this module.

`vtkmodules.numpy_interface.algorithms._lookup_mpi_type(ntype)`

`vtkmodules.numpy_interface.algorithms._reduce_dims(array, comm)`

`vtkmodules.numpy_interface.algorithms._global_func(impl, array, axis, controller)`

`vtkmodules.numpy_interface.algorithms.bitwise_or(array1, array2)`

Implements element by element or (bitwise, | in C/C++) operation. If one of the arrays is a NoneArray, this will return the array that is not NoneArray, treating NoneArray as 0 in the or operation.

`vtkmodules.numpy_interface.algorithms.make_point_mask_from_NaNs(dataset, array)`

This method will create a ghost array corresponding to an input with NaN values. For each NaN value, the output array will have a corresponding value of vtkmodules.vtkCommonDataModel.vtkDataSetAttributes.HIDDENPOINT. These values are also combined with any ghost values that the dataset may have.

`vtkmodules.numpy_interface.algorithms.`**`make_cell_mask_from_NaNs`**`(`*dataset*, *array*`)`

This method will create a ghost array corresponding to an input with NaN values. For each NaN value, the output array will have a corresponding value of vtkmodules.vtkCommonDataModel.vtkDataSetAttributes.HIDDENCELL. These values are also combined with any ghost values that the dataset may have.

`vtkmodules.numpy_interface.algorithms.`**`make_mask_from_NaNs`**`(`*array*, *ghost_array=dsa.NoneArray*, *is_cell=False*`)`

This method will create a ghost array corresponding to an input with NaN values. For each NaN value, the output array will have a corresponding value of vtkmodules.vtkCommonDataModel.vtkDataSetAttributes.HIDDENPOINT or HIDDENCELL is the is_cell argument is true. If an input ghost_array is passed, the array is bitwise_or'ed with it, simply adding the new ghost values to it.

`vtkmodules.numpy_interface.algorithms.`**`sum`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the sum of all values along a particular axis (dimension). Given an array of m tuples and n components:

- Default is to return the sum of all values in an array.
- axis=0: Sum values of all components and return a one tuple, n-component array.
- axis=1: Sum values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will sum across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

sum(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`max`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the max of all values along a particular axis (dimension). Given an array of m tuples and n components:

- Default is to return the max of all values in an array.
- axis=0: Return the max values of all tuples and return a one tuple, n-component array.
- axis=1: Return the max values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the max across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

max(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`min`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the min of all values along a particular axis (dimension). Given an array of m tuples and n components:

- Default is to return the min of all values in an array.
- axis=0: Return the min values of all tuples and return a one tuple, n-component array.
- axis=1: Return the min values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the min across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

min(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`_global_per_block`**`(`*impl*, *array*, *axis=None*, *controller=None*`)` `vtkmodules.numpy_interface.algorithms.`**`sum_per_block`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the sum of all values along a particular axis (dimension) for each block of an VTKCompositeDataArray.

Given an array of m tuples and n components:

- Default is to return the sum of all values in an array.
- axis=0: Sum values of all components and return a one tuple, n-component array.
- axis=1: Sum values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will sum across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

sum_per_block(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`count_per_block`**`(`*array*, *axis=None*, *controller=None*`)`

Return the number of elements of each block in a VTKCompositeDataArray along an axis.

- if axis is None, the number of all elements (ntuples * ncomponents) is returned.
- if axis is 0, the number of tuples is returned.

`vtkmodules.numpy_interface.algorithms.`**`mean_per_block`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the mean of all values along a particular axis (dimension) for each block of a VTKCompositeDataArray.

Given an array of m tuples and n components:

- Default is to return the mean of all values in an array.
- axis=0: Return the mean values of all components and return a one tuple, n-component array.
- axis=1: Return the mean values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the mean across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

mean(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`max_per_block`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the max of all values along a particular axis (dimension) for each block of a VTKCompositeDataArray. Given an array of m tuples and n components:

- Default is to return the max of all values in an array.
- axis=0: Return the max values of all components and return a one tuple, n-component array.
- axis=1: Return the max values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the max across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

max_per_block(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`min_per_block`**`(`*array*, *axis=None*, *controller=None*`)`

Returns the min of all values along a particular axis (dimension) for each block of a VTKCompositeDataArray. Given an array of m tuples and n components:

- Default is to return the min of all values in an array.
- axis=0: Return the min values of all components and return a one tuple, n-component array.
- axis=1: Return the min values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the min across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

min_per_block(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`all`**(*array*, *axis=None*, *controller=None*)

Returns True if all values of an array evaluate to True, returns False otherwise. This is useful to check if all values of an array match a certain condition such as:

algorithms.all(array > 5)

`vtkmodules.numpy_interface.algorithms.`**`_local_array_count`**(*array*, *axis*)

`vtkmodules.numpy_interface.algorithms.`**`_array_count`**(*array*, *axis*, *controller*)

`vtkmodules.numpy_interface.algorithms.`**`mean`**(*array*, *axis=None*, *controller=None*, *size=None*)

Returns the mean of all values along a particular axis (dimension). Given an array of m tuples and n components:

- Default is to return the mean of all values in an array.
- axis=0: Return the mean values of all components and return a one tuple, n-component array.
- axis=1: Return the mean values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the mean across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

mean(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`var`**(*array*, *axis=None*, *controller=None*)

Returns the variance of all values along a particular axis (dimension). Given an array of m tuples and n components:

- Default is to return the variance of all values in an array.
- axis=0: Return the variance values of all components and return a one tuple, n-component array.
- axis=1: Return the variance values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the variance across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

var(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`std`**(*array*, *axis=None*, *controller=None*)

Returns the standard deviation of all values along a particular axis (dimension). Given an array of m tuples and n components:

- Default is to return the standard deviation of all values in an array.
- axis=0: Return the standard deviation values of all components and return a one tuple, n-component array.
- axis=1: Return the standard deviation values of all components of each tuple and return an m-tuple, 1-component array.

When called in parallel, this function will compute the standard deviation across processes when a controller argument is passed or the global controller is defined. To disable parallel summing when running in parallel, pass a dummy controller as follows:

std(array, controller=vtkmodules.vtkParallelCore.vtkDummyController()).

`vtkmodules.numpy_interface.algorithms.`**`shape`**`(`*array*`)`

Returns the shape (dimensions) of an array.

`vtkmodules.numpy_interface.algorithms.`**`make_vector`**`(`*arrayx*, *arrayy*, *arrayz=None*`)`

Given 2 or 3 scalar arrays, returns a vector array. If only 2 scalars are provided, the third component will be set to 0.

`vtkmodules.numpy_interface.algorithms.`**`unstructured_from_composite_arrays`**`(`*points*, *arrays*, *controller=None*`)`

Given a set of VTKCompositeDataArrays, creates a vtkUnstructuredGrid. The main goal of this function is to transform the output of XXX_per_block() methods to a single dataset that can be visualized and further processed. Here arrays is an iterable (e.g. list) of (array, name) pairs. Here is an example:

centroid = mean_per_block(composite_data.Points) T = mean_per_block(composite_data.PointData['Temperature']) ug = unstructured_from_composite_arrays(centroid, (T, 'Temperature'))

When called in parallel, this function makes sure that each array in the input dataset is represented only on 1 process. This is important because methods like mean_per_block() return the same value for blocks that are partitioned on all of the participating processes. If the same point were to be created across multiple processes in the output, filters like histogram would report duplicate values erroneously.

`vtkmodules.numpy_interface.algorithms.`**`in1d`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`isnan`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`sqrt`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`negative`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`reciprocal`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`square`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`exp`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`floor`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`ceil`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`rint`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`sin`**

'_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`cos`**

'_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**tan**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arcsin**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arccos**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arctan**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arctan2**
  '_make_dfunc(...)'

vtkmodules.numpy_interface.algorithms.**sinh**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**cosh**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**tanh**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arcsinh**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arccosh**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**arctanh**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**where**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**flatnonzero**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**nonzero**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**expand_dims**
  '_make_dfunc(...)'

vtkmodules.numpy_interface.algorithms.**abs**
  '_make_ufunc(...)'

vtkmodules.numpy_interface.algorithms.**area**
  '_make_dsfunc2(...)'

vtkmodules.numpy_interface.algorithms.**aspect**
  '_make_dsfunc2(...)'

vtkmodules.numpy_interface.algorithms.**aspect_gamma**
  '_make_dsfunc2(...)'

vtkmodules.numpy_interface.algorithms.**condition**
    '_make_dsfunc2(…)'

vtkmodules.numpy_interface.algorithms.**cross**
    '_make_dfunc(…)'

vtkmodules.numpy_interface.algorithms.**curl**
    '_make_dsfunc(…)'

vtkmodules.numpy_interface.algorithms.**divergence**
    '_make_dsfunc(…)'

vtkmodules.numpy_interface.algorithms.**det**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**determinant**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**diagonal**
    '_make_dsfunc2(…)'

vtkmodules.numpy_interface.algorithms.**dot**
    '_make_dfunc(…)'

vtkmodules.numpy_interface.algorithms.**eigenvalue**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**eigenvector**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**gradient**
    '_make_dsfunc(…)'

vtkmodules.numpy_interface.algorithms.**inv**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**inverse**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**jacobian**
    '_make_dsfunc2(…)'

vtkmodules.numpy_interface.algorithms.**laplacian**
    '_make_dsfunc(…)'

vtkmodules.numpy_interface.algorithms.**ln**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**log**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**log10**
    '_make_ufunc(…)'

vtkmodules.numpy_interface.algorithms.**max_angle**
    '_make_dsfunc2(…)'

`vtkmodules.numpy_interface.algorithms.`**`mag`**
    '_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`matmul`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`min_angle`**
    '_make_dsfunc2(...)'

`vtkmodules.numpy_interface.algorithms.`**`norm`**
    '_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`shear`**
    '_make_dsfunc2(...)'

`vtkmodules.numpy_interface.algorithms.`**`skew`**
    '_make_dsfunc2(...)'

`vtkmodules.numpy_interface.algorithms.`**`strain`**
    '_make_dsfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`surface_normal`**
    '_make_dsfunc2(...)'

`vtkmodules.numpy_interface.algorithms.`**`trace`**
    '_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`volume`**
    '_make_dsfunc2(...)'

`vtkmodules.numpy_interface.algorithms.`**`vorticity`**
    '_make_dsfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`vertex_normal`**
    '_make_dsfunc2(...)'

`vtkmodules.numpy_interface.algorithms.`**`logical_not`**
    '_make_ufunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`divide`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`multiply`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`add`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`subtract`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`mod`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`remainder`**
    '_make_dfunc(...)'

`vtkmodules.numpy_interface.algorithms.`**`power`**

‘_make_dfunc(…)’

`vtkmodules.numpy_interface.algorithms.`**`hypot`**

‘_make_dfunc(…)’

Package Contents

Data

`__all__`

API

`vtkmodules.numpy_interface.`**`__all__`**

[‘algorithms’, ‘dataset_adapter’]

`vtkmodules.gtk`

pyGTK widgets for VTK.

Submodules

`vtkmodules.gtk.GtkVTKRenderWindow`

Description:

Provides a simple VTK widget for pyGtk. This embeds a vtkRenderWindow inside a GTK widget. This is based on vtkTkRenderWidget.py. The GtkVTKRenderWindowBase class provides the abstraction necessary for someone to use their own interaction behaviour. The method names are similar to those in vtkInteractorStyle.h.

The class uses the gtkgl.GtkGLArea widget (gtkglarea). This avoids a lot of problems with flicker.

There is a working example at the bottom.

Credits:

Thanks to Dave Reed for testing the code under various platforms and for his suggestion to use the GtkGLArea widget to avoid flicker related issues.

Created by Prabhu Ramachandran, March 2001.

Using GtkGLArea, March, 2002.

Bugs:

(*) There is a focus related problem. Tkinter has a focus object that handles focus events. I don’t know of an equivalent object under GTK. So, when an ‘enter_notify_event’ is received on the GtkVTKRenderWindow I grab the focus but I don’t know what to do when I get a ‘leave_notify_event’.

(*) Will not work under Win32 because it uses the XID of a window in OnRealize. Suggestions to fix this will be appreciated.

Module Contents

Classes

GtkVTKRenderWindowBase	A base class that enables one to embed a vtkRenderWindow into a pyGTK widget. This class embeds
GtkVTKRenderWindow	An example of a fully functional GtkVTKRenderWindow that is based on the vtkRenderWidget.py pro

Functions

main

API

class vtkmodules.gtk.GtkVTKRenderWindow.**GtkVTKRenderWindowBase**(**args*)

Bases: `gtkgl.GtkGLArea`

A base class that enables one to embed a vtkRenderWindow into a pyGTK widget. This class embeds the RenderWindow correctly. Provided are some empty methods that can be overloaded to provide a user defined interaction behaviour. The event handling functions have names that are somewhat similar to the ones in the vtkInteractorStyle class included with VTK.

Initialization

ConnectSignals()

GetRenderWindow()

GetRenderer()

SetDesiredUpdateRate(*rate*)

Mirrors the method with the same name in vtkRenderWindowInteractor.

GetDesiredUpdateRate()

Mirrors the method with the same name in vtkRenderWindowInteractor.

SetStillUpdateRate(*rate*)

Mirrors the method with the same name in vtkRenderWindowInteractor.

GetStillUpdateRate()

Mirrors the method with the same name in vtkRenderWindowInteractor.

Render()

OnRealize(**args*)

OnConfigure(*wid*, *event=None*)

OnExpose(**args*)

OnDestroy(*event=None*)

OnButtonDown(*wid*, *event*)

Mouse button pressed.

OnButtonUp(*wid*, *event*)

Mouse button released.

OnMouseMove(*wid*, *event*)

Mouse has moved.

OnEnter(*wid*, *event*)

Entering the vtkRenderWindow.

OnLeave(*wid*, *event*)

Leaving the vtkRenderWindow.

OnKeyPress(*wid*, *event*)

Key pressed.

OnKeyRelease(*wid*, *event*)

Key released.

class vtkmodules.gtk.GtkVTKRenderWindow.**GtkVTKRenderWindow**(**args*)

Bases: *vtkmodules.gtk.GtkVTKRenderWindow.GtkVTKRenderWindowBase*

An example of a fully functional GtkVTKRenderWindow that is based on the vtkRenderWidget.py provided with the VTK sources.

Initialization

OnButtonDown(*wid*, *event*)

OnButtonUp(*wid*, *event*)

OnMouseMove(*wid*, *event=None*)

OnEnter(*wid*, *event=None*)

OnLeave(*wid*, *event*)

OnKeyPress(*wid*, *event=None*)

GetZoomFactor()

SetZoomFactor(*zf*)

GetPicker()

Render()

UpdateRenderer(*x*, *y*)

UpdateRenderer will identify the renderer under the mouse and set up _CurrentRenderer, _CurrentCamera, and _CurrentLight.

GetCurrentRenderer()

StartMotion(*wid*, *event=None*)

EndMotion(*wid*, *event=None*)

Rotate(*x*, *y*)

Pan(*x*, *y*)

Zoom(*x*, *y*)

Reset()

Wireframe()

Surface()

PickActor(*x*, *y*)

`vtkmodules.gtk.GtkVTKRenderWindow.`**main**()

`vtkmodules.gtk.GtkGLExtVTKRenderWindow`

Description:

This provides a VTK widget for pyGtk. This embeds a vtkRenderWindow inside a GTK widget. This is based on GtkVTKRenderWindow.py.

The extensions here allow the use of gtkglext rather than gtkgl and pygtk-2 rather than pygtk-0. It requires pygtk-2.0.0 or later.

There is a working example at the bottom.

Credits:

John Hunter jdhunter@ace.bsd.uchicago.edu developed and tested this code based on VTK's GtkVTKRenderWindow.py and extended it to work with pygtk-2.0.0.

License:

VTK license.

Module Contents

Classes

GtkGLExtVTKRenderWindowBase	A base class that enables one to embed a vtkRenderWindow into a pyGTK widget. This class e
GtkGLExtVTKRenderWindow	An example of a fully functional GtkGLExtVTKRenderWindow that is based on the vtkRender

Functions

main

API

class `vtkmodules.gtk.GtkGLExtVTKRenderWindow.`**`GtkGLExtVTKRenderWindowBase`**`(*args)`

Bases: `gtk.gtkgl.DrawingArea`

A base class that enables one to embed a vtkRenderWindow into a pyGTK widget. This class embeds the RenderWindow correctly. Provided are some empty methods that can be overloaded to provide a user defined interaction behaviour. The event handling functions have names that are somewhat similar to the ones in the vtkInteractorStyle class included with VTK.

Initialization

`ConnectSignals``()`

`GetRenderWindow``()`

`GetRenderer``()`

`SetDesiredUpdateRate``(rate)`

Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetDesiredUpdateRate``()`

Mirrors the method with the same name in vtkRenderWindowInteractor.

`SetStillUpdateRate``(rate)`

Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetStillUpdateRate``()`

Mirrors the method with the same name in vtkRenderWindowInteractor.

`Render``()`

`OnRealize``(*args)`

`Created``()`

`OnConfigure``(widget, event)`

`OnExpose``(*args)`

`OnDestroy``(*args)`

`OnButtonDown``(wid, event)`

Mouse button pressed.

`OnButtonUp``(wid, event)`

Mouse button released.

`OnMouseMove``(wid, event)`

Mouse has moved.

`OnEnter``(wid, event)`

Entering the vtkRenderWindow.

`OnLeave``(wid, event)`

Leaving the vtkRenderWindow.

OnKeyPress(*wid*, *event*)

Key pressed.

OnKeyRelease(*wid*, *event*)

Key released.

class vtkmodules.gtk.GtkGLExtVTKRenderWindow.**GtkGLExtVTKRenderWindow**(**args*)

Bases: *vtkmodules.gtk.GtkGLExtVTKRenderWindow.GtkGLExtVTKRenderWindowBase*

An example of a fully functional GtkGLExtVTKRenderWindow that is based on the vtkRenderWidget.py provided with the VTK sources.

Initialization

OnButtonDown(*wid*, *event*)

OnButtonUp(*wid*, *event*)

OnMouseMove(*wid*, *event=None*)

OnEnter(*wid*, *event=None*)

OnKeyPress(*wid*, *event=None*)

GetZoomFactor()

SetZoomFactor(*zf*)

GetPicker()

Render()

UpdateRenderer(*x*, *y*)

UpdateRenderer will identify the renderer under the mouse and set up _CurrentRenderer, _CurrentCamera, and _CurrentLight.

GetCurrentRenderer()

GetCurrentCamera()

StartMotion(*wid*, *event=None*)

EndMotion(*wid*, *event=None*)

Rotate(*x*, *y*)

Pan(*x*, *y*)

Zoom(*x*, *y*)

Reset()

Wireframe()

Surface()

PickActor(*x*, *y*)

vtkmodules.gtk.GtkGLExtVTKRenderWindow.**main**()

vtkmodules.gtk.GtkGLExtVTKRenderWindowInteractor

Description:

Provides a pyGtk vtkRenderWindowInteractor widget. This embeds a vtkRenderWindow inside a GTK widget and uses the vtkGenericRenderWindowInteractor for the event handling. This is similar to GtkVTKRenderWindowInteractor.py.

The extensions here allow the use of gtkglext rather than gtkgl and pygtk-2 rather than pygtk-0. It requires pygtk-2.0.0 or later.

There is a working example at the bottom.

Credits:

John Hunter jdhunter@ace.bsd.uchicago.edu developed and tested this code based on VTK's GtkVTKRenderWindow.py and extended it to work with pygtk-2.0.0.

License:

VTK license.

Module Contents

Classes

GtkGLExtVTKRenderWindowInteractor	Embeds a vtkRenderWindow into a pyGTK widget and uses vtkGenericRenderWindow

Functions

main

API

class vtkmodules.gtk.GtkGLExtVTKRenderWindowInteractor.**GtkGLExtVTKRenderWindowInteractor**(**args*)

Bases: `gtk.gtkgl.DrawingArea`

Embeds a vtkRenderWindow into a pyGTK widget and uses vtkGenericRenderWindowInteractor for the event handling. This class embeds the RenderWindow correctly. A **getattr** hook is provided that makes the class behave like a vtkGenericRenderWindowInteractor.

Initialization

set_size_request(*w*, *h*)

ConnectSignals()

__getattr__(*attr*)

Makes the object behave like a vtkGenericRenderWindowInteractor

CreateTimer(*obj*, *event*)

DestroyTimer(*obj*, *event*)

The timer is a one shot timer so will expire automatically.

GetRenderWindow()

Render()

OnRealize(**args*)

OnConfigure(*widget*, *event*)

OnExpose(**args*)

OnDestroy(*event=None*)

_GetCtrlShift(*event*)

OnButtonDown(*wid*, *event*)

Mouse button pressed.

OnButtonUp(*wid*, *event*)

Mouse button released.

OnMouseMove(*wid*, *event*)

Mouse has moved.

OnEnter(*wid*, *event*)

Entering the vtkRenderWindow.

OnLeave(*wid*, *event*)

Leaving the vtkRenderWindow.

OnKeyPress(*wid*, *event*)

Key pressed.

OnKeyRelease(*wid*, *event*)

Key released.

Initialize()

SetPicker(*picker*)

GetPicker(*picker*)

`vtkmodules.gtk.GtkGLExtVTKRenderWindowInteractor.`**main**()

vtkmodules.gtk.GtkVTKRenderWindowInteractor

Description:

Provides a pyGtk vtkRenderWindowInteractor widget. This embeds a vtkRenderWindow inside a GTK widget and uses the vtkGenericRenderWindowInteractor for the event handling. This is based on vtkTkRenderWindow.py.

The class uses the gtkgl.GtkGLArea widget (gtkglarea). This avoids a lot of problems with flicker.

There is a working example at the bottom.

Created by Prabhu Ramachandran, April 2002.

Bugs:

(*) There is a focus related problem. Tkinter has a focus object that handles focus events. I don't know of an equivalent object under GTK. So, when an 'enter_notify_event' is received on the GtkVTKRenderWindow I grab the focus but I don't know what to do when I get a 'leave_notify_event'.

(*) Will not work under Win32 because it uses the XID of a window in OnRealize. Suggestions to fix this will be appreciated.

Module Contents

Classes

GtkVTKRenderWindowInteractor	Embeds a vtkRenderWindow into a pyGTK widget and uses vtkGenericRenderWindowInterac

Functions

main

API

class vtkmodules.gtk.GtkVTKRenderWindowInteractor.**GtkVTKRenderWindowInteractor**(**args*)

Bases: `gtkgl.GtkGLArea`

Embeds a vtkRenderWindow into a pyGTK widget and uses vtkGenericRenderWindowInteractor for the event handling. This class embeds the RenderWindow correctly. A **getattr** hook is provided that makes the class behave like a vtkGenericRenderWindowInteractor.

Initialization

set_usize(*w*, *h*)

ConnectSignals()

__getattr__(*attr*)

Makes the object behave like a vtkGenericRenderWindowInteractor

CreateTimer(*obj*, *event*)

DestroyTimer(*obj*, *event*)

The timer is a one shot timer so will expire automatically.

GetRenderWindow()

Render()

OnRealize(**args*)

OnConfigure(*wid*, *event=None*)

OnExpose(**args*)

OnDestroy(*event=None*)

_GetCtrlShift(*event*)

OnButtonDown(*wid*, *event*)

Mouse button pressed.

OnButtonUp(*wid*, *event*)

Mouse button released.

OnMouseMove(*wid*, *event*)

Mouse has moved.

OnEnter(*wid*, *event*)

Entering the vtkRenderWindow.

OnLeave(*wid*, *event*)

Leaving the vtkRenderWindow.

OnKeyPress(*wid*, *event*)

Key pressed.

OnKeyRelease(*wid*, *event*)

Key released.

Initialize()

vtkmodules.gtk.GtkVTKRenderWindowInteractor.**main**()

Package Contents

Data

__all__

API

vtkmodules.gtk.**__all__**

['GtkVTKRenderWindow', 'GtkVTKRenderWindowInteractor', 'GtkGLExtVTKRenderWindow', 'GtkGLExtVTKRender...

vtkmodules.test

Modules used for testing VTK-Python wrappers and writing tests for VTK using Python.

Submodules

`vtkmodules.test.BlackBox`

Module Contents

Classes

Tester

API

class vtkmodules.test.BlackBox.**Tester**(*debug=0*)

Initialization

setDebug(*val*)

Sets debug value of the vtkMethodParser. 1 is verbose and 0 is not. 0 is default.

testParse(*obj*)

Testing if the object is parseable.

testGetSet(*obj*, *excluded_methods=[]*)

Testing Get/Set methods.

testBoolean(*obj*, *excluded_methods=[]*)

Testing boolean (On/Off) methods.

test(*obj*)

Test the given vtk object.

`vtkmodules.test.Testing`

This module attempts to make it easy to create VTK-Python unittests. The module uses unittest for the test interface. For more documentation on what unittests are and how to use them, please read these:

http://www.python.org/doc/current/lib/module-unittest.html

http://www.diveintopython.org/roman_divein.html

This VTK-Python test module supports image based tests with multiple images per test suite and multiple images per individual test as well. It also prints information appropriate for CDash (http://open.kitware.com/).

This module defines several useful classes and functions to make writing tests easy. The most important of these are:

class vtkTest: Subclass this for your tests. It also has a few useful internal functions that can be used to do some simple blackbox testing.

compareImage(renwin, img_fname, threshold=0.15): Compares renwin with image and generates image if it does not exist. The threshold determines how closely the images must match. The function also handles multiple images and finds the best matching image.

compareImageWithSavedImage(src_img, img_fname, threshold=0.15): Compares given source image (in the form of a vtkImageData) with saved image and generates the image if it does not exist. The threshold determines how closely the images must match. The function also handles multiple images and finds the best matching image.

getAbsImagePath(img_basename): Returns the full path to the image given the basic image name.

main(cases): Does the testing given a list of tuples containing test classes and the starting string of the functions used for testing.

interact(): Interacts with the user if necessary. The behavior of this is rather trivial and works best when using Tkinter. It does not do anything by default and stops to interact with the user when given the appropriate command line arguments.

isInteractive(): If interact() is not good enough, use this to find if the mode is interactive or not and do whatever is necessary to generate an interactive view.

Examples:

The best way to learn on how to use this module is to look at a few examples. The end of this file contains a trivial example. Please also look at the following examples:

```
Rendering/Testing/Python/TestTkRenderWidget.py,
Rendering/Testing/Python/TestTkRenderWindowInteractor.py
```

Created: September, 2002

Prabhu Ramachandran prabhu@aero.iitb.ac.in

Module Contents

Classes

vtkTest	A simple default VTK test class that defines a few useful blackbox tests that can be readily used. Derive your test cases fro

Functions

skip	Cause the test to be skipped due to insufficient requirements.
interact	Interacts with the user if necessary.
isInteractive	Returns if the currently chosen mode is interactive or not based on command line options.
getAbsImagePath	Returns the full path to the image given the basic image name.
_getTempImagePath	
compareImageWithSavedImage	Compares a source image (src_img, which is a vtkImageData) with the saved image file whos
compareImage	Compares renwin's (a vtkRenderWindow) contents with the image file whose name is given i
_printCDashImageError	Prints the XML data necessary for CDash.
_printCDashImageNotFoundError	Prints the XML data necessary for Dart when the baseline image is not found.
_printCDashImageSuccess	Prints XML data for Dart when image test succeeded.
_handleFailedImage	Writes all the necessary images when an image comparison failed.
main	Pass a list of tuples containing test classes and the starting string of the functions used for tes
test	Pass a list of tuples containing test classes and the functions used for testing.
usage	
parseCmdLine	
processCmdLine	

Data

VTK_DATA_ROOT
VTK_DATA_PATHS
VTK_BASELINE_ROOT
VTK_TEMP_DIR
VTK_BASELINE_PATHS
_VERBOSE
_INTERACT
_NO_IMAGE

API

`vtkmodules.test.Testing.`**`VTK_DATA_ROOT`**` = <Multiline-String>`

`vtkmodules.test.Testing.`**`VTK_DATA_PATHS`**

[]

`vtkmodules.test.Testing.`**`VTK_BASELINE_ROOT`**` = <Multiline-String>`

`vtkmodules.test.Testing.`**`VTK_TEMP_DIR`**` = <Multiline-String>`

`vtkmodules.test.Testing.`**`VTK_BASELINE_PATHS`**

[]

`vtkmodules.test.Testing.`**`_VERBOSE`**

0

`vtkmodules.test.Testing.`**`_INTERACT`**

0

`vtkmodules.test.Testing.`**`_NO_IMAGE`**

0

`vtkmodules.test.Testing.`**`skip`**`()`

Cause the test to be skipped due to insufficient requirements.

`class` `vtkmodules.test.Testing.`**`vtkTest`**`(`*methodName='runTest'*`)`

Bases: `unittest.TestCase`

A simple default VTK test class that defines a few useful blackbox tests that can be readily used. Derive your test cases from this class and use the following if you'd like to.

Note: Unittest instantiates this class (or your subclass) each time it tests a method. So if you do not want that to happen when generating VTK pipelines you should create the pipeline in the class definition as done below for _blackbox.

Initialization

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

`_blackbox`

‘Tester(. . .)’

`dl`

‘vtkDebugLeaks(. . .)’

`_testParse`(*obj*)

Does a blackbox test by attempting to parse the class for its various methods using vtkMethodParser. This is a useful test because it gets all the methods of the vtkObject, parses them and sorts them into different classes of objects.

`_testGetSet`(*obj*, *excluded_methods=[]*)

Checks the Get/Set method pairs by setting the value using the current state and making sure that it equals the value it was originally. This effectively calls _testParse internally.

`_testBoolean`(*obj*, *excluded_methods=[]*)

Checks the Boolean methods by setting the value on and off and making sure that the GetMethod returns the set value. This effectively calls _testParse internally.

`pathToData`(*filename*)

Given a filename with no path (i.e., no leading directories prepended), return the full path to a file as specified on the command line with a ‘-D’ option.

As an example, if a test is run with “-D /path/to/grid.vtu” then calling

```
self.pathToData('grid.vtu')
```

in your test will return “/path/to/grid.vtu”. This is useful in combination with ExternalData, where data may be staged by CTest to a user-configured directory at build time.

In order for this method to work, you must specify the JUST_VALID option for your test in CMake.

`pathToValidatedOutput`(*filename*)

Given a filename with no path (i.e., no leading directories prepended), return the full path to a file as specified on the command line with a ‘-V’ option.

As an example, if a test is run with “-V /path/to/validImage.png” then calling

```
self.pathToData('validImage.png')
```

in your test will return “/path/to/validImage.png”. This is useful in combination with ExternalData, where data may be staged by CTest to a user-configured directory at build time.

In order for this method to work, you must specify the JUST_VALID option for your test in CMake.

`prepareTestImage`(*interactor*, ***kwargs*)

`assertImageMatch`(*renwin*, *baseline*, ***kwargs*)

Throw an error if a rendering in the render window does not match the baseline image.

This method accepts a threshold keyword argument (with a default of 0.15) that specifies how different a baseline may be before causing a failure.

`vtkmodules.test.Testing.`**`interact`**`()`

Interacts with the user if necessary.

`vtkmodules.test.Testing.`**`isInteractive`**`()`

Returns if the currently chosen mode is interactive or not based on command line options.

`vtkmodules.test.Testing.`**`getAbsImagePath`**`(`*img_basename*`)`

Returns the full path to the image given the basic image name.

`vtkmodules.test.Testing.`**`_getTempImagePath`**`(`*img_fname*`)`

`vtkmodules.test.Testing.`**`compareImageWithSavedImage`**`(`*src_img*, *img_fname*, *threshold=0.15*`)`

Compares a source image (src_img, which is a vtkImageData) with the saved image file whose name is given in the second argument. If the image file does not exist the image is generated and stored. If not the source image is compared to that of the figure. This function also handles multiple images and finds the best matching image.

`vtkmodules.test.Testing.`**`compareImage`**`(`*renwin*, *img_fname*, *threshold=0.15*`)`

Compares renwin's (a vtkRenderWindow) contents with the image file whose name is given in the second argument. If the image file does not exist the image is generated and stored. If not the image in the render window is compared to that of the figure. This function also handles multiple images and finds the best matching image.

`vtkmodules.test.Testing.`**`_printCDashImageError`**`(`*img_err*, *err_index*, *img_base*`)`

Prints the XML data necessary for CDash.

`vtkmodules.test.Testing.`**`_printCDashImageNotFoundError`**`(`*img_fname*`)`

Prints the XML data necessary for Dart when the baseline image is not found.

`vtkmodules.test.Testing.`**`_printCDashImageSuccess`**`(`*img_err*, *err_index*`)`

Prints XML data for Dart when image test succeeded.

`vtkmodules.test.Testing.`**`_handleFailedImage`**`(`*idiff*, *pngr*, *img_fname*`)`

Writes all the necessary images when an image comparison failed.

`vtkmodules.test.Testing.`**`main`**`(`*cases*`)`

Pass a list of tuples containing test classes and the starting string of the functions used for testing.

Example:

main ([(vtkTestClass, 'test'), (vtkTestClass1, 'test')])

`vtkmodules.test.Testing.`**`test`**`(`*cases*`)`

Pass a list of tuples containing test classes and the functions used for testing.

It returns a unittest._TextTestResult object.

Example:

test = test_suite([(vtkTestClass, 'test'), (vtkTestClass1, 'test')])

`vtkmodules.test.Testing.`**`usage`**`()`

`vtkmodules.test.Testing.`**`parseCmdLine`**`()`

`vtkmodules.test.Testing.`**`processCmdLine`**`()`

vtkmodules.test.ErrorObserver

Module Contents

Classes

vtkErrorObserver	

API

class vtkmodules.test.ErrorObserver.**vtkErrorObserver**

Bases: object

Initialization

__call__(*caller*, *event*, *data*)

_check(*seen*, *actual*, *expect*, *what*)

check_error(*expect*)

check_warning(*expect*)

reset()

property saw_error

property error_message

property saw_warning

property warning_message

vtkmodules.test.rtImageTest

Module Contents

Functions

_GetController	
main	Run a regression test, and compare the contents of the window against against a valid image. This will use argum

API

`vtkmodules.test.rtImageTest._GetController()`

`vtkmodules.test.rtImageTest.main(`*test_script*`)`

Run a regression test, and compare the contents of the window against against a valid image. This will use arguments from sys.argv to set the testing options via the vtkTesting class, run the test script, and then call vtkTesting.RegressionTest() to validate the image. The return value will the one provided by vtkTesting.RegressionTest().

Package Contents

Data

`__all__`

API

`vtkmodules.test.__all__`

[‘Testing’, ‘BlackBox’, ‘ErrorObserver’, ‘rtImageTest’]

`vtkmodules.tk`

Tkinter widgets for VTK.

Submodules

`vtkmodules.tk.vtkTkRenderWidget`

A simple vtkTkRenderWidget for tkinter.

Created by David Gobbi, April 1999

May ??, 1999 - Modifications performed by Heather Drury, to rewrite _pan to match method in TkInteractor.tcl May 11, 1999 - Major rewrite by David Gobbi to make the interactor bindings identical to the TkInteractor.tcl bindings. July 14, 1999 - Added modification by Ken Martin for VTK 2.4, to use vtk widgets instead of Togl. Aug 29, 1999 - Renamed file to vtkRenderWidget.py Nov 14, 1999 - Added support for keyword ‘rw’ Mar 23, 2000 - Extensive but backwards compatible changes, improved documentation

A few important notes:

This class is meant to be used as a base-class widget for doing VTK rendering in Python.

In VTK (and C++) there is a very important distinction between public ivars (attributes in pythonspeak), protected ivars, and private ivars. When you write a python class that you want to ‘look and feel’ like a VTK class, you should follow these rules.

1) Attributes should never be public. Attributes should always be either protected (prefixed with a single underscore) or private (prefixed with a double underscore). You can provide access to attributes through public Set/Get methods (same as VTK).

2) Use a single underscore to denote a protected attribute, e.g. self._RenderWindow is protected (can be accessed from this class or a derived class).
3) Use a double underscore to denote a private attribute, e.g. self.__InExpose cannot be accessed outside of this class.

All attributes should be 'declared' in the **init**() function i.e. set to some initial value. Don't forget that 'None' means 'NULL' - the python/vtk wrappers guarantee their equivalence.

Module Contents

Classes

vtkTkRenderWidget	A vtkTkRenderWidget for Python.

Functions

vtkRenderWidgetConeExample	Like it says, just a simple example

API

class vtkmodules.tk.vtkTkRenderWidget.**vtkTkRenderWidget**(*master*, *cnf={}*, ***kw*)

Bases: `tkinter.Widget`

A vtkTkRenderWidget for Python.

Use GetRenderWindow() to get the vtkRenderWindow.

Create with the keyword stereo=1 in order to generate a stereo-capable window.

Create with the keyword focus_on_enter=1 to enable focus-follows-mouse. The default is for a click-to-focus mode.

Initialization

Constructor.

Keyword arguments:

rw – Use passed render window instead of creating a new one.

stereo – If True, generate a stereo-capable window. Defaults to False.

focus_on_enter – If True, use a focus-follows-mouse mode. Defaults to False where the widget will use a click-to-focus mode.

__getattr__(*attr*)

BindTkRenderWidget()

Bind some default actions.

GetZoomFactor()

`SetDesiredUpdateRate`(*rate*)
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetDesiredUpdateRate`()
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`SetStillUpdateRate`(*rate*)
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetStillUpdateRate`()
: Mirrors the method with the same name in vtkRenderWindowInteractor.

`GetRenderWindow`()

`GetPicker`()

`Expose`()

`Render`()

`UpdateRenderer`(*x*, *y*)
: UpdateRenderer will identify the renderer under the mouse and set up _CurrentRenderer, _CurrentCamera, and _CurrentLight.

`GetCurrentRenderer`()

`Enter`(*x*, *y*)

`Leave`(*x*, *y*)

`StartMotion`(*x*, *y*)

`EndMotion`(*x*, *y*)

`Rotate`(*x*, *y*)

`Pan`(*x*, *y*)

`Zoom`(*x*, *y*)

`Reset`(*x*, *y*)

`Wireframe`()

`Surface`()

`PickActor`(*x*, *y*)

`vtkmodules.tk.vtkTkRenderWidget.vtkRenderWidgetConeExample`()
: Like it says, just a simple example

vtkmodules.tk.vtkTkRenderWindowInteractor

A fully functional VTK widget for tkinter that uses vtkGenericRenderWindowInteractor. The widget is called vtkTkRenderWindowInteractor. The initialization part of this code is similar to that of the vtkTkRenderWidget.

Created by Prabhu Ramachandran, April 2002

Module Contents

Classes

vtkTkRenderWindowInteractor	A vtkTkRenderWidndowInteractor for Python.

Functions

vtkRenderWindowInteractorConeExample	Like it says, just a simple example

API

`class vtkmodules.tk.vtkTkRenderWindowInteractor.`**`vtkTkRenderWindowInteractor`**`(`*master*, *cnf={}*, ***kw*`)`

Bases: `tkinter.Widget`

A vtkTkRenderWidndowInteractor for Python.

Use GetRenderWindow() to get the vtkRenderWindow.

Create with the keyword stereo=1 in order to generate a stereo-capable window.

Create with the keyword focus_on_enter=1 to enable focus-follows-mouse. The default is for a click-to-focus mode.

getattr is used to make the widget also behave like a vtkGenericRenderWindowInteractor.

Initialization

Constructor.

Keyword arguments:

rw – Use passed render window instead of creating a new one.

stereo – If True, generate a stereo-capable window. Defaults to False.

focus_on_enter – If True, use a focus-follows-mouse mode. Defaults to False where the widget will use a click-to-focus mode.

`__getattr__``(`*attr*`)`

`BindEvents``()`

Bind all the events.

`CreateTimer`(*obj*, *evt*)

`DestroyTimer`(*obj*, *event*)

The timer is a one shot timer so will expire automatically.

`_GrabFocus`(*enter=0*)

`MouseMoveEvent`(*event*, *ctrl*, *shift*)

`LeftButtonPressEvent`(*event*, *ctrl*, *shift*)

`LeftButtonReleaseEvent`(*event*, *ctrl*, *shift*)

`MiddleButtonPressEvent`(*event*, *ctrl*, *shift*)

`MiddleButtonReleaseEvent`(*event*, *ctrl*, *shift*)

`RightButtonPressEvent`(*event*, *ctrl*, *shift*)

`RightButtonReleaseEvent`(*event*, *ctrl*, *shift*)

`MouseWheelEvent`(*event*, *ctrl*, *shift*)

`MouseWheelForwardEvent`(*event*, *ctrl*, *shift*)

`MouseWheelBackwardEvent`(*event*, *ctrl*, *shift*)

`KeyPressEvent`(*event*, *ctrl*, *shift*)

`KeyReleaseEvent`(*event*, *ctrl*, *shift*)

`ConfigureEvent`(*event*)

`EnterEvent`(*event*, *ctrl*, *shift*)

`LeaveEvent`(*event*, *ctrl*, *shift*)

`ExposeEvent()`

`GetRenderWindow()`

`Render()`

`vtkmodules.tk.vtkTkRenderWindowInteractor.vtkRenderWindowInteractorConeExample()`

Like it says, just a simple example

`vtkmodules.tk.vtkTkPhotoImage`

A subclass of tkinter.PhotoImage that connects a vtkImageData to a photo widget.

Created by Daniel Blezek, August 2002

Module Contents

Classes

vtkTkPhotoImage	A subclass of PhotoImage with helper functions for displaying vtkImageData

API

class `vtkmodules.tk.vtkTkPhotoImage.`**`vtkTkPhotoImage`**(***kw*)

Bases: `tkinter.PhotoImage`

A subclass of PhotoImage with helper functions for displaying vtkImageData

Initialization

Create an image with NAME.

Valid resource names: data, format, file, gamma, height, palette, width.

`PutImageSlice`(*image*, *z*, *orientation='transverse'*, *window=256*, *level=128*)

`vtkmodules.tk.vtkLoadPythonTkWidgets`

Module Contents

Functions

vtkLoadPythonTkWidgets	vtkLoadPythonTkWidgets(interp) – load vtk-tk widget extensions

API

`vtkmodules.tk.vtkLoadPythonTkWidgets.`**`vtkLoadPythonTkWidgets`**(*interp*)

vtkLoadPythonTkWidgets(interp) – load vtk-tk widget extensions

This is a mess of mixed python and tcl code that searches for the shared object file that contains the python-vtk-tk widgets. Both the python path and the tcl path are searched.

vtkmodules.tk.vtkTkImageViewerWidget

A vtkTkImageViewerWidget for python, which is based on the vtkTkImageWindowWidget.

Specify double=1 to get a double-buffered window.

Created by David Gobbi, Nov 1999

Module Contents

Classes

vtkTkImageViewerWidget	A vtkTkImageViewerWidget for Python.

API

class `vtkmodules.tk.vtkTkImageViewerWidget.`**`vtkTkImageViewerWidget`**(*master*, *cnf={}*, ***kw*)

Bases: `tkinter.Widget`

A vtkTkImageViewerWidget for Python.

Use GetImageViewer() to get the vtkImageViewer.

Create with the keyword double=1 in order to generate a double-buffered viewer.

Create with the keyword focus_on_enter=1 to enable focus-follows-mouse. The default is for a click-to-focus mode.

Initialization

Constructor.

Keyword arguments:

iv – Use passed image viewer instead of creating a new one.

double – If True, generate a double-buffered viewer. Defaults to False.

focus_on_enter – If True, use a focus-follows-mouse mode. Defaults to False where the widget will use a click-to-focus mode.

`__getattr__`(*attr*)

`GetImageViewer`()

`Render`()

`BindTkImageViewer`()

`_GrabFocus`()

`EnterTkViewer`()

`LeaveTkViewer`()

ExposeTkImageViewer()

StartWindowLevelInteraction(*x*, *y*)

EndWindowLevelInteraction()

UpdateWindowLevelInteraction(*x*, *y*)

ResetTkImageViewer()

StartQueryInteraction(*x*, *y*)

EndQueryInteraction()

UpdateQueryInteraction(*x*, *y*)

Package Contents

Data

__all__

API

vtkmodules.tk.**__all__**

[‘vtkTkRenderWidget’, ‘vtkTkImageViewerWidget’, ‘vtkTkRenderWindowInteractor’, ‘vtkTkPhotoImage’]

Submodules

vtkmodules.generate_pyi

This program will generate .pyi files for all the VTK modules in the “vtkmodules” package (or whichever package you specify). These files are used for type checking and autocompletion in some Python IDEs.

The VTK modules must be in Python’s path when you run this script. Options are as follows:

-p PACKAGE The package to generate .pyi files for [vtkmodules] -o OUTPUT The output directory [default is the package directory] -e EXT The file suffix [.pyi] -i IMPORTER The static module importer (for static builds only) -h HELP

With no arguments, the script runs with the defaults (the .pyi files are put inside the existing vtkmodules package). This is equivalent to the following:

```
python -m vtkmodules.generate_pyi -p vtkmodules
```

To put the pyi files somewhere else, perhaps with a different suffix:

```
python -m vtkmodules.generate_pyi -o /path/to/vtkmodules -e .pyi
```

To generate pyi files for just one or two modules:

```
python -m vtkmodules.generate_pyi -p vtkmodules vtkCommonCore vtkCommonDataModel
```

To generate pyi files for your own modules in your own package:

```
python -m vtkmodules.generate_pyi -p mypackage mymodule [mymodule2 ...]
```

Module Contents

Classes

Graph	A graph for topological sorting.
Node	A node for the graph.

Functions

isvtkmethod	Check for VTK's custom method descriptor
isnamespace	Check for namespaces within a module
isenum	Check for enums (currently derived from int)
typename	Generate a typename that can be used for annotation.
typename_forward	Generate a typename, or if necessary, a forward reference.
build_graph	Build a graph from a module's dictionary.
sorted_graph_helper	Helper for topological sorting.
sorted_graph	Sort a graph and return the sorted items.
topologically_sorted_items	Return the items from a module's dictionary, topologically sorted.
parse_error	Print a parse error, syntax or otherwise.
annotation_text	Return the new text to be used for an annotation.
fix_annotations	Fix the annotations in a method definition. The signature must be a single-line function def, no d
push_signature	Process a method signature and add it to the list.
get_signatures	Return a list of method signatures found in the docstring.
get_constructors	Get constructors from the class documentation.
handle_static	If method has no "self", add @static decorator.
add_indent	Add the given indent before every line in the string.
namespace_pyi	Fake a namespace by creating a dummy class.
class_pyi	Generate all the method stubs for a class.
module_pyi	Generate the contents of a .pyi file for a VTK module.
main	

Data

types
ismethod
isclass
vtkmethod
template

continues on next page

Table 67 – continued from previous page

string
identifier
indent
has_self
keychar

API

`vtkmodules.generate_pyi.`**`types`**

‘set(…)’

`vtkmodules.generate_pyi.`**`ismethod`**

None

`vtkmodules.generate_pyi.`**`isclass`**

None

`vtkmodules.generate_pyi.`**`vtkmethod`**

‘type(…)’

`vtkmodules.generate_pyi.`**`template`**

‘type(…)’

`vtkmodules.generate_pyi.`**`isvtkmethod`**(*m*)

Check for VTK’s custom method descriptor

`vtkmodules.generate_pyi.`**`isnamespace`**(*m*)

Check for namespaces within a module

`vtkmodules.generate_pyi.`**`isenum`**(*m*)

Check for enums (currently derived from int)

`vtkmodules.generate_pyi.`**`typename`**(*o*)

Generate a typename that can be used for annotation.

`vtkmodules.generate_pyi.`**`typename_forward`**(*o*)

Generate a typename, or if necessary, a forward reference.

`class` `vtkmodules.generate_pyi.`**`Graph`**

A graph for topological sorting.

Initialization

`__getitem__`(*name*)

`__setitem__`(*name*, *node*)

`class` `vtkmodules.generate_pyi.`**`Node`**(*o*, *d*)

A node for the graph.

Initialization

`vtkmodules.generate_pyi.`**`build_graph`**`(`*d*`)`

Build a graph from a module's dictionary.

`vtkmodules.generate_pyi.`**`sorted_graph_helper`**`(`*graph*, *m*, *visited*, *items*`)`

Helper for topological sorting.

`vtkmodules.generate_pyi.`**`sorted_graph`**`(`*graph*`)`

Sort a graph and return the sorted items.

`vtkmodules.generate_pyi.`**`topologically_sorted_items`**`(`*d*`)`

Return the items from a module's dictionary, topologically sorted.

`vtkmodules.generate_pyi.`**`string`**

'compile(…)'

`vtkmodules.generate_pyi.`**`identifier`**

'compile(…)'

`vtkmodules.generate_pyi.`**`indent`**

'compile(…)'

`vtkmodules.generate_pyi.`**`has_self`**

'compile(…)'

`vtkmodules.generate_pyi.`**`keychar`**

'compile(…)'

`vtkmodules.generate_pyi.`**`parse_error`**`(`*message*, *text*, *begin*, *pos*`)`

Print a parse error, syntax or otherwise.

`vtkmodules.generate_pyi.`**`annotation_text`**`(`*a*, *text*, *is_return*`)`

Return the new text to be used for an annotation.

`vtkmodules.generate_pyi.`**`fix_annotations`**`(`*signature*`)`

Fix the annotations in a method definition. The signature must be a single-line function def, no decorators.

`vtkmodules.generate_pyi.`**`push_signature`**`(`*o*, *l*, *signature*`)`

Process a method signature and add it to the list.

`vtkmodules.generate_pyi.`**`get_signatures`**`(`*o*`)`

Return a list of method signatures found in the docstring.

`vtkmodules.generate_pyi.`**`get_constructors`**`(`*c*`)`

Get constructors from the class documentation.

`vtkmodules.generate_pyi.`**`handle_static`**`(`*o*, *signature*`)`

If method has no "self", add @static decorator.

`vtkmodules.generate_pyi.`**`add_indent`**`(`*s*, *indent*`)`

Add the given indent before every line in the string.

`vtkmodules.generate_pyi.`**`namespace_pyi`**`(`*c*, *mod*`)`

Fake a namespace by creating a dummy class.

`vtkmodules.generate_pyi.`**`class_pyi`**`(`*c*`)`

Generate all the method stubs for a class.

`vtkmodules.generate_pyi.`**`module_pyi`**`(`*mod*, *output*`)`

Generate the contents of a .pyi file for a VTK module.

`vtkmodules.generate_pyi.`**`main`**`(`*argv=sys.argv*`)`

Package Contents

Functions

_windows_dll_path
_load_vtkmodules_static

Data

__all__
__version__

API

`vtkmodules.`**`_windows_dll_path`**`()`

`vtkmodules.`**`_load_vtkmodules_static`**`()`

`vtkmodules.`**`__all__`**

['vtkCommonCore', 'vtkWebCore', 'vtkCommonMath', 'vtkCommonTransforms', 'vtkCommonDataModel', 'vtkCo...

`vtkmodules.`**`__version__`**

'9.2.6'

8.2.2 Doxygen-style documentation

VTK is implemented in C++ and it is made available in Python via its Python Wrappers. Although, the VTK doxygen documentation is derived from the C++ API, the corresponding Python API uses the same classes and methods. There are however some conventions in place for how wrapping is constructed. To quickly inspect the available methods of a class you can use the `help` method:

```
>> import vtk
help(vtk.vtkSphereSource)

Help on vtkSphereSource object:

class vtkSphereSource(vtkmodules.vtkCommonExecutionModel.vtkPolyDataAlgorithm)
 |  vtkSphereSource - create a polygonal sphere centered at the origin
 |
 |  Superclass: vtkPolyDataAlgorithm
 |
```

(continues on next page)

(continued from previous page)

```
 |  vtkSphereSource creates a sphere (represented by polygons) of
 |  specified radius centered at the origin. The resolution (polygonal
 |  discretization) in both the latitude (phi) and longitude (theta)
 |  directions can be specified. It also is possible to create partial
 |  spheres by specifying maximum phi and theta angles. By default, the
 |  surface tessellation of the sphere uses triangles; however you can
 |  set LatLongTessellation to produce a tessellation using
 |  quadrilaterals.
 |
 |  @warning
 |  Resolution means the number of latitude or longitude lines for a
 |  complete sphere. If you create partial spheres the number of
 |  latitude/longitude lines may be off by one.
 |
 |  Method resolution order:
 |      vtkSphereSource
 |      vtkmodules.vtkCommonExecutionModel.vtkPolyDataAlgorithm
 |      vtkmodules.vtkCommonExecutionModel.vtkAlgorithm
 |      vtkmodules.vtkCommonCore.vtkObject
 |      vtkmodules.vtkCommonCore.vtkObjectBase
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  GenerateNormalsOff(...)
 |      GenerateNormalsOff(self) -> None
 |      C++: virtual void GenerateNormalsOff()
 |
 |  GenerateNormalsOn(...)
 |      GenerateNormalsOn(self) -> None
 |      C++: virtual void GenerateNormalsOn()
 |
 |  GetCenter(...)
 |      GetCenter(self) -> (float, float, float)
 |      C++: virtual double *GetCenter()
...
```

For a more in-depth description of the Python Wrappers see the dedicated *section*.

8.3 CMake

For a thorough description of the module system see the *Module System* section.

The CMake API can be separated into several categories:

- **Module APIs** *module*

  This category includes functions to find and build VTK modules. A module is a set of related functionality. These are then compiled together into libraries at the "kit" level. Each module may be enabled or disabled individually and its dependencies will be built as needed.

  All functions strictly check their arguments. Any unrecognized or invalid values for a function cause errors to be raised.

- **Internal APIs** *module-internal*

  The VTK module system provides some API functions for use by other code which consumes VTK modules (primarily language wrappers). This file documents these APIs. They may start with *_vtk_module*, but they are intended for use in cases of language wrappers or dealing with trickier third party packages.

- **Implementation APIs** *module-impl*

  These functions are purely internal implementation details. No guarantees are made for them and they may change at any time (including wrapping code calls). Note that these functions are usually very lax in their argument parsing.

- **Python Wrapping APIs** *module-wrapping-python*

  APIs for wrapping modules for Python.

- **Java Wrapping APIs** *module-wrapping-java*

  APIs for wrapping modules for Java.

- **Support APIs** *module-support*

  Miscellaneous utilities.

8.3.1 Module System

VTK 9.0 introduces a new build system compared to previous versions. This version uses CMake's built-in functionality for behaviors that were performed manually in the previous iteration of the build system.

Terminology

- **module**: A unit of API provided by a project. This is the core of the system and there are lots of features available through this mechanism that are not provided by CMake's library or other usage requirements.
- **group**: A configure-time collection of modules. These may be used to control whether member modules will be built or not with a single flag.
- **kit**: A collection of modules for which all the compiled code is placed in a single library.
- **property**: An attribute of a module. Only of real interest to developers of the module system and its extensions.
- **autoinit**: A mechanism for triggering registration to global registries based on the complete set of linked-to libraries.
- **third party**: A module representing an external dependency.
- **enable status**: A 4-way state to allow for "weak" and "strong" selection or deselection of a module or group for building.

Principles

The module system was designed with a number of principles in mind. These should be followed as much as possible when developing extensions as well.

- The minimum CMake version required by the module system should be as low as possible to get the required features. For example, if a new feature is available in 3.15 that improves core module functionality, that'd be a reasonable reason to require it. But a bugfix in 3.10 that can be worked around should not bump the minimum version. Currently CMake 3.8 is expected to work, though various features (such as kits) are only available with newer CMake versions.

- Build tree looks like the install tree. The layout of the build tree is set up to mirror the layout of the install tree. This allows more code content to be shared between build and install time.
- Convention over configuration. CMake conventions should be followed. Of note, projects are assumed to be "well-behaved" including, but not limited to:
  - use of `BUILD_SHARED_LIBS` to control shared vs. static library compilation;
  - use of `GNUInstallDirs`; and
  - sensible defaults based on things like `CMAKE_PROJECT_NAME` as set by the `project()` function.
- Configuration through API. Where configuration is provided, instead of using global state or "magic" variables, configuration should be provided through parameters to the API functions provided. Concessions are made for rarely-used functionality or where the API would be complicated to plumb through the required information. These variables (which are typically parameterized) are documented at the end of this document. Such variables should be named so that it is unambiguous that they are for the module system.
- Don't pollute the environment. Variables should be cleaned up at the end of macros and functions should use variable names that don't conflict with the caller environment (usually by prefixing with `_function_name_` or the like).
- Relocatable installs. Install trees should not bake-in paths from the build tree or build machine (at least by default). This makes it easier to create packages from install trees instead of having to run a post-processing step over it before it may be used for distributable packages.

Build process

Building modules involves two phases. The first phase is called "scanning" and involves collecting all the information necessary for the second phase, "building". Scanning uses the `vtk_module_scan()` function to search the *vtk.module* files for metadata, gathers the set of modules to build and returns them to the caller. That list of modules is eventually passed to `vtk_module_build()` which sorts the modules for their build order and then builds each module in turn. This separation allows for scanning and building modules in different groups. For example, the main set of modules may be scanned to determine which of some internal set of modules are required by those which is then scanned separately with different options.

Scanning should occur from the leaf-most module set and work its way inward to the lower levels. This is done so that modules in the lower level that are required higher up can be enabled gracefully. Builds should start at the lower level and move up the tree so that targets required by the higher groups exist when they are built.

Modules

Modules are described by *vtk.module* files. These files are "scanned" using the `vtk_module_scan()` function. They provide all the information necessary for the module system to:

- provide cache variables for selecting the module (e.g., `VTK_MODULE_ENABLE_ModuleName`);
- construct the dependency tree to automatically enable or disable modules based on whether it is built or not;
- provide module-level metadata (such as exclusion from any wrapping and marking modules as third party)

The *vtk.module* files are read and "parsed", but not executed directly. This ensures that the module files do not contain any procedural CMake code. The files may contain comments starting with # like CMake code. They may either be passed manually to `vtk_module_scan()` or discovered by using the `vtk_module_find_modules()` convenience function.

The most important (and only required) parameter is the `NAME` of a module. This is used as the target name in CMake and is how the module's target should be referred to in all CMake code, inside the build and from the `find_package` which provides the module. To change the name of the compiled artifact (library or executable), the `LIBRARY_NAME` argument may be used.

It is highly recommended to provide a `DESCRIPTION` for the module. This is added to the documentation for the cache variable so that the user has more than just the module name to know what the module's purpose is.

Modules may also belong to groups which are created implicitly by adding modules to the same-named group. Groups are listed under the `GROUPS` argument and are checked in order for a non-default setting to use.

A module may be hidden by using the `CONDITION` argument. The values passed to this field is added into a CMake `if` statement and checked for validity (all quoting is passed along verbatim). If the condition evaluates to `FALSE`, the module is treated as if it did not exist at all.

Module metadata

A number of pieces of metadata are considered important enough to indicate them at the module level. These are used for managing slightly different workflows for modules which have these properties.

- `EXCLUDE_WRAP`: This marks the module with a flag that all language wrapping facilities should use to know that this module is not meant for wrapping in any language. Usually this is for modules containing user interface classes, low-level functionality, or logic that is language specific.
- `IMPLEMENTABLE` and `IMPLEMENTS`: These are used by the *autoinit* functionality to trigger the static factory registration calls. A module which is listed under an `IMPLEMENTS` list must be marked as `IMPLEMENTABLE` itself.
- `THIRD_PARTY`: Indicates that the module represents a third party dependency. It may be internal or external to the source tree, but may be used as an additional configuration point if necessary. These modules are implicitly `EXCLUDE_WRAP`, not `IMPLEMENTABLE` and do not `IMPLEMENTS` any module.

Enabling modules for build

Modules are enabled in a number of ways. These ways allow for project control and user control of which modules should be built or not. There are 4 states for controlling a module's *enable status* as well as a `DEFAULT` setting which is used to allow for other mechanisms to select the enable status:

- `YES`: The module must be built.
- `NO`: The module must not be built. If a `YES` module has a `NO` module in its dependency tree, an error is raised.
- `WANT`: The module should be built. It will not be built, however, if it depends on a `NO` module.
- `DONT_WANT`: The module doesn't need to be built. It will be built if a `YES` or `WANT` module depends on it.
- `DEFAULT`: Look at other metadata to determine the status.

The first check for modules are via the `REQUEST_MODULES` and `REJECT_MODULES` arguments to the `vtk_module_scan` function. Modules passed to `REQUEST_MODULES` are treated as if they use `YES` and `REJECT_MODULES` as if they use `NO`. A module may not be passed to both arguments. Modules selected in this way do not have CMake cache variables exposed for them (since it is assumed they are selected via some other mechanism outside the module system).

The next selector is the `VTK_MODULE_ENABLE_` variable for the module. This is added to the cache and defaults to `DEFAULT`. Assuming `HIDE_MODULES_FROM_CACHE` is not set to `ON`, this setting is exposed in the cache and allows users to change the status of modules not handled via the `REQUEST_MODULES` and `REJECT_MODULES` mechanism.

If a module is still selected as `DEFAULT`, the list of `GROUPS` it is a member of is used. In order, each group is looked at for a non-`DEFAULT` value. If so, its value is used for the module. Groups also default to using `DEFAULT` for their setting, but a project may set the `_vtk_module_group_default_${group}` variable to change this default value.

After all of the above logic, if a module is still marked as `DEFAULT`, the `WANT_BY_DEFAULT` argument to *`vtk_module_scan()`* is used to determine whether it is treated as a `WANT` or `DONT_WANT` request.

Now that all modules have a non-`DEFAULT` enable setting, the set of modules and kits that are available may be determined by traversing the dependency tree of the modules.

Dependencies

Modules have three types of dependencies:

- `DEPENDS`: These are dependencies which must be available and are transitively provided to modules depending on this module. The API of the module may be affected by changes in these modules. This includes, but is not limited to, classes in this module inherit or expose classes from the dependent modules.
- `PRIVATE_DEPENDS`: Dependencies which are only used in the implementation details of the module. The API of the module is not affected by changes in these modules.
- `OPTIONAL_DEPENDS`: Dependencies which will be used if available, but the implementation can cope with their absence. These are always treated as `PRIVATE_DEPENDS` if they are available.

Modules which are listed in `DEPENDS` or `PRIVATE_DEPENDS` are always available to the module and can be assumed to exist if the module is being built. Modules listed in `OPTIONAL_DEPENDS` cannot be assumed to exist. In CMake code, a `TARGET optional_depend` condition may be used to detect whether it is available or not. The module system will add a `VTK_MODULE_ENABLE_${module}` compilation definition set to either `0` or `1` if it is available for use in the module's code. This flag is made preprocessor-safe by replacing any `::` in the module name with `_`. So an optional dependency on `Namespace::Target` will use a flag named `VTK_MODULE_ENABLE_Namespace_Target`.

At this stage, the dependency tree for all scanned modules is traversed, marking dependencies of `YES` modules as those that should be built, marking modules depending on `NO` modules as not to be built (and triggering an error if a conflict is found). Any `WANT` modules that have not been found in the trees of `YES` or `NO` modules are then enabled with their dependencies.

There is a script to help figuring out dependencies when building your own modules or VTK-dependant code (*.cxx, *.h) in order to generate a `find_package` command. The required json argument is only available in a build tree though.

```
Utilities/Maintenance/FindNeededModules.py -s /path/to/sources -j path/to/vtk_build/
modules.json
```

Testing

There is some support for testing in the module system, but it is not as comprehensive as the build side. This is because testing infrastructure and strategies vary wildly between projects. Rather than trying to handle the minimum baseline of any plausible testing infrastructure or framework, the module system merely handles dependency management for testing and entering a subdirectory with the tests.

Modules may have `TEST_DEPENDS` and `TEST_OPTIONAL_DEPENDS` lists provided as well. These modules are required or optionally used by the testing code for the module.

When scanning, the `ENABLE_TESTS` argument may be set to `ON`, `OFF`, `WANT` (the default), or `DEFAULT`. Modules which appear in `TEST_DEPENDS` for the module are affected by this setting.

- `ON`: Modules required for testing are treated as required. Tests will be enabled.
- `OFF`: Tests will not be enabled.
- `WANT`: If possible, `TEST_DEPENDS` modules will also be enabled if they are not disabled in some other way.

- DEFAULT: Check when tests are checked whether all of TEST_DEPENDS are available. If they are, enable testing for the module, otherwise skip it.

The only guarantee for testing provided is that all modules in the TEST_DEPENDS will be available before the testing is added and TEST_OPTIONAL_DEPENDS are available if they'd be available at all (i.e., they won't be made available later).

Modules may also have TEST_LABELS set to ease labeling all tests for the module. The module system itself does nothing with this other than set a global property with the value. It is up to any test infrastructure used within the module's CMake code to make use of the value.

The tests for a module are expected to live in a subdirectory of the module code itself. The name of this directory is given by the TEST_DIRECTORY_NAME argument to the *vtk_module_build()* function. If the directory is available and the module's testing is enabled, the module system will add_subdirectory this directory at the appropriate time. This is decoupled so that testing code can depend on modules that depend on the module that is being tested and the same TARGET ${dependency} check can be used for optional module dependencies.

Building modules

After scanning is complete, *vtk_module_scan()* returns a list of modules and kits to build in the variables given by the PROVIDES_MODULES and PROVIDES_KITS arguments to it. It also provides lists of modules that were found during scanning that were not scanned by that call. These are given back in the variables passed to the UNRECOGNIZED_MODULES and REQUIRES_MODULES variables.

The UNRECOGNIZED_MODULES list contains modules passed to REQUIRES_MODULES and REJECT_MODULES that were not found during the scan. This typically indicates that the values passed to those arguments were not constructed properly. However, it may also mean that they should be passed on to further scans if they may be found elsewhere. Callers should handle the variable as necessary for their use case.

The REQUIRES_MODULES are modules that were named as dependencies of the scanned modules and need to be provided in some way before building the provided modules (the build step will require that they exist when it tries to build the modules which required them). These can be passed on to future REQUIRES_MODULES arguments in future scans or used to error out depending on the use case of the caller.

When using *vtk_module_build()*, the PROVIDES_MODULES and PROVIDES_KITS from a single scan should be passed together. Multiple scans may be built together as well if they all use the same build parameters as each other.

Build-time parameters

The *vtk_module_build()* function is where the decision to build with or without kits is decided through the BUILD_WITH_KITS option. Only if this is set will kits be built for this set of modules.

The decision to default third party modules to using an external or internal copy (where such a decision is possible) is done using the USE_EXTERNAL argument.

Where build artifacts end up in the build tree are left to CMake's typical variables for controlling these locations:

- CMAKE_ARCHIVE_OUTPUT_DIRECTORY
- CMAKE_LIBRARY_OUTPUT_DIRECTORY
- CMAKE_RUNTIME_OUTPUT_DIRECTORY

The defaults for these place outputs into the binary directory where the targets were added. The module system will set these to be sensible for itself if they are not already set, but it is recommended to set these at the top-level so that targets not built under *vtk_module_build()* also end up at a sensible location.

Library parameters

When building libraries, it is sometimes useful to have top-level control of library metadata. For example, VTK suffixes its library filenames with a version number. The variables that control this include:

- `LIBRARY_NAME_SUFFIX`: If non-empty, all libraries and executable names will be suffixed with this value prefixed with a hyphen (e.g., a suffix of `foo` will make `Namespace::Target`'s library be named `Target-foo` or, if the module sets its `LIBRARY_NAME` to `nsTarget`, `nsTarget-foo`).
- `VERSION`: Controls the `VERSION` property for all library modules.
- `SOVERSION`: Controls the `SOVERSION` property for all library modules.

Installation support

`vtk_module_build()` also offers arguments to aid in installing module artifacts. These include destinations for pieces that are installed, CMake packaging controls, and components to use for the installations.

A number of destinations control arguments are provided:

- `ARCHIVE_DESTINATION`
- `HEADERS_DESTINATION`
- `LIBRARY_DESTINATION`
- `RUNTIME_DESTINATION`
- `CMAKE_DESTINATION`
- `LICENSE_DESTINATION`
- `HIERARCHY_DESTINATION`

See the API documentation for default values for each which are based on `GNUInstallDirs` variables. Note that all installation destinations are expected to be relative paths. This is because the conveniences provided by the module system are all assumed to be installed to a single prefix (`CMAKE_INSTALL_PREFIX`) and placed underneath it.

Suppression of header installation is provided via the `INSTALL_HEADERS` argument to *`vtk_module_build()`*. Setting this to `OFF` will suppress the installation of:

- headers
- CMake package files
- hierarchy files (since their use requires headers)

Basically, suppression of headers means that SDK components for the built modules are not available in the install tree.

Components for the installation are provided via the `HEADERS_COMPONENT` and `TARGETS_COMPONENT` arguments. The former is used for SDK bits and the latter for runtime bits (libraries, executables, etc.).

For CMake package installation, the `PACKAGE` and `INSTALL_EXPORT` arguments are available. The former controls the names used by the CMake files created by the module system while the former is the export set to use for the member modules when creating those CMake files. Non-module targets may also exist in this export set when *`vtk_module_build()`* is called, but the export set is considered "closed" afterwards since it has already been exported (if `INSTALL_HEADERS` is true).

Test data information

The directory that is looked for in each module is specified by using the `TEST_DIRECTORY_NAME` argument. If it is set to the value of `NONE`, no testing directories will be searched for. It defaults to `Testing` due to VTK's conventions.

The module system, due to VTK's usage of it, has convenience parameters for controlling the `ExternalData` module that is available to testing infrastructure. These include:

- `TEST_DATA_TARGET`: The data target to use for tests.
- `TEST_INPUT_DATA_DIRECTORY`: Where `ExternalData` should look for data files.
- `TEST_OUTPUT_DATA_DIRECTORY`: Where `ExternalData` should place the downloaded data files.
- `TEST_OUTPUT_DIRECTORY`: Where tests should place output files.

Each is provided in the testing subdirectory as `_vtk_build_${name}`, so the `TEST_DATA_TARGET` argument is available as `_vtk_build_TEST_DATA_TARGET`.

Building a module

Building a module is basically the same as a normal CMake library or executable, but is wrapped to use arguments to facilitate wrapping, exporting, and installation of the tools as well.

There are two main functions provided for this:

- *`vtk_module_add_module()`*
- *`vtk_module_add_executable()`*

The former creates a library for the module being built while the latter can create an executable for the module itself or create utility executable associated with the module. The module system requires that the `CMakeLists.txt` for a module create a target with the name of the module. In the case of `INTERFACE` modules, it suffices to create the module manually in many cases.

Libraries

Most modules end up being libraries that can be linked against by other libraries. Due to cross-platform support generally being a good thing, the `EXPORT_MACRO_PREFIX` argument is provided to specify the prefix for macro names to be used by `GenerateExportHeader`. By default, the `LIBRARY_NAME` for the module is transformed to uppercase to make the prefix.

Some modules may need to add additional information to the library name that will be used that is not statically know and depends on other environmental settings. The `LIBRARY_NAME_SUFFIX` may be specified to add an additional suffix to the `LIBRARY_NAME` for the module. The *`vtk_module_build()`* `LIBRARY_NAME_SUFFIX` argument value will be appended to this name as well.

Normally, libraries are built according to the `BUILD_SHARED_LIBS` variable, however, some modules may need to be built statically all the time. The `FORCE_STATIC` parameter exists for this purpose. This is generally only necessary if the module is in some other must-be-static library's dependency tree (which may happen for a number of reasons). It is not an escape hatch for general usage; it is there because use cases which only support static libraries (even in a shared build) exist.

If a library module is part of a kit and it is being built via the *`vtk_module_build()`* `BUILD_WITH_KITS` argument, it will be built as an `OBJECT` library and the kit machinery in *`vtk_module_build()`* will create the resulting kit library artifact.

Header-only modules must pass `HEADER_ONLY` to create an `INTERFACE` library instead of expecting a linkable artifact.

Note: `HEADER_ONLY` modules which are part of kits is currently untested. This should be supported, but might not work at the moment.

Source listing

Instead of using CMake's "all sources in a single list" pattern for `add_library`, *`vtk_module_add_module()`* classifies its source files explicitly:

- `SOURCES`
- `HEADERS`
- `TEMPLATES`

The `HEADERS` and `TEMPLATES` are installed into the `HEADERS_DESTINATION` specified to *`vtk_module_build()`* and may be added to a subdirectory of this destination by using the `HEADERS_SUBDIR` argument. Note that the structure of the header paths passed is ignored. If more structure is required from the installed header layout, `vtk_module_install_headers()` should be used.

Files passed via `HEADERS` are treated as the API interface to the code of the module and are added to properties so that *language wrappers* can discover the API of the module.

Note: Only headers passed via `HEADERS` are eligible for wrapping; those installed via `vtk_module_install_headers()` are not. This is a known limitation at the moment.

There are also private variations for `HEADERS` and `TEMPLATES` named `PRIVATE_HEADERS` and `PRIVATE_TEMPLATES` respectively. These are never installed nor exposed to wrapping mechanisms.

There are also a couple of convenience parameters that use VTK's file naming conventions to ease usage. These include:

- `CLASSES`: For each value `<class>`, adds `<class>.cxx` to `SOURCES` and `<class>.h` to `HEADERS`.
- `TEMPLATE_CLASSES`: For each value `<class>`, adds `<class>.txx` to `TEMPLATES` and `<class>.h` to `HEADERS`.
- `PRIVATE_CLASSES`: For each value `<class>`, adds `<class>.cxx` to `SOURCES` and `<class>.h` to `PRIVATE_HEADERS`.
- `PRIVATE_TEMPLATE_CLASSES`: For each value `<class>`, adds `<class>.txx` to `PRIVATE_TEMPLATES` and `<class>.h` to `PRIVATE_HEADERS`.

Executables

Executables may be created using *`vtk_module_add_executable()`*. The first argument is the name of the executable to build. Since the scanning phase does not know what kind of target will be created for each module (and it may change based on other configuration values), an executable module which claims it is part of a kit raises an error since this is not possible to do.

For modules that are executables using this function, the metadata from the module information is used to set the relevant properties. The module dependencies are also automatically linked in the same way as a library module would do so.

For utility executables, `NO_INSTALL` may be passed to keep it within the build tree. It will not be available to consumers of the project. If the name of the executable is different from the target name, `BASENAME` may be used to change the executable's name.

Module APIs

All of CMake's `target_` function calls have *analogues* for modules. This is primarily due to the kits feature which causes the target name created by the module system that is required to use the `target_` functions dependent on whether the module is a member of a kit and kits are being built. The CMake version of the function and the module API analogue (as well as differences, if any) is:

- set_target_properties becomes *`vtk_module_set_properties()`*
- set_property(TARGET) becomes *`vtk_module_set_property()`*
- get_property(TARGET) becomes *`vtk_module_get_property()`*
- add_dependencies becomes *`vtk_module_depend()`*
- target_include_directories becomes *`vtk_module_include()`*
- target_compile_definitions becomes *`vtk_module_definitions()`*
- target_compile_options becomes *`vtk_module_compile_options()`*
- target_compile_features becomes *`vtk_module_compile_features()`*
- target_link_libraries becomes *`vtk_module_link()`*: When kits are enabled, any `PRIVATE` links are forwarded to the kit itself. This necessitates making all of these targets globally scoped rather than locally scoped.
- target_link_options becomes *`vtk_module_link_options()`*

Packaging support

Getting installed packages to work for CMake is, unfortunately, not trivial. The module system provides some support for helping with this, but it does place some extra constraints on the project so that some assumptions that vastly simplify the process can be made.

Assumptions

The main assumption is that all modules passed to a single *`vtk_module_build()`* have the same CMake namespace (the part up to and including the `::`, if any, in a module name. For exporting dependencies, that namespace matches the `PACKAGE` argument for *`vtk_module_build()`*. These are done so that the generated code can use `CMAKE_FIND_PACKAGE_NAME` variable can be used to discover information about the package that is being found.

The package support also assumes that all modules may be queried using `COMPONENTS` and `OPTIONAL_COMPONENTS` and that the component name for a module corresponds to the name of a module without the namespace.

These rules basically mean that a module named `Namespace::Target` will be found using `find_package(Namespace)`, that `COMPONENTS Target` may be passed to ensure that that module exists, and `OPTIONAL_COMPONENTS Target` may be passed to allow the component to not exist while not failing the main `find_package` call.

Creating a full package

The module system provides no support for the top-level file that is used by `find_package`. This is because this logic is highly project-specific and hard to generalize in a useful way. Instead, files are generated which should be included from the main file.

Here, the list of files generated are based on the `PACKAGE` argument passed to *`vtk_module_build()`*:

- `<PACKAGE>-targets.cmake`: The CMake-generated export file for the targets in the `INSTALL_EXPORT`.
- `<PACKAGE>-vtk-module-properties.cmake`: Properties for the targets exported into the build.

The module properties file must be included after the targets file so that they exist when it tries to add properties to the imported targets.

External dependencies

Since the module system is heavily skewed towards using imported targets, these targets show up by name in the `find_package` of the project as well. This means that these external projects need to be found to recreate their imported targets at that time. To this end, there is the *`vtk_module_export_find_packages()`* function. This function writes a file named according to its `FILE_NAME` argument and place it in the build and install trees according to its `CMAKE_DESTINATION` argument.

This file will be populated with logic to determine whether third party packages found using *`vtk_module_find_package()`* are required during the `find_package` of the package or not. It will forward `REQUIRED` and `QUIET` parameters to other `find_package` calls as necessary based on the `REQUIRED` and `QUIET` flags for the package and whether that call is involved in a non-optional `COMPONENT` (a component-less `find_package` call is assumed to mean "all components").

This file should be included after the `<PACKAGE>-vtk-module-properties.cmake` file generated by the *`vtk_module_build()`* call so that it can use the module dependency information set via that file.

After this file is included, for each component that it checks, it will set `${CMAKE_FIND_PACKAGE_NAME}_<component>_FOUND` to `0` if it is not valid and append a reason to `${CMAKE_FIND_PACKAGE_NAME}_<component>_NOT_FOUND_MESSAGE` so that the package can collate the reason why things are not available.

Setting the `_FOUND` variable

The module system does not currently help in determining the top-level `${CMAKE_FIND_PACKAGE_NAME}_FOUND` variable based on the results of the components that were requested and the status of dependent packages. This may be provided at some point, but there has not currently been enough experience to determine what patterns are available for factoring it out as a utility function.

The general pattern should be to go through the list of components requested, determine whether targets for those components exist. Then for each found component, use the module dependency information to ensure that all targets in the dependency trees are found (propagating not-found statuses through the dependency tree). The `${CMAKE_FIND_PACKAGE_NAME}_NOT_FOUND_MESSAGE` should be built up based on the reasons the `find_package` call did not work based on these discoveries.

This is the process for modules in a package, but packages may contain non-module components, and it is hard for the module system to provide support for them, so they are not attempted. See the CMake documentation for more details about creating a package configuration.

Advanced topics

There are a number of advanced features provided by the module system that are not normally required in a simple project.

Kits

Kits are described in *vtk.kit* files which act much like *vtk.module* files. However, they only have `NAME`, `LIBRARY_NAME`, and `DESCRIPTION` fields. These all act just like they do in the `vtk.module` context. These files may either be passed manually to *`vtk_module_scan()`* or discovered by using the *`vtk_module_find_kits()`* convenience function.

Before a module may be a member of a kit, a *vtk.kit* must declare it and be scanned at the same time. This means that kits may only contain modules that are scanned with them and cannot be extended later nor may kits be made of modules that they do not know about.

Requirements

In order to actually use kits, CMake 3.12 is necessary in order to do the `OBJECT` library manipulations done behind the scenes to make it Just Work. 3.8 is still the minimum version for using a project that is built with kits however. This is only checked when kits are actually in use, so projects requiring older CMake versions as their minimum version may still provide kits so that users with newer CMake versions can use them.

Kits create a single library on disk, but the usage requirements of the modules should still be the same (except for that which is inherently required to be different by combining libraries). So include directories, compile definitions, and other usage requirements should not leak from other modules that are members of the same kit.

Autoinit

The module system supports a mechanism for triggering static code construction for modules which require it. This cannot be done through normal CMake usage requirements because the requirements are intersectional. For example, a module `F` having a factory where module `I` provides an implementation for it means that a target linking to both `F` and `I` needs to ensure that `I` registers its implementation to the factory code. There is no such support in CMake and due to the complexities and code generation involved with this support, it is unlikely to exist.

Code which uses modules may call the *`vtk_module_autoinit()`* function to use this functionality. The list of modules passed to the function are used to compute the defines necessary to trigger the registration to factories when necessary.

For details on the implementation of the autoinit system, please see *the relevant section* in the API documentation.

Wrapping

VTK comes with support for wrapping its classes into other languages. Currently, VTK supports wrapping its classes for use in the Python and Java languages. In order to wrap a set of modules for a language, a separate function is used for each language.

All languages read the headers of classes with a `__VTK_WRAP__` preprocessor definition defined. This may be used to hide methods or other details from the wrapping code if wanted.

Python

For Python, the *`vtk_module_wrap_python()`* function must be used. This function takes a list of modules in its `MODULES` argument and creates Python modules for use under the `PYTHON_PACKAGE` package. No `__init__.py` for this package is created automatically and must be provided in some other way.

A target named by the `TARGET` argument is created and installed. This target may be linked to in order to be able to import static Python modules. In this case, a header and function named according to the basename of `TARGET` (e.g., `VTK::PythonWrapped` has a basename of `PythonWrapped`) must be used. The header is named `<TARGET_BASENAME>.h` and the function which adds the wrapped modules to the static import table is `<void TARGET_BASENAME>_load()`. This function is also created in shared builds, but does nothing so that it may always be called in static or shared builds.

The modules will be installed under the `MODULE_DESTINATION` given to the function into the `PYTHON_PACKAGE` directory needed for it. The *`vtk_module_python_default_destination()`* function is used to determine a default if one is not passed.

The Python wrappers define a `__VTK_WRAP_PYTHON__` preprocessor definition when reading code which may be used to hide methods or other details from the Python wrapping code.

Java

For Java, the *`vtk_module_wrap_java()`* function must be used. This function creates Java sources for classes in the modules passed in its `MODULES` argument. The sources are written to a `JAVA_OUTPUT` directory. These then can be compiled by CMake normally.

For this purpose, there are `<MODULE>Java` targets which contain a `_vtk_module_java_files` properties containing a list of `.java` sources generated for the given module. There is also a `<MODULE>Java-java-sources` target which may be depended upon if just the source generation needs to used in an `add_dependencies` call.

The Java wrappers define a `__VTK_WRAP_JAVA__` preprocessor definition when reading code which may be used to hide methods or other details from the Java wrapping code.

Hierarchy files

Hierarchy files are used by the language wrapper tools to know the class inheritance for classes within a module. Each module has a hierarchy file associated with it. The path to a module's hierarchy file is stored in its `hierarchy` module property.

Third party

The module system has support for representing third party modules in its build. These may be built as part of the project or represented using other mechanisms (usually `find_package` and a set of imported targets from it).

The primary API is *`vtk_module_third_party()`* which creates a `VTK_MODULE_USE_EXTERNAL_Namespace_Target` option for the module to switch between an internal and external source for the third party code. This value defaults to the setting of the `USE_EXTERNAL` argument for the calling *`vtk_module_build()`* function. Arguments passed under the `INTERNAL` and `EXTERNAL` arguments to this command are then passed on to *`vtk_module_third_party_internal()`* or *`vtk_module_third_party_external()`*, respectively, depending on the `VTK_MODULE_USE_EXTERNAL_Namespace_Target` option.

Note that third party modules (marked as such by adding the `THIRD_PARTY` keyword to a `vtk.module` file) may not be part of a kit, be wrapped, or participate in autoinit.

External third party modules

External modules are found using CMake's `find_package` mechanism. In addition to the arguments supported by *`vtk_module_find_package()`* (except `PRIVATE` and `PRIVATE_IF_SHARED`), information about the found package is used to construct a module target which represents the third party package. The preferred mechanism is to give a list of imported targets to the `LIBRARIES` argument. These will be added to the `INTERFACE` of the module and provide the third party package for use within the module system.

If imported targets are not available (they really should be created if not), variable names may be passed to `INCLUDE_DIRS`, `LIBRARIES`, and `DEFINITIONS` to create the module interface.

In addition, any variables which should be forwarded from the package to the rest of the build may be specified using the `USE_VARIABLES` argument.

The `STANDARD_INCLUDE_DIRS` argument creates an include interface for the module target which includes the "standard" module include directories to. Basically, the source and binary directories of the module.

Internal third party modules

Internal modules are those that may be built as part of the build. These should ideally specify a set of `LICENSE_FILES` indicating the license status of the third party code. These files will be installed along with the third party package to aid in any licensing requirements of the code. It is also recommended to set the `VERSION` argument so that it is known what version of the code is provided at a glance.

By default, the `LIBRARY_NAME` of the module is used as the name of the subdirectory to include, but this may be changed by using the `SUBDIRECTORY` argument.

Header-only third party modules may be indicated by using the `HEADER_ONLY` argument. Modules which represent multiple libraries at once from a project may use the `INTERFACE` argument.

The `STANDARD_INCLUDE_DIRS` argument creates an include interface for the module target which includes the "standard" module include directories to. Basically, the source and binary directories of the module. A subdirectory may be used by setting the `HEADERS_SUBDIR` option. It is implied for `HEADERS_ONLY` third party modules.

After the subdirectory is added a target with the module's name must exist. However, a target is automatically created if it is `HEADERS_ONLY`.

Properly shipping internal third party code

There are many things that really should be done to ship internal third party code (also known as vendoring) properly. The issue is mainly that the internal code may conflict with other code bringing in another copy of the same package into a process. Most platforms do not behave well in this situation.

In order to avoid conflicts at every level possible, a process called "name mangling" should be performed. A non-exhaustive list of name manglings that must be done to fully handle this case includes:

- moving headers to a subdirectory (to avoid compilations from finding incompatible headers);
- changing the library name (to avoid DLL lookups from finding incompatible copies); and
- mangling symbols (to avoid symbol lookup from confusing two copies in the same process).

Some projects may need further work like editing CMake APIs or the like to be mangled as well.

Moving headers and changing library names is fairly straightforward by editing CMake code. Mangling symbols usually involves creating a header which has a `#define` for each public symbol to change its name at runtime to be distinct from another copy that may end up existing in the same process from another project.

Typically, a header needs to be created at the module level which hides the differences between third party code which may or may not be provided by an external package. In this case, it is recommended that code using the third party module use unmangled names and let the module interface and mangling headers handle the mangling at that level.

Debugging

The module system can output debugging information about its inner workings by using the `_vtk_module_log` variable. This variable is a list of "domains" to log about, or the special `ALL` value causes all domains to log output. The following domains are used in the internals of the module system:

- `kit`: discovery and membership of kits
- `module`: discovery and `CONDITION` results of modules
- `enable`: resolution of the enable status of modules
- `provide`: determination of module provision
- `building`: when building a module occurs
- `testing`: missing test dependencies

It is encouraged that projects expose user-friendly flags to control logging rather than exposing `_vtk_module_log` directly.

Control variables

These variables do not follow the API convention and are used if set:

- `_vtk_module_warnings`: If enabled, "strict" warnings are generated. These are not strictly problems, but may be used as linting for improving usage of the module system.
- `_vtk_module_log`: A list of "domains" to output debugging information.
- `_vtk_module_group_default_${group}`: used to set a non-`DEFAULT` default for group settings.

Some mechanisms use global properties instead:

- `_vtk_module_autoinit_include`: The file that needs to be included in order to make the `VTK_MODULE_AUTOINIT` symbol available for use in the *autoinit* support.

SPDX files generation

The generation of VTK module SPDX files relies on three components:

- SPDX arguments in *`vtk_module_build()`*
- SPDX arguments in each *vtk.module*
- SPDX Tags in the *sources files*

SPDX files are named after `<ModuleName>.spdx` and are generated for all VTK modules.

Generated SPDX files are based on the SPDX 2.2 specification.

If some information is missing, VTK will warn during configuration or during build but the SPDX file will still be generated with unknown fields being attributed a `NOASSERTION` or other default value.

The collected license identifiers are joined together using `AND` keyword.

Similarly all collected copyright texts are joined using a new line.

SPDX arguments in `vtk_module_build`

Support for SPDX file generation requires to specify the following *`vtk_module_build()`* arguments:

- `GENERATE_SPDX`
- `SPDX_DOCUMENT_NAMESPACE`
- `SPDX_DOWNLOAD_LOCATION`

`GENERATE_SPDX` is used to enable the generation and install of SPDX file for each modules. Set this to `ON` to enable it.

`SPDX_DOCUMENT_NAMESPACE` is used as a basename for the `DocumentNamespace` SPDX field. The name of the module will simply be appended to the basename. If not provided, `https://vtk.org/spdx` will be used. This is the value VTK project uses as well. Note that the namespace does not need to be an actual website URL, but just a unique Uniform Resource Identifier (URI).

Caution: If VTK decide to host SPDX files in the future, the namespace in use for the VTK SPDX files may change accordingly.

`SPDX_DOWNLOAD_LOCATION` is used as a basename for the `PackageDownloadLocation` when not provided at module level. The relative path to the module will simply be appended in order to generate the actual `PackageDownloadLocation` SPDX field. If not provided at module or in *`vtk_module_build()`*, `NOASSERTION` will be used.

SPDX arguments in `vtk.module`

Defining these three arguments in *vtk.module* is required:

- `SPDX_LICENSE_IDENTIFIER`
- `SPDX_COPYRIGHT_TEXT`
- `SPDX_DOWNLOAD_LOCATION`

`SPDX_LICENSE_IDENTIFIER` is an expected field corresponding to the `PackageLicenseDeclared` SPDX field that is considered as the global license for all files of the module that are not parsed during generation. This field is used to set the `PackageLicenseConcluded` SPDX field.

Note: The SPDX generation system do not and cannot replace the `LICENSE_FILES` mechanism. Indeed, certains license (e.g Apache 2.0) requires additional files (e.g `NOTICE`) to also be distributed.

`SPDX_COPYRIGHT_TEXT` is an expected field that correspond to the copyright applying to all files that are not parsed during generation, it is used to generate `PackageCopyrightText`.

`SPDX_DOWNLOAD_LOCATION` is a optional field for modules (see above for setting this in `vtk_module_build`) and expected field for *third parties*. If provided, it is used as is for the `PackageDownloadLocation` SPDX field.

SPDX arguments in `vtk_module_add_module`

It is possible to specify a `SPDX_SKIP_REGEX` when adding a module in order to skip specific file during *SPDX tags parsing*. It is a python regex which is used to match with the filename of the source files.

Custom license support

If the VTK module contains a custom license that is not part of the SPDX license list then adding a custom license may be needed.

The SPDX generation system support to specify exactly one custom license by module, supplemental to standard licenses. The text of this license should be made available in a file and added to the module definition using `SPDX_CUSTOM_LICENSE_FILE` , the name of the license should be specified using `SPDX_CUSTOM_LICENSE_NAME` (eg: `LicenseName` and the `SPDX_LICENSE_IDENTIFIER` for this license should be `LicenseRef-` followed by the name (eg: `LicenseRef-licenseName`). See *this entry* for more info.

Note: If this custom license is to be added to VTK proper, it must be compatible with the BSD-3-Clause license of VTK and not add more restriction to the code.

SPDX Tags in the sources files

For VTK modules (except the one declared as `THIRD_PARTY`), *sources files* are parsed for specific SPDX tags in a specific order.

First `N` lines of with the the `SPDX-FileCopyrightText` tag, then one line with the `SPDX-License-Identifier` tag. Like this:

```
// SPDX-FileCopyrightText: Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
// SPDX-FileCopyrightText: Copyright (c) Awesome contributor
// SPDX-License-Identifier: BSD-3-Clause
```

If a source file does not contain both `SPDX-FileCopyrightText` and `SPDX-License-Identifier` tags, a warning at build time is reported.

Limitations

- Correctness of the `SPDX-FileCopyrightText` and `SPDX-License-Identifier` tags is not ensured. The value will be used as is.
- The generated SPDX files only include the Package information section. This means that there are no File information sections describing source files or build artifacts.
- Third party source files are not parsed for SPDX tags.
- Adding empty lines between `// SPDX-FileCopyrightText` and `// SPDX-License-Identifier` tags is not supported.
- Certain files are not parsed at all, eg: cmake files, python files, test files, …

8.3.2 vtkModule

`_vtk_module_debug`

Conditionally output debug statements *module-internal*

The `_vtk_module_debug()` function is provided to assist in debugging. It is controlled by the *_vtk_module_log* variable which contains a list of "domains" to debug.

```
_vtk_module_debug(<domain> <format>)
```

If the *domain* is enabled for debugging, the *format* argument is configured and printed. It should contain @ variable expansions to replace rather than it being done outside. This helps to avoid the cost of generating large strings when debugging is disabled.

`vtk_module_find_kits`

Find *vtk.kit* files in a set of directories *module*

```
vtk_module_find_kits(<output> [<directory>...])
```

This scans the given directories recursively for *vtk.kit* files and put the paths into the output variable.

`vtk_module_find_modules`

Find *vtk.module* files in a set of directories *module*

```
vtk_module_find_modules(<output> [<directory>...])
```

This scans the given directories recursively for `vtk.module` files and put the paths into the output variable. Note that module files are assumed to live next to the `CMakeLists.txt` file which will build the module.

`_vtk_module_split_module_name`

Split a module name into a namespace and target component *module-internal*

Module names may include a namespace. This function splits the name into a namespace and target name part.

```
_vtk_module_split_module_name(<name> <prefix>)
```

The `<prefix>_NAMESPACE` and `<prefix>_TARGET_NAME` variables will be set in the calling scope.

`_vtk_module_optional_dependency_exists`

Detect whether an optional dependency exists or not. *module-internal*

Optional dependencies need to be detected namespace and target name part.

```
_vtk_module_optional_dependency_exists(<dependency>
  SATISFIED_VAR <var>)
```

The result will be returned in the variable specified by `SATISFIED_VAR`.

vtk.module file contents

The *vtk.module* file is parsed and used as arguments to a CMake function which stores information about the module for use when building it. Note that no variable expansion is allowed and it is not CMake code, so no control flow is allowed. Comments are supported and any content after a # on a line is treated as a comment. Due to the breakdown of the content, quotes are not meaningful within the files.

Example:

```
NAME
  VTK::CommonCore
LIBRARY_NAME
  vtkCommonCore
DESCRIPTION
  The base VTK library.
LICENSE_FILES
  Copyright.txt
SPDX_COPYRIGHT_TEXT
  Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
SPDX_LICENSE_IDENTIFIER
  BSD-3-Clause
GROUPS
  StandAlone
DEPENDS
  VTK::kwiml
PRIVATE_DEPENDS
  VTK::vtksys
  VTK::utf8
```

All values are optional unless otherwise noted. The following arguments are supported:

- `NAME`: (Required) The name of the module.
- `LIBRARY_NAME`: The base name of the library file. It defaults to the module name, but any namespaces are removed. For example, a `NS::Foo` module will have a default `LIBRARY_NAME` of `Foo`.
- `DESCRIPTION`: (Recommended) Short text describing what the module is for.
- `KIT`: The name of the kit the module belongs to (see `Kits files` for more information).
- `IMPLEMENTABLE`: If present, the module contains logic which supports the autoinit functionality.
- `GROUPS`: Modules may belong to "groups" which is exposed as a build option. This allows for enabling a set of modules with a single build option.
- `CONDITION`: Arguments to CMake's `if` command which may be used to hide the module for certain platforms or other reasons. If the expression is false, the module is completely ignored.
- `DEPENDS`: A list of modules which are required by this module and modules using this module.
- `PRIVATE_DEPENDS`: A list of modules which are required by this module, but not by those using this module.

- OPTIONAL_DEPENDS: A list of modules which are used by this module if enabled; these are treated as PRIVATE_DEPENDS if they exist.
- ORDER_DEPENDS: Dependencies which only matter for ordering. This does not mean that the module will be enabled, just guaranteed to build before this module.
- IMPLEMENTS: A list of modules for which this module needs to register with.
- TEST_DEPENDS: Modules required by the test suite for this module.
- TEST_OPTIONAL_DEPENDS: Modules used by the test suite for this module if available.
- TEST_LABELS: Labels to apply to the tests of this module. By default, the module name is applied as a label.
- EXCLUDE_WRAP: If present, this module should not be wrapped in any language.
- INCLUDE_MARSHAL: If present, this module opts into automatic code generation of (de)serializers. This option requires that the module is not excluded from wrapping with *EXCLUDE_WRAP*.
- THIRD_PARTY: If present, this module is a third party module.
- LICENSE_FILES: A list of license files to install for the module.
- SPDX_LICENSE_IDENTIFIER: A license identifier for SPDX file generation.
- SPDX_DOWNLOAD_LOCATION: A download location for the SPDX file generation.
- SPDX_COPYRIGHT_TEXT: A copyright text for the SPDX file generation.
- SPDX_CUSTOM_LICENSE_FILE: A relative path to a single custom license file to include in generated SPDX file.
- SPDX_CUSTOM_LICENSE_NAME: The name of the single custom license, without the LicenseRef-

_vtk_module_parse_module_args

Parse vtk.module file contents

module-impl

This macro places all vtk.module keyword "arguments" into the caller's scope prefixed with the value of name_output which is set to the NAME of the module.

```
_vtk_module_parse_module_args(name_output <vtk.module args...>)
```

For example, this vtk.module file:

```
NAME
  Namespace::Target
LIBRARY_NAME
  nsTarget
```

called with _vtk_module_parse_module_args(name ...) will set the following variables in the calling scope:

- name: Namespace::Target
- Namespace::Target_LIBRARY_NAME: nsTarget

With namespace support for module names, the variable should instead be referenced via ${${name}_LIBRARY_NAME} instead.

vtk.kit file contents

The *vtk.kit* file is parsed similarly to *vtk.module* files. Kits are intended to bring together related modules into a single library in order to reduce the number of objects that linkers need to deal with.

Example:

```
NAME
  VTK::Common
LIBRARY_NAME
  vtkCommon
DESCRIPTION
  Core utilities for VTK.
```

All values are optional unless otherwise noted. The following arguments are supported:

- `NAME`: (Required) The name of the kit.
- `LIBRARY_NAME`: The base name of the library file. It defaults to the module name, but any namespaces are removed. For example, a `NS::Foo` module will have a default `LIBRARY_NAME` of `Foo`.
- `DESCRIPTION`: (Recommended) Short text describing what the kit contains.

`_vtk_module_parse_kit_args`

Parse *vtk.kit* file contents *module-impl*

Just like *`_vtk_module_parse_module_args()`*, but for kits.

Enable status values

Modules and groups are enable and disable preferences are specified using a 5-way flag setting:

- `YES`: The module or group must be built.
- `NO`: The module or group must not be built.
- `WANT`: The module or group should be built if possible.
- `DONT_WANT`: The module or group should only be built if required (e.g., via a dependency).
- `DEFAULT`: Acts as either `WANT` or `DONT_WANT` based on the group settings for the module or `WANT_BY_DEFAULT` option to *`vtk_module_scan()`* if no other preference is specified. This is usually handled via another setting in the main project.

If a `YES` module preference requires a module with a `NO` preference, an error is raised.

A module with a setting of `DEFAULT` will look for its first non-`DEFAULT` group setting and only if all of those are set to `DEFAULT` is the `WANT_BY_DEFAULT` setting used.

`_vtk_module_verify_enable_value`

Verify enable values *module-impl*

Verifies that the variable named as the first parameter is a valid *enable status* value.

```
_vtk_module_verify_enable_value(var)
```

`vtk_module_scan`

Scan modules and kits *module*

Once all of the modules and kits files have been found, they are "scanned" to determine what modules are enabled or required.

```
vtk_module_scan(
  MODULE_FILES              <file>...
  [KIT_FILES                <file>...]
  PROVIDES_MODULES          <variable>
  [PROVIDES_KITS            <variable>]
  [REQUIRES_MODULES         <variable>]
  [REQUEST_MODULES          <module>...]
  [REJECT_MODULES           <module>...]
  [UNRECOGNIZED_MODULES     <variable>]
  [WANT_BY_DEFAULT          <ON|OFF>]
  [HIDE_MODULES_FROM_CACHE  <ON|OFF>]
  [ENABLE_TESTS             <ON|OFF|WANT|DEFAULT>])
```

The MODULE_FILES and PROVIDES_MODULES arguments are required. Modules which refer to kits must be scanned at the same time as their kits. This is so that modules may not add themselves to kits declared prior. The arguments are as follows:

- MODULE_FILES: (Required) The list of module files to scan.
- KIT_FILES: The list of kit files to scan.
- PROVIDES_MODULES: (Required) This variable will contain the list of modules which are enabled due to this scan.
- PROVIDES_KITS: (Required if KIT_FILES are provided) This variable will contain the list of kits which are enabled due to this scan.
- REQUIRES_MODULES: This variable will contain the list of modules required by the enabled modules that were not scanned.
- REQUEST_MODULES: The list of modules required by previous scans.
- REJECT_MODULES: The list of modules to exclude from the scan. If any of these modules are required, an error will be raised.
- UNRECOGNIZED_MODULES: This variable will contain the list of requested modules that were not scanned.
- WANT_BY_DEFAULT: (Defaults to OFF) Whether modules should default to being built or not.
- HIDE_MODULES_FROM_CACHE: (Defaults to OFF) Whether or not to hide the control variables from the cache or not. If enabled, modules will not be built unless they are required elsewhere.
- ENABLE_TESTS: (Defaults to DEFAULT) Whether or not modules required by the tests for the scanned modules should be enabled or not.
  - ON: Modules listed as TEST_DEPENDS will be required.
  - OFF: Test modules will not be considered.
  - WANT: Test dependencies will enable modules if possible. Note that this has known issues where modules required only via testing may not have their dependencies enabled.
  - DEFAULT: Test modules will be enabled if their required dependencies are satisfied and skipped otherwise.

To make error messages clearer, modules passed to REQUIRES_MODULES and REJECT_MODULES may have a _vtk_module_reason_<MODULE> variable set to the reason for the module appearing in either argument. For example, if the Package::Frobnitz module is required due to a ENABLE_FROBNITZ cache variable:

```
set("_vtk_module_reason_Package::Frobnitz"
  "via the `ENABLE_FROBNITZ` setting")
```

Additionally, the reason for the WANT_BY_DEFAULT value may be provided via the _vtk_module_reason_WANT_BY_DEFAULT variable.

Scanning multiple groups of modules

When scanning complicated projects, multiple scans may be required to get defaults set properly. The REQUIRES_MODULES, REQUEST_MODULES, and UNRECOGNIZED_MODULES arguments are meant to deal with this case. As an example, imagine a project with its source code, third party dependencies, as well as some utility modules which should only be built as necessary. Here, the project would perform three scans, one for each "grouping" of modules:

```
# Scan our modules first because we need to know what of the other groups we
# need.
vtk_module_find_modules(our_modules "${CMAKE_CURRENT_SOURCE_DIR}/src")
vtk_module_scan(
  MODULE_FILES      ${our_modules}
  PROVIDES_MODULES  our_enabled_modules
  REQUIRES_MODULES  required_modules)

# Scan the third party modules, requesting only those that are necessary, but
# allowing them to be toggled during the build.
vtk_module_find_modules(third_party_modules "${CMAKE_CURRENT_SOURCE_DIR}/third-party")
vtk_module_scan(
  MODULE_FILES            ${third_party_modules}
  PROVIDES_MODULES        third_party_enabled_modules
  # These modules were requested by an earlier scan.
  REQUEST_MODULES         ${required_modules}
  REQUIRES_MODULES        required_modules
  UNRECOGNIZED_MODULES    unrecognized_modules)

# These modules are internal and should only be built if necessary. There is no
# need to support them being enabled independently, so hide them from the
# cache.
vtk_module_find_modules(utility_modules "${CMAKE_CURRENT_SOURCE_DIR}/utilities")
vtk_module_scan(
  MODULE_FILES            ${utility_modules}
  PROVIDES_MODULES        utility_enabled_modules
  # These modules were either requested or unrecognized by an earlier scan.
  REQUEST_MODULES         ${required_modules}
                          ${unrecognized_modules}
  REQUIRES_MODULES        required_modules
  UNRECOGNIZED_MODULES    unrecognized_modules
  HIDE_MODULES_FROM_CACHE ON)

if (required_modules OR unrecognized_modules)
  # Not all of the modules we required were found. This should probably error out.
endif ()
```

Module-as-target functions

Due to the nature of VTK modules supporting being built as kits, the module name might not be usable as a target to CMake's *target_* family of commands. Instead, there are various wrappers around them which take the module name as an argument. These handle the forwarding of relevant information to the kit library as well where necessary.

- *vtk_module_set_properties()*
- *vtk_module_set_property()*
- *vtk_module_get_property()*
- *vtk_module_depend()*
- *vtk_module_include()*
- *vtk_module_definitions()*
- *vtk_module_compile_options()*
- *vtk_module_compile_features()*
- *vtk_module_link()*
- *vtk_module_link_options()*

Module target internals

When manipulating modules as targets, there are a few functions provided for use in wrapping code to more easily access them.

- *_vtk_module_real_target()*
- *_vtk_module_real_target_kit()*

  _vtk_module_real_target

  The real target for a module *module-internal*

  ```
  _vtk_module_real_target(<var> <module>)
  ```

  Sometimes the actual, core target for a module is required (e.g., setting CMake-level target properties or install rules). This function returns the real target for a module.

_vtk_module_real_target_kit

The real target for a kit *module-internal*

```
_vtk_module_real_target_kit(<var> <kit>)
```

Sometimes the actual, core target for a module is required (e.g., setting CMake-level target properties or install rules). This function returns the real target for a kit.

vtk_module_set_properties

Set multiple properties on a module *module*

A wrapper around *set_target_properties* that works for modules.

```
vtk_module_set_properties(<module>
  [<property> <value>]...)
```

`vtk_module_set_property`

Set a property on a module. *module*

A wrapper around `set_property(TARGET)` that works for modules.

```
vtk_module_set_property(<module>
  [APPEND] [APPEND_STRING]
  PROPERTY  <property>
  VALUE     <value>...)
```

`vtk_module_get_property`

Get a property from a module *module*

A wrapper around *get_property(TARGET)* that works for modules.

```
vtk_module_get_property(<module>
  PROPERTY  <property>
  VARIABLE  <variable>)
```

The variable name passed to the `VARIABLE` argument will be unset if the property is not set (rather than the empty string).

`_vtk_module_target_function`

Generate arguments for target function wrappers *module-impl*

Create the `INTERFACE`, `PUBLIC`, and `PRIVATE` arguments for a function wrapping CMake's `target_` functions to call the wrapped function.

This is necessary because not all of the functions support empty lists given a keyword.

`vtk_module_depend`

Add dependencies to a module *module*

A wrapper around `add_dependencies` that works for modules.

```
vtk_module_depend(<module> <depend>...)
```

`vtk_module_sources`

Add source files to a module. *module*

A wrapper around *target_sources* that works for modules.

```
vtk_module_sources(<module>
  [PUBLIC     <source>...]
  [PRIVATE    <source>...]
  [INTERFACE  <source>...])
```

`vtk_module_include`

Add include directories to a module *module*

A wrapper around *target_include_directories* that works for modules.

```
vtk_module_include(<module>
  [SYSTEM]
  [PUBLIC     <directory>...]
  [PRIVATE    <directory>...]
  [INTERFACE  <directory>...])
```

vtk_module_definitions

Add compile definitions to a module. *module*

A wrapper around `target_compile_definitions` that works for modules.

```
vtk_module_definitions(<module>
  [PUBLIC     <define>...]
  [PRIVATE    <define>...]
  [INTERFACE  <define>...])
```

vtk_module_compile_options

Add compile options to a module. *module*

A wrapper around `target_compile_options` that works for modules.

```
vtk_module_compile_options(<module>
  [PUBLIC     <option>...]
  [PRIVATE    <option>...]
  [INTERFACE  <option>...])
```

vtk_module_compile_features

Add compile features to a module. *module*

A wrapper around *target_compile_features* that works for modules.

```
vtk_module_compile_features(<module>
  [PUBLIC     <feature>...]
  [PRIVATE    <feature>...]
  [INTERFACE  <feature>...])
```

_vtk_private_kit_link_target

Manage the private link target for a module. *module-impl*

This function manages the private link target for a module.

```
_vtk_private_kit_link_target(<module>
  [CREATE_IF_NEEDED]
  [SETUP_TARGET_NAME <var>]
  [USAGE_TARGET_NAME <var>])
```

vtk_module_link

Add link libraries to a module. *module*

A wrapper around *target_link_libraries* that works for modules. Note that this function does extra work in kit builds, so circumventing it may break in kit builds.

The `NO_KIT_EXPORT_IF_SHARED` argument may be passed to additionally prevent leaking `PRIVATE` link targets from kit builds. Intended to be used for targets coming from a `vtk_module_find_package(PRIVATE_IF_SHARED)` call. Applies to all `PRIVATE` arguments; if different treatment is needed for subsets of these arguments, use a separate call to `vtk_module_link`.

```
vtk_module_link(<module>
  [NO_KIT_EXPORT_IF_SHARED]
  [PUBLIC     <link item>...]
  [PRIVATE    <link item>...]
  [INTERFACE  <link item>...])
```

vtk_module_link_options

Add link options to a module. *module*

A wrapper around *target_link_options* that works for modules.

```
vtk_module_link_options(<module>
  [PUBLIC     <option>...]
  [PRIVATE    <option>...]
  [INTERFACE  <option>...])
```

Module properties

module-internal

The VTK module system leverages CMake's target propagation and storage. As such, there are a number of properties added to the targets representing modules. These properties are intended for use by the module system and associated functionality. In particular, more properties may be available by language wrappers.

Naming properties

When creating properties for use with the module system, they should be prefixed with `INTERFACE_vtk_module_`. The `INTERFACE_` portion is required in order to work with interface libraries. The `vtk_module_` portion is to avoid colliding with any other properties. This function assumes this naming scheme for some of its convenience features as well.

Properties should be the same in the local build as well as when imported to ease use.

VTK module system properties

There are a number of properties that are used and expected by the core of the module system. These are generally module metadata (module dependencies, whether to wrap or not, etc.). The properties all have the `INTERFACE_vtk_module_` prefix mentioned in the previous section.

- `third_party`: If set, the module represents a third party dependency and should be treated specially. Third party modules are very restricted and generally do not have any other properties set on them.
- `exclude_wrap`: If set, the module should not be wrapped by an external language.
- `depends`: The list of dependent modules. Language wrappers will generally require this to satisfy references to parent classes of the classes in the module.
- `private_depends`: The list of privately dependent modules. Language wrappers may require this to satisfy references to parent classes of the classes in the module.
- `optional_depends`: The list of optionally dependent modules. Language wrappers may require this to satisfy references to parent classes of the classes in the module.
- `kit`: The kit the module is a member of. Only set if the module is actually a member of the kit (i.e., the module was built with `BUILD_WITH_KITS ON`).
- `implements`: The list of modules for which this module registers to. This is used by the autoinit subsystem and generally is not required.
- `implementable`: If set, this module provides registries which may be populated by dependent modules. It is used to check the `implements` property to help minimize unnecessary work from the autoinit subsystem.

- `needs_autoinit`: If set, linking to this module requires the autoinit subsystem to ensure that registries in modules are fully populated.
- `headers`: Paths to the public headers from the module. These are the headers which should be handled by language wrappers.
- `hierarchy`: The path to the hierarchy file describing inheritance of the classes for use in language wrappers.
- **`forward_link`: Usage requirements that must be forwarded even though the**
  module is linked to privately.
- `include_marshal`: If set, the whole module opts into automatic code generation of (de)serializers. Note that only classes annotated with VTK_MARSHALAUTO are considered for code generation.

Kits have the following properties available (but only if kits are enabled):

- `kit_modules`: Modules which are compiled into the kit.

`_vtk_module_set_module_property`

Set a module property. *module-internal*

This function sets a *module property* on a module. The required prefix will automatically be added to the passed name.

```
_vtk_module_set_module_property(<module>
  [APPEND] [APPEND_STRING]
  PROPERTY  <property>
  VALUE     <value>...)
```

`_vtk_module_get_module_property`

Get a module property. *module-internal*

Get a *module property* from a module.

```
_vtk_module_get_module_property(<module>
  PROPERTY  <property>
  VARIABLE  <variable>)
```

As with *`vtk_module_get_property()`*, the output variable will be unset if the property is not set. The property name is automatically prepended with the required prefix.

`_vtk_module_check_destinations`

Check that destinations are valid. *module-internal*

All installation destinations are expected to be relative so that `CMAKE_INSTALL_PREFIX` can be relied upon in all code paths. This function may be used to verify that destinations are relative.

```
_vtk_module_check_destinations(<prefix> [<suffix>...])
```

For each `suffix`, `prefix` is prefixed to it and the resulting variable name is checked for validity as an install prefix. Raises an error if any is invalid.

`_vtk_module_write_import_prefix`

Write an import prefix statement. *module-internal*

CMake files, once installed, may need to construct paths to other locations within the install prefix. This function writes a prefix computation for file given its install destination.

```
_vtk_module_write_import_prefix(<file> <destination>)
```

The passed file is cleared so that it occurs at the top of the file. The prefix is available in the file as the `_vtk_module_import_prefix` variable. It is recommended to unset the variable at the end of the file.

`_vtk_module_export_properties`

Export properties on modules and targets. *module-internal*

This function is intended for use in support functions which leverage the module system, not by general system users. This function supports exporting properties from the build into dependencies via target properties which are loaded from a project's config file which is loaded via CMake's `find_package` function.

```
_vtk_module_export_properties(
  [MODULE       <module>]
  [KIT          <kit>]
  BUILD_FILE    <path>
  INSTALL_FILE  <path>
  [PROPERTIES                <property>...]
  [FROM_GLOBAL_PROPERTIES    <property fragment>...]
  [SPLIT_INSTALL_PROPERTIES <property fragment>...])
```

The `BUILD_FILE` and `INSTALL_FILE` arguments are required. Exactly one of `MODULE` and `KIT` is also required. The `MODULE` or `KIT` argument holds the name of the module or kit that will have properties exported. The `BUILD_FILE` and `INSTALL_FILE` paths are *appended to*. As such, when setting up these files, it should be preceded with:

```
file(WRITE "${build_file}")
file(WRITE "${install_file}")
```

To avoid accidental usage of the install file from the build tree, it is recommended to store it under a `CMakeFiles/` directory in the build tree with an additional `.install` suffix and use `install(RENAME)` to rename it at install time.

The set of properties exported is computed as follows:

- `PROPERTIES` queries the module target for the given property and exports its value as-is to both the build and install files. In addition, these properties are set on the target directly as the same name.
- `FROM_GLOBAL_PROPERTIES` queries the global `_vtk_module_<MODULE>_<fragment>` property and exports it to both the build and install files as `INTERFACE_vtk_module_<fragment>`.
- `SPLIT_INSTALL_PROPERTIES` queries the target for `INTERFACE_vtk_module_<fragment>` and exports its value to the build file and `INTERFACE_vtk_module_<fragment>_install` to the install file as the non-install property name. This is generally useful for properties which change between the build and installation.

`vtk_module_build`

Build modules and kits. *module*

Once all of the modules have been scanned, they need to be built. Generally, there will be just one build necessary for a set of scans, though they may be built distinctly as well. If there are multiple calls to this function, they should generally in reverse order of their scans.

```
vtk_module_build(
  MODULES       <module>...
  [KITS          <kit>...]

  [LIBRARY_NAME_SUFFIX  <suffix>]
  [VERSION              <version>]
```

(continues on next page)

(continued from previous page)

```
  [SOVERSION            <soversion>]

  [PACKAGE              <package>]

  [BUILD_WITH_KITS  <ON|OFF>]

  [ENABLE_WRAPPING <ON|OFF>]
  [ENABLE_SERIALIZATION <ON|OFF>]

  [USE_EXTERNAL <ON|OFF>]

  [INSTALL_HEADERS              <ON|OFF>]
  [HEADERS_COMPONENT            <component>]
  [HEADERS_EXCLUDE_FROM_ALL     <ON|OFF>]
  [USE_FILE_SETS                <ON|OFF>]

  [TARGETS_COMPONENT  <component>]
  [INSTALL_EXPORT     <export>]

  [TARGET_SPECIFIC_COMPONENTS <ON|OFF>]

  [LICENSE_COMPONENT  <component>]

  [UTILITY_TARGET     <target>]

  [TEST_DIRECTORY_NAME        <name>]
  [TEST_DATA_TARGET           <target>]
  [TEST_INPUT_DATA_DIRECTORY  <directory>]
  [TEST_OUTPUT_DATA_DIRECTORY <directory>]
  [TEST_OUTPUT_DIRECTORY      <directory>]

  [GENERATE_SPDX            <ON|OFF>]
  [SPDX_COMPONENT           <component>]
  [SPDX_DOCUMENT_NAMESPACE  <uri>]
  [SPDX_DOWNLOAD_LOCATION   <url>]

  [ARCHIVE_DESTINATION    <destination>]
  [HEADERS_DESTINATION    <destination>]
  [LIBRARY_DESTINATION    <destination>]
  [RUNTIME_DESTINATION    <destination>]
  [CMAKE_DESTINATION      <destination>]
  [LICENSE_DESTINATION    <destination>]
  [HIERARCHY_DESTINATION  <destination>])
```

The only requirement of the function is the list of modules to build, the rest have reasonable defaults if not specified.

- `MODULES`: (Required) The list of modules to build.
- `KITS`: (Required if `BUILD_WITH_KITS` is `ON`) The list of kits to build.
- `LIBRARY_NAME_SUFFIX`: (Defaults to `""`) A suffix to add to library names. If it is not empty, it is prefixed with `-` to separate it from the kit name.
- `VERSION`: If specified, the `VERSION` property on built libraries will be set to this value.

- SOVERSION: If specified, the SOVERSION property on built libraries will be set to this value.
- PACKAGE: (Defaults to ${CMAKE_PROJECT_NAME}) The name the build is meant to be found as when using find_package. Note that separate builds will require distinct PACKAGE values.
- BUILD_WITH_KITS: (Defaults to OFF) If enabled, kit libraries will be built.
- ENABLE_WRAPPING: (Default depends on the existence of VTK::WrapHierarchy or VTKCompileTools::WrapHierarchy targets) If enabled, wrapping will be available to the modules built in this call.
- ENABLE_SERIALIZATION: (Defaults to OFF) If enabled, (de)serialization code will be autogenerated for classes with the correct wrapping hints.
- USE_EXTERNAL: (Defaults to OFF) Whether third party modules should find external copies rather than building their own copy.
- INSTALL_HEADERS: (Defaults to ON) Whether or not to install public headers.
- HEADERS_COMPONENT: (Defaults to development) The install component to use for header installation. Note that other SDK-related bits use the same component (e.g., CMake module files).
- HEADERS_EXCLUDE_FROM_ALL: (Defaults to OFF) Whether to install the headers component in ALL or not.
- USE_FILE_SETS: (Defaults to OFF) Whether to use FILE_SET source specification or not.
- TARGETS_COMPONENT: Defaults to ``runtime) The install component to use for the libraries built.
- TARGET_SPECIFIC_COMPONENTS: (Defaults to OFF) If ON, place artifacts into target-specific install components (<TARGET>-<COMPONENT>).
- LICENSE_COMPONENT: (Defaults to licenses) The install component to use for licenses.
- UTILITY_TARGET: If specified, all libraries and executables made by the VTK Module API will privately link to this target. This may be used to provide things such as project-wide compilation flags or similar.
- TARGET_NAMESPACE: Defaults to ``\<AUTO\>) The namespace for installed targets. All targets must have the same namespace. If set to \<AUTO\>, the namespace will be detected automatically.
- INSTALL_EXPORT: (Defaults to "") If non-empty, targets will be added to the given export. The export will also be installed as part of this build command.
- TEST_DIRECTORY_NAME: (Defaults to Testing) The name of the testing directory to look for in each module. Set to NONE to disable automatic test management.
- TEST_DATA_TARGET: (Defaults to <PACKAGE>-data) The target to add testing data download commands to.
- TEST_INPUT_DATA_DIRECTORY: (Defaults to ${CMAKE_CURRENT_SOURCE_DIR}/Data) The directory which will contain data for use by tests.
- TEST_OUTPUT_DATA_DIRECTORY: (Defaults to ${CMAKE_CURRENT_BINARY_DIR}/Data) The directory which will contain data for use by tests.
- TEST_OUTPUT_DIRECTORY: (Defaults to ${CMAKE_BINARY_DIR}/<TEST_DIRECTORY_NAME>/Temporary) The directory which tests may write any output files to.
- GENERATE_SPDX: (Defaults to OFF) Whether or not to generate and install SPDX file for each modules and third parties.
- SPDX_COMPONENT: (Defaults to spdx) The install component to use for SPDX files.
- SPDX_DOCUMENT_NAMESPACE: (Defaults to "") Document namespace to use when generating SPDX files.
- SPDX_DOWNLOAD_LOCATION: (Defaults to "") Download location to use when generating SPDX files.

The remaining arguments control where to install files related to the build. See CMake documentation for the difference between ARCHIVE, LIBRARY, and RUNTIME.

- ARCHIVE_DESTINATION: (Defaults to ${CMAKE_INSTALL_LIBDIR}) The install destination for archive files.
- HEADERS_DESTINATION: (Defaults to ${CMAKE_INSTALL_INCLUDEDIR}) The install destination for header files.
- LIBRARY_DESTINATION: (Defaults to ${CMAKE_INSTALL_LIBDIR}) The install destination for library files.
- RUNTIME_DESTINATION: (Defaults to ${CMAKE_INSTALL_BINDIR}) The install destination for runtime files.
- CMAKE_DESTINATION: (Defaults to <LIBRARY_DESTINATION>/cmake/<PACKAGE>) The install destination for CMake files.
- LICENSE_DESTINATION: (Defaults to ${CMAKE_INSTALL_DATAROOTDIR}/licenses/${CMAKE_PROJECT_NAME}) The install destination for license files.
- SPDX_DESTINATION: (Defaults to ${CMAKE_INSTALL_DATAROOTDIR}/doc/${CMAKE_PROJECT_NAME}/spdx/) The install destination for SPDX files.
- HIERARCHY_DESTINATION: (Defaults to <LIBRARY_DESTINATION>/vtk/hierarchy/<PACKAGE>) The install destination for hierarchy files (used for language wrapping).

_vtk_module_standard_includes

Add "standard" include directories to a module. *module-impl*

Add the "standard" includes for a module to its interface. These are the source And build directories for the module itself. They are always either PUBLIC or INTERFACE (depending on the module's target type).

```
_vtk_module_standard_includes(
  [SYSTEM]
  [INTERFACE]
  TARGET                <target>
  [HEADERS_DESTINATION  <destination>])
```

_vtk_module_default_library_name

Determine the default export macro for a module. *module-impl*

Determines the export macro to be used for a module from its metadata. Assumes it is called from within a vtk_module_build call().

```
_vtk_module_default_library_name(<varname>)
```

Autoinit

module

When a module contains a factory which may be populated by other modules, these factories need to be populated when the modules are loaded by the dynamic linker (for shared builds) or program load time (for static builds). To provide for this, the module system contains an autoinit "subsystem".

Leveraging the autoinit subsystem

The subsystem provides the following hooks for use by projects:

- In modules which `IMPLEMENTS` other modules, in the generated `<module>Module.h` header (which provides export symbols as well) will include the modules which are implemented.
- In modules which are `IMPLEMENTABLE` or `IMPLEMENTS` another module, the generated `<module>Module.h` file will include the following block:

```
#ifdef <module>_AUTOINIT_INCLUDE
#include <module>_AUTOINIT_INCLUDE
#endif
#ifdef <module>_AUTOINIT
#include <header>
VTK_MODULE_AUTOINIT(<module>)
#endif
```

The *vtk_module_autoinit()* function will generate an include file and provide its path via the `<module>_AUTOINIT_INCLUDE` define. once it has been included, if the `<module>_AUTOINIT` symbol is defined, a header is included which is intended to provide the `VTK_MODULE_AUTOINIT` macro. This macro is given the module name and should use `<module>_AUTOINIT` to fill in the factories in the module with those from the `IMPLEMENTS` modules listed in that symbol.

The `<module>_AUTOINIT` symbol's value is:

```
<count>(<module1>,<module2>,<module3>)
```

where `<count>` is the number of modules in the parentheses and each module listed need to register something to `<module>`.

If not provided via the `AUTOINIT_INCLUDE` argument to the *vtk_module_add_module()* function, the header to use is fetched from the `_vtk_module_autoinit_include` global property. This only needs to be managed in modules that `IMPLEMENTS` or are `IMPLEMENTABLE`. This should be provided by projects using the module system at its lowest level. Projects not implementing the `VTK_MODULE_AUTOINIT` macro should have its value provided by `find_package` dependencies in some way.

vtk_module_autoinit

Linking to autoinit-using modules. *module*

When linking to modules, in order for the autoinit system to work, modules need to declare their registration. In order to do this, defines may need to be provided to targets in order to trigger registration. These defines may be added to targets by using this function.

```
vtk_module_autoinit(
  TARGETS <target>...
  MODULES <module>...)
```

After this call, the targets given to the `TARGETS` argument will gain the preprocessor definitions to trigger registrations properly.

_vtk_module_depfile_args

Compute supported depfile tracking arguments. *module-internal*

Support for `add_custom_command(DEPFILE)` has changed over the CMake timeline. Generate the required arguments as supported for the current CMake version and generator.

```
_vtk_module_depfile_args(
  [MULTI_CONFIG_NEEDS_GENEX]
  TOOL_ARGS <variable>
  CUSTOM_COMMAND_ARGS <variable>
  DEPFILE_PATH <path>
  [SOURCE <path>]
  [SOURCE_LANGUAGE <lang>]
  [DEPFILE_NO_GENEX_PATH <path>]
  [TOOL_FLAGS <flag>...])
```

The arguments to pass to the tool are returned in the variable given to `TOOL_ARGS` while the arguments for `add_custom_command` itself are returned in the variable given to `CUSTOM_COMMAND_ARGS`. `DEPFILE_PATH` is the path to the depfile to use. If a generator expression can optionally be used, `DEPFILE_NO_GENEX_PATH` can be specified as a fallback in case of no generator expression support (unless `MULTI_CONFIG_NEEDS_GENEX` is specified and a multi-config generator is used). `TOOL_FLAGS` specifies the flags the tool needs to specify the depfile if used. If support is not available, the path given to `SOURCE` is used for `IMPLICIT_DEPENDS` using `SOURCE_LANGUAGE` (which defaults to `CXX`).

`_vtk_module_write_wrap_hierarchy`

Generate the hierarchy for a module. *module-impl*

Write wrap hierarchy files for the module currently being built. This also installs the hierarchy file for use by dependent projects if `INSTALL_HEADERS` is set. This function honors the `HEADERS_COMPONENT`, and `HEADERS_EXCLUDE_FROM_ALL` arguments to *`vtk_module_build()`*.

```
_vtk_module_write_wrap_hierarchy()
```

`_vtk_module_add_file_set`

Add a file set to a target. *module-internal*

```
_vtk_module_add_file_set(<target>
  NAME <name>
  [VIS <visibility>]
  [TYPE <type>]
  [BASE_DIRS <base directory>...]
  FILES
    [members...])
```

Add a file set to the `<target>` named `<name>`.

- `NAME`: The name of the file set.
- `VIS`: The visibility of the file set. Defaults to `PRIVATE`. Must be a valid CMake visibility (`PUBLIC`, `PRIVATE`, or `INTERFACE`).
- `TYPE`: The type of the file set. Defaults to `HEADERS`. File sets types that are recognized and known to not be supported by the CMake version in use will be added as `PRIVATE` sources not part of any file set.
- `BASE_DIRS`: Base directories for the files. Defaults to `${CMAKE_CURRENT_SOURCE_DIR}` and `${CMAKE_CURRENT_BINARY_DIR}` if not specified.
- `FILES`: The paths to add to the file set.

Note that prior to CMake 3.19, usage of `FILE_SET` with `INTERFACE` targets is severely restricted and instead this function will do nothing. Any `PUBLIC` files specified this way need installed using standard mechanisms.

`vtk_module_add_module`

Create a module library. *module*

```
vtk_module_add_module(<name>
  [NO_INSTALL] [FORCE_STATIC|FORCE_SHARED]
  [HEADER_ONLY] [HEADER_DIRECTORIES]
  [EXPORT_MACRO_PREFIX      <prefix>]
  [HEADERS_SUBDIR           <subdir>]
  [LIBRARY_NAME_SUFFIX      <suffix>]
  [CLASSES                  <class>...]
  [TEMPLATE_CLASSES         <template class>...]
  [NOWRAP_CLASSES           <nowrap class>...]
  [NOWRAP_TEMPLATE_CLASSES  <nowrap template class>...]
  [SOURCES                  <source>...]
  [HEADERS                  <header>...]
  [NOWRAP_HEADERS           <header>...]
  [TEMPLATES                <template>...]
  [PRIVATE_CLASSES          <class>...]
  [PRIVATE_TEMPLATE_CLASSES <template class>...]
  [PRIVATE_HEADERS          <header>...]
  [PRIVATE_TEMPLATES        <template>...]
  [SPDX_SKIP_REGEX          <regex>])
```

The PRIVATE_ arguments are analogous to their non-PRIVATE_ arguments, but the associated files are not installed or available for wrapping (SOURCES are always private, so there is no PRIVATE_ variant for that argument).

- NO_INSTALL: Skip installation of the module and all its installation artifacts. Note that if this target is used by any other target that is exported, this option may not be used because CMake (in addition to VTK module APIs such as vtk_module_export_find_packages and ‘ ‘) will generate references to the target that are expected to be satisfied. It is highly recommended to test that the build and install exports (as used) be tested to make sure that the module is not actually referenced.
- FORCE_STATIC or FORCE_SHARED: For a static (respectively, shared) library to be created. If neither is provided, BUILD_SHARED_LIBS will control the library type.
- HEADER_ONLY: The module only contains headers (or templates) and contains no compilation steps. Mutually exclusive with FORCE_STATIC.
- HEADER_DIRECTORIES: The headers for this module are in a directory structure which should be preserved in the install tree.
- EXPORT_MACRO_PREFIX: The prefix for the export macro definitions. Defaults to the library name of the module in all uppercase.
- HEADERS_SUBDIR: The subdirectory to install headers into in the install tree.
- LIBRARY_NAME_SUFFIX: The suffix to the module's library name if additional information is required.
- CLASSES: A list of classes in the module. This is a shortcut for adding <class>.cxx to SOURCES and <class>.h to HEADERS.
- TEMPLATE_CLASSES: A list of template classes in the module. This is a shortcut for adding <class>.txx to TEMPLATES and <class>.h to HEADERS.
- SOURCES: A list of source files which require compilation.
- HEADERS: A list of header files which will be available for wrapping and installed.
- NOWRAP_CLASSES: A list of classes which will not be available for wrapping but installed. This is a shortcut for adding <class>.cxx to SOURCES and <class>.h to NOWRAP_HEADERS.
- NOWRAP_TEMPLATE_CLASSES: A list of template classes which will not be

- available for wrapping but installed. This is a shortcut for adding `<class>.txx` to `TEMPLATES` and `<class>.h` to `NOWRAP_HEADERS`.
- `NOWRAP_HEADERS`: A list of header files which will not be available for wrapping but installed.
- `TEMPLATES`: A list of template files which will be installed.
- **`SPDX_SKIP_REGEX`: A python regex to skip a file based on its name**
  when parsing for SPDX headers.

`_vtk_module_add_header_tests`

Add header tests for a module. *module-impl*

Todo: Move this function out to be VTK-specific, probably into *vtkModuleTesting.cmake*. Each module would then need to manually call this function. It currently assumes it is in VTK itself.

```
_vtk_module_add_header_tests()
```

`_vtk_module_apply_properties`

Apply properties to a module. *module-internal*

Apply build properties to a target. Generally only useful to wrapping code or other modules that cannot use *`vtk_module_add_module()`* for some reason.

```
_vtk_module_apply_properties(<target>
  [BASENAME <basename>])
```

If `BASENAME` is given, it will be used instead of the target name as the basis for `OUTPUT_NAME`. Full modules (as opposed to third party or other non-module libraries) always use the module's `LIBRARY_NAME` setting.

The following target properties are set based on the arguments to the calling `vtk_module_build call()`

- `OUTPUT_NAME` (based on the module's `LIBRARY_NAME` and `vtk_module_build(LIBRARY_NAME_SUFFIX)`)
- `VERSION` (based on `vtk_module_build(VERSION)`)
- `SOVERSION` (based on `vtk_module_build(SOVERSION)`)
- `DEBUG_POSTFIX` (on Windows unless already set via `CMAKE_DEBUG_POSTFIX`)

`_vtk_module_install`

Install a module target. *module-internal*

Install a target within the module context. Generally only useful to wrapping code, modules that cannot use *`vtk_module_add_module()`* for some reason, or modules which create utility targets that need installed.

```
_vtk_module_install(<target>)
```

This function uses the various installation options to *`vtk_module_build()`* function to keep the install uniform.

`vtk_module_add_executable`

Create a module executable. *module*

Some modules may have associated executables with them. By using this function, the target will be installed following the options given to the associated *`vtk_module_build()`* command. Its name will also be changed according to the `LIBRARY_NAME_SUFFIX` option.

```
vtk_module_add_executable(<name>
  [NO_INSTALL]
  [DEVELOPMENT]
  [BASENAME <basename>]
  <source>...)
```

If NO_INSTALL is specified, the executable will not be installed. If BASENAME is given, it will be used as the name of the executable rather than the target name.

If DEVELOPMENT is given, it marks the executable as a development tool and will not be installed if INSTALL_HEADERS is not set for the associated *vtk_module_build()* command.

If the executable being built is the module, its module properties are used rather than BASENAME. In addition, the dependencies of the module will be linked.

vtk_module_find_package

Find a package. *module*

A wrapper around find_package that records information for use so that the same targets may be found when finding this package.

Modules may need to find external dependencies. CMake often provides modules to find these dependencies, but when imported targets are involved, these.need to also be found from dependencies of the current project. Since the benefits of imported targets greatly outweighs not using them, it is preferred to use them.

The module system provides the *vtk_module_find_package()* function in order to extend find_package support to include finding the dependencies from an install of the project.

```
vtk_module_find_package(
  [PRIVATE] [PRIVATE_IF_SHARED] [CONFIG_MODE]
  PACKAGE               <package>
  [VERSION              <version>]
  [COMPONENTS           <component>...]
  [OPTIONAL_COMPONENTS  <component>...]
  [FORWARD_VERSION_REQ  <MAJOR|MINOR|PATCH|EXACT>]
  [VERSION_VAR          <variable>])
```

- PACKAGE: The name of the package to find.
- VERSION: The minimum version of the package that is required.
- COMPONENTS: Components of the package which are required.
- OPTIONAL_COMPONENTS: Components of the package which may be missing.
- FORWARD_VERSION_REQ: If provided, the found version will be promoted to the minimum version required matching the given version scheme.
- VERSION_VAR: The variable to use as the provided version (defaults to <PACKAGE>_VERSION). It may contain @ in which case it will be configured. This is useful for modules which only provide components of the actual version number.
- CONFIG_MODE: If present, pass CONFIG to the underlying find_package call.
- PRIVATE: The dependency should not be exported to the install.
- PRIVATE_IF_SHARED: The dependency should not be exported to the install if the module is built as a SHARED library.

The PACKAGE argument is the only required argument. The rest are optional.

Note that PRIVATE is *only* applicable for private dependencies on interface targets (basically, header libraries) because some platforms require private shared libraries dependencies to be present when linking dependent libraries and executables as well. Such usages should additionally be used only via a $<BUILD_INTERFACE> generator expression to avoid putting the target name into the install tree at all.

vtk_module_export_find_packages

Export find_package calls for dependencies. *module*

When installing a project that is meant to be found via find_package from CMake, using imported targets in the build means that imported targets need to be created during the find_package as well. This function writes a file suitable for inclusion from a <package>-config.cmake file to satisfy dependencies. It assumes that the exported targets are named ${CMAKE_FIND_PACKAGE_NAME}::${component}. Dependent packages will only be found if a requested component requires the package to be found either directly or transitively.

```
vtk_module_export_find_packages(
  CMAKE_DESTINATION <directory>
  FILE_NAME         <filename>
  [COMPONENT        <component>]
  MODULES           <module>...)
```

The file will be named according to the FILE_NAME argument will be installed into CMAKE_DESTINATION in the build and install trees with the given filename. If not provided, the development component will be used.

The vtk_module_find_package calls made by the modules listed in MODULES will be exported to this file.

Third party support

module

The module system acknowledges that third party support is a pain and offers APIs to help wrangle them. Sometimes third party code needs a shim introduced to make it behave better, so an INTERFACE library to add that in is very useful. Other times, third party code is hard to ensure that it exists everywhere, so it is bundled. When that happens, the ability to select between the bundled copy and an external copy is useful. All three (and more) of these are possible.

The following functions are used to handle third party modules:

- *vtk_module_third_party()*
- *vtk_module_third_party_external()*
- *vtk_module_third_party_internal()*

vtk_module_third_party

Third party module.|module|

When a project has modules which represent third party packages, there are some convenience functions to help deal with them. First, there is the meta-wrapper:

```
vtk_module_third_party(
  [INTERNAL <internal arguments>...]
  [EXTERNAL <external arguments>...])
```

This offers a cache variable named VTK_MODULE_USE_EXTERNAL_<module name> that may be set to trigger between the internal copy and an externally provided copy. This is available as a local variable named VTK_MODULE_USE_EXTERNAL_<library name>. See the *vtk_module_third_party_external()* and :cmake:command`vtk_module_third_party_internal` functions for the arguments supported by the EXTERNAL and INTERNAL arguments, respectively.

_vtk_module_mark_third_party

Mark a module as being third party. *module-impl*

Mark a module as being a third party module.

```
_vtk_module_mark_third_party(<target>)
```

vtk_module_third_party_external

External third party package. *module*

A third party dependency may be expressed as a module using this function. Third party packages are found using CMake's `find_package` function. It is highly recommended that imported targets are used to make usage easier. The module itself will be created as an `INTERFACE` library which exposes the package.

```
vtk_module_third_party_external(
  PACKAGE               <package>
  [VERSION              <version>]
  [COMPONENTS           <component>...]
  [OPTIONAL_COMPONENTS  <component>...]
  [TARGETS              <target>...]
  [INCLUDE_DIRS <path-or-variable>...]
  [LIBRARIES    <target-or-variable>...]
  [DEFINITIONS  <variable>...]
  [FORWARD_VERSION_REQ  <MAJOR|MINOR|PATCH|EXACT>]
  [VERSION_VAR          <version-spec>]
  [USE_VARIABLES        <variable>...]
  [CONFIG_MODE]
  [STANDARD_INCLUDE_DIRS])
```

Only the `PACKAGE` argument is required. The arguments are as follows:

- `PACKAGE`: (Required) The name of the package to find.
- `VERSION`: If specified, the minimum version of the dependency that must be found.
- `COMPONENTS`: The list of components to request from the package.
- `OPTIONAL_COMPONENTS`: The list of optional components to request from the package.
- `TARGETS`: The list of targets to search for when using this package. Targets which do not exist will be ignored to support different versions of a package using different target names.
- `STANDARD_INCLUDE_DIRS`: If present, standard include directories will be added to the module target. This is usually only required if both internal and external are supported for a given dependency.
- `INCLUDE_DIRS`: If specified, this is added as a `SYSTEM INTERFACE` include directory for the target. If a variable name is given, it will be dereferenced.
- `LIBRARIES`: The libraries to link from the package. If a variable name is given, it will be dereferenced, however a warning that imported targets are not being used will be emitted.
- `DEFINITIONS`: If specified, the given variables will be added to the target compile definitions interface.
- `CONFIG_MODE`: Force `CONFIG` mode.
- `FORWARD_VERSION_REQ` and `VERSION_VAR`: See documentation for *`vtk_module_find_package()`*.
- `USE_VARIABLES`: List of variables from the `find_package` to make available to the caller.

vtk_module_third_party_internal

Internal third party package. *module*

Third party modules may also be bundled with the project itself. In this case, it is an internal third party dependency. The dependency is assumed to be in a subdirectory that will be used via `add_subdirectory`. Unless it is marked as `HEADERS_ONLY`, it is assumed that it will create a target with the name of the module.

SPDX generation requires that `SPDX_LICENSE_IDENTIFIER` and `SPDX_COPYRIGHT_TEXT` are specified.

```
vtk_module_third_party_internal(
  [SUBDIRECTORY   <path>]
  [HEADERS_SUBDIR <subdir>]
  [LICENSE_FILES  <file>...]
  [VERSION        <version>]
  [HEADER_ONLY]
  [INTERFACE]
  [STANDARD_INCLUDE_DIRS])
```

All arguments are optional, however warnings are emitted if `LICENSE_FILES`, `VERSION`, `SPDX_LICENSE_IDENTIFIER` or `SPDX_COPYRIGHT_TEXT` are not specified.

They are as follows:

- `SUBDIRECTORY`: (Defaults to the library name of the module) The subdirectory containing the `CMakeLists.txt` for the dependency.
- `HEADERS_SUBDIR`: If non-empty, the subdirectory to use for installing headers.
- `LICENSE_FILES`: A list of license files to install for the dependency. If not given, a warning will be emitted.
- `SPDX_LICENSE_IDENTIFIER`: A license identifier for SPDX file generation
- `SPDX_DOWNLOAD_LOCATION`: A download location for SPDX file generation
- `SPDX_COPYRIGHT_TEXT`: A copyright text for SPDX file generation
- `SPDX_CUSTOM_LICENSE_FILE`: A relative path to a single custom license file to include in generated SPDX file.
- `SPDX_CUSTOM_LICENSE_NAME`: The name of the single custom license, without the `LicenseRef-`
- `VERSION`: The version of the library that is included.
- `HEADER_ONLY`: The dependency is header only and will not create a target.
- `INTERFACE`: The dependency is an `INTERFACE` library.
- `STANDARD_INCLUDE_DIRS`: If present, module-standard include directories will be added to the module target.

_vtk_module_generate_spdx

SPDX file generation at build time. *module-internal*

Modules can specify a copyright and a license identifier as well as other information to generate a SPDX file in order to provide a Software Bill Of Materials (SBOM). Inputs files can be parsed for SPDX copyrights and license identifier to add to the SPDX file as well.

```
_vtk_module_generate_spd(
  [MODULE_NAME <name>]
  [TARGET      <target>]
  [OUTPUT      <file>]
```

(continues on next page)

(continued from previous page)

```
  [SKIP_REGEX  <regex>]
  [INPUT_FILES <file>...]
```

All arguments are required except for `INPUT_FILES`.

- `MODULE_NAME`: The name of the module that will be used as package name in the SPDX file.
- `TARGET`: A CMake target for the generation of the SPDX file at build time
- `OUTPUT`: Path to the SPDX file to generate
- `SKIP_REGEX`: A python regex to exclude certain source files from SPDX parsing
- `INPUT_FILES`: A list of input files to parse for SPDX copyrights and license identifiers, some files are automatically excluded from parsing.

8.3.3 vtkModuleTesting

VTK uses the ExternalData CMake module to handle the data management for its test suite. Test data is only downloaded when a test which requires it is enabled and it is cached so that every build does not need to redownload the same data.

To facilitate this workflow, there are a number of CMake functions available in order to indicate that test data is required.

Loading data

`vtk_module_test_data`

Download test data. *module*

Data may be downloaded manually using this function:

```
vtk_module_test_data(<PATHSPEC>...)
```

This will download data inside of the input data directory for the modules being built at that time (see the `TEST_INPUT_DATA_DIRECTORY` argument of `vtk_module_build`).

For supported *PATHSPEC* syntax, see the associated documentation in ref:*ExternalData*. These arguments are already wrapped in the `DATA{}` syntax and are assumed to be relative paths from the input data directory.

Creating test executables

`vtk_module_test_executable`

This function creates an executable from the list of sources passed to it. It is automatically linked to the module the tests are intended for as well as any declared test dependencies of the module.

```
vtk_module_test_executable(<NAME> <SOURCE>...)
```

This function is not usually used directly, but instead through the other convenience functions.

Test name parsing

Test names default to using the basename of the filename which contains the test. Two tests may share the same file by prefixing with a custom name for the test and a comma.

The two parsed syntaxes are: - `CustomTestName,TestFile` - `TestFile`

Note that `TestFile` should already have had its extension stripped (usually done by `_vtk_test_parse_args`).

In general, the name of a test will be `<EXENAME>-<TESTNAME>`, however, by setting `vtk_test_prefix`, the test name will instead be `<EXENAME>-<PREFIX><TESTNAME>`.

Test function arguments

Each test is specified using one of the two following syntaxes

- `<NAME>.<SOURCE_EXT>`
- `<NAME>.<SOURCE_EXT>,<OPTIONS>`

Where `NAME` is a valid test name. If present, the specified `OPTIONS` are only for the associated test. The expected extension is specified by the associated test function.

`_vtk_test_parse_args`

module-internal Given a list of valid "options", this function will parse out a the following variables:

- `args`: Unrecognized arguments. These should be interpreted as arguments that should be passed on the command line to all tests in this parse group.
- `options`: Options specified globally (for all tests in this group).
- `names`: A list containing all named tests. These should be parsed by `_vtk_test_parse_name`.
- `_<NAME>_options`: Options specific to a certain test.

```
_vtk_test_parse_args(<OPTIONS> <SOURCE_EXT> <ARG>...)
```

In order to be recognized as a source file, the `SOURCE_EXT` must be used. Without it, all non-option arguments are placed into `args`. Each test is parsed out matching these:

`_vtk_test_set_options`

For handling global option settings *module-internal*, this function sets variables in the calling scoped named *<PREFIX><OPTION>* to either *0* or *1* if the option is present in the remaining argument list.

```
_vtk_test_set_options(<OPTIONS> <PREFIX> <ARG>...)
```

Additionally, a non-*0* default for a given option may be specified by a variable with the same name as the option and specifying a prefix for the output variables.

C++ tests

`vtk_add_test_cxx`

This function declares C++ tests *module*. Source files are required to use the *cxx* extension.

```
vtk_add_test_cxx(<EXENAME> <VARNAME> <ARG>...)
```

Each argument should be either an option, a test specification, or it is passed as flags to all tests declared in the group. The list of tests is set in the `<VARNAME>` variable in the calling scope.

Options:

- `NO_DATA`: The test does not need to know the test input data directory. If it does, it is passed on the command line via the `-D` flag.
- `NO_VALID`: The test does not have a valid baseline image. If it does, the baseline is assumed to be in `../Data/Baseline/<NAME>.png` relative to the current source directory. If alternate baseline images are required, `<NAME>` may be suffixed by `_1`, `_2`, etc. The valid image is passed via the `-V` flag. - `TIGHT_VALID`: Uses euclidian type metrics to compare baselines. Baseline comparison is sensitive to outliers in this setting. - `LOOSE_VALID`: Uses L1 type metrics to compare baselines. Baseline comparison is somewhat more forgiving. Typical use cases involve rendering that is highly GPU dependent, and baselines with text. - `LEGACY_VALID`: Uses legacy image compare. This metric generates a lot of false negatives. It is recommended not to use it.
- `NO_OUTPUT`: The test does not need to write out any data to the filesystem. If it does, a directory which may be written to is passed via the `-T` flag.

Additional flags may be passed to tests using the `${_vtk_build_test}_ARGS` variable or the `<NAME>_ARGS` variable.

MPI tests

`vtk_add_test_mpi`

This function declares C++ tests which should be run under an MPI environment. *module* Source files are required to use the *cxx* extension.

```
vtk_add_test_mpi(<EXENAME> <VARNAME> <ARG>...)
```

Each argument should be either an option, a test specification, or it is passed as flags to all tests declared in the group. The list of tests is set in the `<VARNAME>` variable in the calling scope.

Options:

- `TESTING_DATA`
- `NO_VALID`: The test does not have a valid baseline image. If it does, the baseline is assumed to be in `../Data/Baseline/<NAME>.png` relative to the current source directory. If alternate baseline images are required, `<NAME>` may be suffixed by `_1`, `_2`, etc. The valid image is passed via the `-V` flag.

Each test is run using the number of processors specified by the following variables (using the first one which is set):

- `<NAME>_NUMPROCS`
- `<EXENAME>_NUMPROCS`
- `VTK_MPI_NUMPROCS` (defaults to 2)

Additional flags may be passed to tests using the `${_vtk_build_test}_ARGS` variable or the *<NAME>_ARGS* variable.

C++ test executable

`vtk_test_cxx_executable`

```
vtk_test_cxx_executable(<EXENAME> <VARNAME> [RENDERING_FACTORY] [<SRC>...])
```

Creates an executable named `EXENAME` which contains the tests listed in the variable named in the `VARNAME` argument. The `EXENAME` must match the `EXENAME` passed to the test declarations when building the list of tests.

If `RENDERING_FACTORY` is provided, VTK's rendering factories are initialized during the test.

By default, VTK's rendering tests enable FP exceptions to find floating point errors in debug builds. If `DISABLE_FLOATING_POINT_EXCEPTIONS` is provided, FP exceptions are not enabled for the test. This is useful when testing against external libraries to ignore exceptions in third-party code.

Any additional arguments are added as additional sources for the executable.

`vtk_test_mpi_executable`

MPI executables used to have their own test executable function.|module-internal| This is no longer necessary and is deprecated. Instead, *vtk_test_cxx_executable* should be used instead.

Python tests

`vtk_add_test_python`

This function declares Python tests.|module| Test files are required to use the *py* extension.

```
vtk_add_test_python(<EXENAME> <VARNAME> <ARG>...)
```

If the `_vtk_testing_python_exe` variable is not set, the `vtkpython` binary is used by default. Additional arguments may be passed in this variable as well.

If the given Python executable supports VTK's `--enable-bt` flag, setting `_vtk_testing_python_exe_supports_bt` to 1 is required to enable it.

JavaScript tests

`vtk_add_test_module_javascript_node`

This function declares JavaScript tests run with NodeJS. Test files are required to use the *mjs* extension. Additional arguments to *node* can be passed via *_vtk_node_args* variable.

```
vtk_add_test_module_javascript_node(<VARNAME> <ARG>...)
```

The `_vtk_testing_nodejs_exe` variable must point to the path of a *node* interpreter.

MPI tests

`vtk_add_test_python_mpi`

A small wrapper around `vtk_add_test_python` which adds support for running MPI-aware tests written in Python.

The `$<module library name>_NUMPROCS` variable may be used to use a non-default number of processors for a test.

This forces running with the `pvtkpython` executable.

ABI Mangling tests

vtk_add_test_mangling

This function declares a test to verify that all of the exported symbols in the VTK module library contain the correct ABI mangling prefix. This test requires setting the option VTK_ABI_NAMESPACE_NAME to a value that is not "<DEFAULT>".

Current limitations of this test are: - Does not run on non-UNIX platforms - Is not compatible with the option "VTK_ENABLE_KITS" - May not work outside of VTK itself

```
vtk_add_test_mangling(module_name [EXEMPTIONS ...])
```

Options: - `EXEMPTIONS`: List of symbol patterns to excluded from the ABI mangling test

where it is known that the symbols do not support the ABI mangling but are still exported. This option should be extremely rare to use, see the documentation on ABI mangling for how the handle C and C++ symbols before adding an EXEMPTION.

8.3.4 vtkModuleWrapPython

APIs for wrapping modules for Python

Limitations

Known limitations include:

- Shared Python modules only really support shared builds of modules. VTK does not provide mangling facilities for itself, so statically linking VTK into its Python modules precludes using VTK's C++ interface anywhere else within the Python environment.
- Only supports CPython. Other implementations are not supported by the *VTK::WrapPython* executable.
- Links directly to a Python library. See the *VTK::Python* module for more details.

vtk_module_python_default_destination

Determine Python module destination. *module-wrapping-python*

Some projects may need to know where Python expects its modules to be placed in the install tree (assuming a shared prefix). This function computes the default and sets the passed variable to the value in the calling scope.

```
vtk_module_python_default_destination(<var>
 [MAJOR_VERSION <major>])
```

By default, the destination is `${CMAKE_INSTALL_BINDIR}/Lib/site-packages` on Windows and `${CMAKE_INSTALL_LIBDIR}/python<VERSION>/site-packages` otherwise.

`<MAJOR_VERSION>`, if specified, must be `3`.

_vtk_module_wrap_python_sources

Generate sources for using a module's classes from Python. *module-impl*

This function generates the wrapped sources for a module. It places the list of generated source files and classes in variables named in the second and third arguments, respectively.

```
_vtk_module_wrap_python_sources(<module> <sources> <classes>)
```

_vtk_module_wrap_python_library

Generate a CPython library for a set of modules. *module-impl*

A Python module library may consist of the Python wrappings of multiple modules. This is useful for kit-based builds where the modules part of the same kit belong to the same Python module as well.

```
_vtk_module_wrap_python_library(<name> <module>...)
```

The first argument is the name of the Python module. The remaining arguments are modules to include in the Python module.

The remaining information it uses is assumed to be provided by the `vtk_module_wrap_python function()`.

vtk_module_wrap_python

Wrap a set of modules for use in Python.|module-wrapping-python|

```
vtk_module_wrap_python(
  MODULES <module>...
  [TARGET <target>]
  [WRAPPED_MODULES <varname>]

  [BUILD_STATIC <ON|OFF>]
  [INSTALL_HEADERS <ON|OFF>]
  [BUILD_PYI_FILES <ON|OFF>]

  [DEPENDS <target>...]
  [UTILITY_TARGET <target>]

  [MODULE_DESTINATION <destination>]
  [STATIC_MODULE_DESTINATION <destination>]
  [CMAKE_DESTINATION <destination>]
  [LIBRARY_DESTINATION <destination>]
  [HEADERS_DESTINATION <destination>]

  [PYTHON_PACKAGE <package>]
  [SOABI <soabi>]
  [USE_DEBUG_SUFFIX <ON|OFF>]
  [REPLACE_DEBUG_SUFFIX <ON|OFF>]

  [INTERPRETER <interpreter>]

  [INSTALL_EXPORT <export>]
  [COMPONENT <component>])
  [TARGET_SPECIFIC_COMPONENTS <ON|OFF>]

  [WARNINGS <warning>...]
)
```

- `MODULES`: (Required) The list of modules to wrap.
- `TARGET`: (Recommended) The target to create which represents all wrapped Python modules. This is mostly useful when supporting static Python modules in order to add the generated modules to the built-in table.
- `WRAPPED_MODULES`: (Recommended) Not all modules are wrappable. This variable will be set to contain the list of modules which were wrapped. These modules will have a

INTERFACE_vtk_module_python_package property set on them which is the name that should be given to import statements in Python code.

- BUILD_STATIC: Defaults to ${BUILD_SHARED_LIBS}. Note that shared modules with a static build is not completely supported. For static Python module builds, a header named <TARGET>.h will be available with a function void <TARGET>_load() which will add all Python modules created by this call to the imported module table. For shared Python module builds, the same function is provided, but it is a no-op.
- INSTALL_HEADERS (Defaults to ON): If unset, CMake properties will not be installed.
- BUILD_PYI_FILES (Defaults to OFF): If set, .pyi files will be built and installed for the generated modules.
- TARGET_SPECIFIC_COMPONENTS (Defaults to OFF): If set, prepend the output target name to the install component (<TARGET>-<COMPONENT>).
- DEPENDS: This is list of other Python modules targets i.e. targets generated from previous calls to vtk_module_wrap_python that this new target depends on. This is used when BUILD_STATIC is true to ensure that the void <TARGET>_load() is correctly called for each of the dependencies.
- UTILITY_TARGET: If specified, all libraries made by the Python wrapping will link privately to this target. This may be used to add compile flags to the Python libraries.
- MODULE_DESTINATION: Modules will be placed in this location in the build tree. The install tree should remove $<CONFIGURATION> bits, but it currently does not. See vtk_module_python_default_destination for the default value.
- STATIC_MODULE_DESTINATION: Defaults to ${CMAKE_INSTALL_LIBDIR}. This default may change in the future since the best location for these files is not yet known. Static libraries containing Python code will be installed to the install tree under this path.
- CMAKE_DESTINATION: (Required if INSTALL_HEADERS is ON) Where to install Python-related module property CMake files.
- LIBRARY_DESTINATION (Recommended): If provided, dynamic loader information will be added to modules for loading dependent libraries.
- HEADERS_DESTINATION: Defaults to (${CMAKE_INSTALL_INCLUDEDIR}. Module loader headers will be installed to this directory.
- PYTHON_PACKAGE: (Recommended) All generated modules will be added to this Python package. The format is in Python syntax (e.g., package.subpackage).
- SOABI: (Required for wheel support): If given, generate libraries with the SOABI tag in the module filename.
- USE_DEBUG_SUFFIX (Defaults to OFF): If ON, Windows modules will have a _d suffix appended to the module name. This is intended for use with debug Python builds.
- REPLACE_DEBUG_SUFFIX (Defaults to OFF): If ON, any project-wide debug suffix will be replaced with the local debug suffix (if enabled).
- INTERPRETER (Defaults to VTK::Python or Python3::Interpreter): If provided, this interpreter will be used to run supplemental processes which involve Python scripts including .pyi file generation. If a target name is provided, its path will be used, otherwise a string which expands to the path to an interpreter executable may be provided. If the string DISABLE is given, any support using interpreters will be disabled.
- INSTALL_EXPORT: If provided, static installs will add the installed libraries to the provided export set.
- COMPONENT: Defaults to python. All install rules created by this function will use this installation component.
- WARNINGS: Warnings to enable. Supported warnings: empty.

vtk_module_add_python_package

Install Python packages with a module *module-wrapping-python*.

Some modules may have associated Python code. This function should be used to install them.

```
vtk_module_add_python_package(<module>
  PACKAGE <package>
  FILES <files>...
  [MODULE_DESTINATION <destination>]
  [COMPONENT <component>])
```

The `<module>` argument must match the associated VTK module that the package is with. Each package is independent and should be installed separately. That is, `package` and `package.subpackage` should each get their own call to this function.

- `PACKAGE`: (Required) The package installed by this call. Currently, subpackages must have their own call to this function.
- `FILES`: (Required) File paths should be relative to the source directory of the calling `CMakeLists.txt`. Upward paths are not supported (nor are checked for). Absolute paths are assumed to be in the build tree and their relative path is computed relative to the current binary directory.
- `MODULE_DESTINATION`: Modules will be placed in this location in the build tree. The install tree should remove `$<CONFIGURATION>` bits, but it currently does not. See `vtk_module_python_default_destination` for the default value.
- `COMPONENT`: Defaults to `python`. All install rules created by this function will use this installation component.

A `<module>-<package>` target is created which ensures that all Python modules have been copied to the correct location in the build tree.

Todo: Support freezing the Python package. This should create a header and the associated target should provide an interface for including this header. The target should then be exported and the header installed properly.

vtk_module_add_python_module

Use a Python package as a module. *module-wrapping-python*

If a module is a Python package, this function should be used instead of *`vtk_module_add_module()`*.

```
vtk_module_add_python_module(<name>
  PACKAGES <packages>...)
```

- `PACKAGES`: (Required) The list of packages installed by this module. These must have been created by the *`vtk_module_add_python_package()`* function.

8.3.5 vtkModuleWrapJava

APIs for wrapping modules for Java

_vtk_module_wrap_java_sources

Generate sources for using a module's classes from Java. *module-impl*

This function generates the wrapped sources for a module. It places the list of generated source files and Java source files in variables named in the second and third arguments, respectively.

```
_vtk_module_wrap_java_sources(<module> <sources> <classes>)
```

_vtk_module_wrap_java_library

Generate a JNI library for a set of modules. *module-impl*

A single JNI library may consist of the Java wrappings of multiple modules. This is useful for kit-based builds where the modules part of the same kit belong to the same JNI library as well.

```
_vtk_module_wrap_java_library(<name> <module>...)
```

The first argument is the name of the JNI library. The remaining arguments are modules to include in the JNI library.

The remaining information it uses is assumed to be provided by the `vtk_module_wrap_java()` function.

vtk_module_wrap_java

Wrap a set of modules for use in Java. *module-wrapping-java*

```
vtk_module_wrap_java(
  MODULES <module>...
  [WRAPPED_MODULES <varname>]

  [UTILITY_TARGET <target>]

  [JAVA_OUTPUT <destination>]

  [LIBRARY_DESTINATION <destination>]
  [JNILIB_DESTINATION <destination>]
  [JNILIB_COMPONENT <component>]

  [WARNINGS <warning>...])
```

- `MODULES`: (Required) The list of modules to wrap.
- `WRAPPED_MODULES`: (Recommended) Not all modules are wrappable. This variable will be set to contain the list of modules which were wrapped.
- `UTILITY_TARGET`: If specified, all libraries made by the Java wrapping will link privately to this target. This may be used to add compile flags to the Java libraries.
- `JAVA_OUTPUT`: Defaults to `${CMAKE_CURRENT_BINARY_DIR}/CMakeFiles/vtkJava`. Java source files are written to this directory. After generation, the files may be compiled as needed.
- `LIBRARY_DESTINATION` (Recommended): If provided, dynamic loader information will be added to modules for loading dependent libraries.
- `JNILIB_DESTINATION`: Where to install JNI libraries.
- `JNILIB_COMPONENT`: Defaults to `jni`. The install component to use for JNI libraries.

- `WARNINGS`: Warnings to enable. Supported warnings: `empty`.

For each wrapped module, a `<module>Java` target will be created. These targets will have a `_vtk_module_java_files` property which is the list of generated Java source files for that target.

For dependency purposes, the `<module>Java-java-sources` target may also be used.

8.3.6 vtkModuleJSON

`_vtk_json_bool`

Output a boolean to JSON. *module-impl*

Appends a condition as a JSON boolean with the given dictionary key name to the given string variable.

```
_vtk_json_bool(<output> <name> <cond>)
```

`_vtk_json_string_list`

Output a string list to JSON. *module-impl*

Appends a variable as a JSON list of strings with the given dictionary key name to the given string variable.

```
_vtk_json_string_list(<output> <name> <cond>)
```

`vtk_module_json`

JSON metadata representation of modules. *module-support*

Information about the modules built and/or available may be dumped to a JSON file.

```
vtk_module_json(
  MODULES   <module>...
  OUTPUT    <path>)
```

- `MODULES`: (Required) The modules to output information for.
- **`OUTPUT`: (Required) A JSON file describing the modules built will**
  be output to this path. Relative paths are rooted to `CMAKE_BINARY_DIR`.

Example output:

```
{
  "modules": [
    {
      "name": "...",
      "library_name": "...",
      "enabled": <bool>,
      "implementable": <bool>,
      "third_party": <bool>,
      "wrap_exclude": <bool>,
      "kit": "...",
      "depends": [
        "..."
      ],
      "optional_depends": [
        "..."
```

(continues on next page)

(continued from previous page)

```
      ],
      "private_depends": [
        "..."
      ],
      "implements": [
        "..."
      ],
      "headers": [
        "..."
      ]
    }
  ],
  "kits": [
    {
      "name": "...",
      "enabled": <bool>,
      "modules": [
      ]
    }
  ]
}
```

8.3.7 vtkModuleGraphviz

_vtk_module_graphviz_module_node

Output a node in the graph *module-impl*

Queries the properties for modules and generates the node for it in the graph and its outgoing dependency edges.

vtk_module_graphviz

Generate graphviz output for a module dependency graph. *module-support*

Information about the modules built and/or available may be dumped to a Graphviz *.dot* file.

```
vtk_module_graphviz(
  MODULES   <module>...
  OUTPUT    <path>

  [PRIVATE_DEPENDENCIES <ON|OFF>]
  [KIT_CLUSTERS <ON|OFF>])
```

- `MODULES`: (Required) The modules to output information for.
- `OUTPUT`: (Required) A Graphviz file describing the modules built will be output to this path. Relative paths are rooted to `CMAKE_BINARY_DIR`.
- `PRIVATE_DEPENDENCIES`: (Default `ON`) Whether to draw private dependency edges or not..
- `KIT_CLUSTERS`: (Default `OFF`) Whether to draw modules as part of a kit as a cluster or not.

CHAPTER

NINE

ADVANCED TOPICS

9.1 Additional Python Wheels

Python wheels for VTK are available in pypi

```
pip install vtk
```

More wheels can be accessed from the GitLab Package Registry.

To install the **latest release** wheel from the GitLab registry:

```
pip install --extra-index-url https://wheels.vtk.org vtk
```

To install the latest wheel **from master**:

```
pip install --extra-index-url https://wheels.vtk.org vtk --pre --no-cache
```

The wheels available at PyPi are built using the default rendering backend which leverages any available hardware graphics for generating the scene. There is also a OSMesa wheel variant that leverages offscreen rendering with OSMesa. This wheel is being built for both Linux and Windows at this time and bundles all of the necessary libraries into the wheel. These wheels are intended to be used by downstream projects in headless, CI-like environments or cloud application deployments, preventing the need to install any addition system packages.

To install the **OSMesa variant** from the latest release

```
pip install --extra-index-url https://wheels.vtk.org vtk-osmesa
```

For more information see here.

Note: conda-forge packages are also available and maintained by the community.

9.2 SPDX & SBOM

9.2.1 Overview

Software Bill of Materials (SBOM) are becoming increasingly important for software development, especially when it comes to supply chain security. Software Package Data Exchange (SPDX) is an open standard for communicating SBOM information that supports accurate identification of software components, explicit mapping of relationships between components, and the association of security and licensing information with each component.

To support this, each VTK module may be described by a `.spdx` file. See *examples*.

Configuring VTK with the option `VTK_GENERATE_SPDX` set to `ON` enables the *SPDX files generation* for each VTK module.

Caution: The generation of SPDX files is considered experimental and both the VTK Module system API and the `SPDXID` used in the generated files may change.

9.2.2 Frequently Asked Questions

How to update your module to generate a valid SPDX file ?

In the `vtk.module` file, make sure to specify `SPDX_LICENSE_IDENTIFIER` and `SPDX_COPYRIGHT_TEXT` as follows:

```
SPDX_LICENSE_IDENTIFIER
  BSD-3-Clause
SPDX_COPYRIGHT_TEXT
  Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
```

Then add SPDX tags on top of all source files in the module, as follows:

```
// SPDX-FileCopyrightText: Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
// SPDX-FileCopyrightText: Copyright (c) Awesome contributor
// SPDX-License-Identifier: BSD-3-Clause
```

Tip: Refer to the *limitations* section for more information on any potential issues that may arise when updating your module to generate a valid SPDX file.

How to update a third party to generate a valid SPDX file ?

In the third party `CMakeLists.txt`, make sure to specify, in the `vtk_module_third_party` call, `SPDX_LICENSE_IDENTIFIER` and `SPDX_COPYRIGHT_TEXT` as follows:

```
vtk_module_third_party(
   SPDX_LICENSE_IDENTIFIER
     "BSD-3-Clause"
   SPDX_COPYRIGHT_TEXT
     "Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen"
   SPDX_DOWNLOAD_LOCATION
     "git+https://gitlab.kitware.com/third-party/repo.git@hash_or_tag"
   [...]
```

Tip: Refer to the *limitations* section for more information on any potential issues that may arise when updating your module to generate a valid SPDX file.

How to correctly specify custom license for a module ?

In the module, provide a file containing the license. Then in `vtk.module` file, make sure to specify `SPDX_CUSTOM_LICENSE_FILE` with the path of the license file, `SPDX_CUSTOM_LICENSE_NAME` with the name of the license and `SPDX_LICENSE_IDENTIFIER` with a valid SPDX LicenseRef, as follows:

```
SPDX_LICENSE_IDENTIFIER
  LicenseRef-CustomLicense
SPDX_CUSTOM_LICENSE_FILE
  LICENSE
SPDX_CUSTOM_LICENSE_NAME
  CustomLicense
```

If needed, you can add SPDX tags on top of all source file specifically concerned by this license

```
// SPDX-FileCopyrightText: Copyright (c) Awesome contributor
// SPDX-License-Identifier: LicenseRef-CustomLicense
```

9.2.3 Examples

This section lists examples of generated SPDX files for different type of VTK modules.

VTK Module

Example of generated SPDX files for a module in VTK (once the module have been ported to the system):

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: IOPLY
DocumentNamespace: https://vtk.org/vtkIOPly
Creator: Tool: CMake
Created: 2023-05-16T16:08:29Z

##### Package: IOPLY

PackageName: IOPLY
SPDXID: SPDXRef-Package-IOPLY
PackageDownloadLocation: https://gitlab.kitware.com/vtk/vtk/-/tree/master/IO/PLY
FilesAnalyzed: true
PackageLicenseConcluded: BSD-3-Clause
PackageLicenseDeclared: BSD-3-Clause
PackageLicenseInfoFromFiles: BSD-3-Clause
PackageCopyrightText: <text>
Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
</text>

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-IOPLY
```

Example of a SPDX file generated without any information for a module that have not been ported to the system:

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: vtkFiltersVerdict
DocumentNamespace: https://vtk.org/vtkFiltersVerdict
Creator: Tool: CMake
Created: 2023-05-25T15:16:20Z

##### Package: vtkFiltersVerdict

PackageName: vtkFiltersVerdict
SPDXID: SPDXRef-Package-vtkFiltersVerdict
PackageDownloadLocation: https://gitlab.kitware.com/vtk/vtk/-/tree/master/Filters/Verdict
FilesAnalyzed: false
PackageLicenseConcluded: NOASSERTION
PackageLicenseDeclared: NOASSERTION
PackageLicenseInfoFromFiles: NOASSERTION
PackageCopyrightText: <text>
NOASSERTION
</text>

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-vtkFiltersVerdict
```

VTK ThirdParty Module

Example of a complete SPDX file for a 3rd party in VTK (once the 3rd party have been ported to the system):

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: VTK::loguru
DocumentNamespace: https://vtk.org/vtkloguru
Creator: Tool: CMake
Created: 2023-05-22T15:56:52Z

##### Package: VTK::loguru

PackageName: VTK::loguru
SPDXID: SPDXRef-Package-VTK::loguru
PackageDownloadLocation: https://github.com/Delgan/loguru
FilesAnalyzed: no
PackageLicenseConcluded: BSD-3-Clause
PackageLicenseDeclared: BSD-3-Clause
PackageLicenseInfoFromFiles: NOASSERTION
PackageCopyrightText: <text>
LOGURU Team
</text>

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-VTK::loguru
```

VTK Remote Module

Example of a complete SPDX file for a VTK module from outside of VTK (once the module have been ported to the system):

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: MyModule
DocumentNamespace: https://my-website/MyModule
Creator: Tool: CMake
Created: 2023-05-16T16:08:29Z

##### Package: MyModule

PackageName: MyModule
SPDXID: SPDXRef-Package-MyModule
PackageDownloadLocation: https://github/myorg/mymodule
FilesAnalyzed: true
PackageLicenseConcluded: BSD-3-Clause AND MIT
PackageLicenseDeclared: BSD-3-Clause
PackageLicenseInfoFromFiles: BSD-3-Clause AND MIT
PackageCopyrightText: <text>
Copyright (c) 2023 Popeye
Copyright (c) 2023 Wayne "The Dock" Sonjhon
</text>

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-MyModule
```

VTK Module with custom license

Example of a complete SPDX file for a VTK module with a custom license:

```
SPDXVersion: SPDX-2.2
DataLicense: CC0-1.0
SPDXID: SPDXRef-DOCUMENT
DocumentName: IOPLY
DocumentNamespace: https://vtk.org/vtkCustomModule
Creator: Tool: CMake
Created: 2023-05-16T16:08:29Z

##### Package: CustomModule

PackageName: CustomModule
SPDXID: SPDXRef-Package-CustomModule
PackageDownloadLocation: https://gitlab.kitware.com/vtk/vtk/-/tree/master/Custom/Module
FilesAnalyzed: true
PackageLicenseConcluded: BSD-3-Clause
PackageLicenseDeclared: BSD-3-Clause AND LicenseRef-CustomLicense
PackageLicenseInfoFromFiles: BSD-3-Clause
PackageCopyrightText: <text>
Copyright (c) Ken Martin, Will Schroeder, Bill Lorensen
```

(continues on next page)

(continued from previous page)

```
</text>

LicenseID: LicenseRef-CustomLicense
ExtractedText: <text>My License

This is a custom license that is not more restrictive
than BSD license.
</text>

Relationship: SPDXRef-DOCUMENT DESCRIBES SPDXRef-Package-IOPLY
```

9.2.4 Resources

- https://spdx.dev/
- https://en.wikipedia.org/wiki/Software_supply_chain
- https://www.linuxfoundation.org/blog/blog/spdx-its-already-in-use-for-global-software-bill-of-materials-sbom-and-supply-chai
- https://spdx.dev/specifications/
- https://spdx.dev/wp-content/uploads/sites/41/2020/08/SPDX-specification-2-2.pdf
- https://github.com/spdx/spdx-examples
- https://spdx.dev/wp-content/uploads/sites/41/2017/12/spdx_onepager.pdf

9.3 Building Python Wheels

Tip: For complete build instructions see here.

VTK also supports creating a Python wheel containing its Python wrappers for Python3. This is supported by setting the `VTK_WHEEL_BUILD` flag. This changes the build directory structure around to match that expected by wheels. Once configured, the build tree may be built as it would be normally and then the generated `setup.py` file used to create the wheel. Note that the `bdist_wheel` command requires that the `wheel` package is available (`pip install wheel`).

```
cmake -GNinja -DVTK_WHEEL_BUILD=ON -DVTK_WRAP_PYTHON=ON path/to/vtk/source
ninja
python3 setup.py bdist_wheel
```

Any modules may be turned on or off as in a normal VTK build. Certain modules add features to the generated wheel to indicate their availability. These flags are not meant to be comprehensive, but any reasonable feature flags may be added to `CMake/vtkWheelFinalization.cmake` as needed.

Note that the wheel will not include any external third party libraries in its wheel (e.g., X11, OpenGL, etc.) to avoid conflicts with systems or other wheels doing the same.

9.3.1 Modifying Version and/or Distribution Name

When generating a wheel, you can modify the distribution name and/or add a suffix to the wheel version string.

By default, the distribution name is `vtk` though you can add a suffix via the `VTK_DIST_NAME_SUFFIX` CMake variable (e.g., set `VTK_DIST_NAME_SUFFIX` to `'osmesa'` to have the distribution name be `vtk_osmesa`). An underscore (`_`) character is automatically placed between `vtk` and the value of `VTK_DIST_NAME_SUFFIX`. Please use `_` characters for further delimination in the suffix value. Example setting:

```
set(VTK_DIST_NAME_SUFFIX "osmesa" CACHE STRING "")
```

By default (outside of a CI release build), `dev0` is appended to the version of the package (e.g., `9.2.2.dev0`). This suffix can be controlled through the `VTK_VERSION_SUFFIX` CMake variable and is useful if generating multiple wheels and wanting to differentiate the build variants by the version string of the package.

```
set(VTK_VERSION_SUFFIX "dev0" CACHE STRING "")
```

9.4 Building using emscripten for WebAssembly

9.4.1 Introduction

This page describes how to build and install VTK using emscripten on any platform. These steps can be followed inside a docker container that comes with preinstalled `emsdk` such as dockcross/web-wasm. In fact, the VTK CI stage `webassembly-build` uses that container to configure and build VTK wasm.

Note: Guide created using

- VTK v9.2.6-2535-gc8cebe56fb
- dockcross/web-wasm:20230222-162287d

9.4.2 Prerequisites

For this guide, you will need the following:

1. **CMake**: CMake version 3.12 or higher and a working compiler. CMake is a tool that makes cross-platform building simple. On several systems it will probably be already installed. If it is not, please use the following instructions to install it. There are several precompiled binaries available at the CMake download page. Add CMake to your PATH environment variable if you downloaded an archive and not an installer.
2. **Emscripten SDK**: emsdk and any dependencies needed by emsdk. Emscripten is a complete compiler toolchain to WebAssembly, using LLVM, with a special focus on speed, size, and the Web platform. Please download the SDK from github.com/emscripten-core/emsdk.git. Then,
   - Install latest toolchain with `./emsdk install latest`
   - Activate the toolchain `./emsdk activate latest`
   - Run `emsdk_env.bat` or `emsdk_env.ps1` (Windows) or `source ./emsdk_env.sh` (Linux and OS X) to set up the environment for the calling terminal.

   For more detailed instructions see emsdk/README.md.

3. **VTK source-code**: If you have these then you can skip the rest of this section and proceed to *Build project*. Download VTK source for the version you want from https://vtk.org/download/ (zip or tar.gz (Do NOT download the exe - this is not the VTK library.)) You will probably want the latest one (highest version number) unless you have a specific reason to use an older one.

   Alternatively the source-code can be obtained from the repository as well. This is recommended only if you intent to make changes and contribute to VTK. Please refer to *git/develop.md* for help with `git`.

9.4.3 Build project

These instructions use a specific convention for the source, build and install directories that is appropriate when building VTK for wasm inside a docker container. Please replace these *root-directory* paths if VTK is being built outside a docker container.

Install emscripten ports (IMPORTANT!)

Emscripten relies on SDL2 to link user input events from the browser's event subsystem to native C/C++ code. If this is your initial download of the EMSDK, you'll need to build the SDL2 port. The "embuilder" script will be accessible on the path if you've successfully installed and activated the EMSDK, as outlined in the prerequisites.

```
$ embuilder build sdl2
```

Build VTK

1. Configure the project with CMake. `emcmake` tells CMake to use the `emscripten` toolchain for cross compilation.

```
cd /work/src/build
$ emcmake cmake \
  -S .. \
  -B . \
  -G "Ninja" \
  -DCMAKE_BUILD_TYPE=Release \
  -DBUILD_SHARED_LIBS:BOOL=OFF \
  -DVTK_ENABLE_LOGGING:BOOL=OFF \
  -DVTK_ENABLE_WRAPPING:BOOL=OFF \
  -DVTK_MODULE_ENABLE_VTK_RenderingLICOpenGL2:STRING=DONT_WANT
```

2. Compile.

```
$ cd /work/src/build
$ ninja
```

3. Install the project.

```
$ cd /work/src/build
$ ninja install
```

The binaries are now installed and you may use `-DVTK_DIR=/work/install/lib/cmake/vtk-9.2` to configure VTK wasm applications with CMake.

9.4.4 Verify installation

If everything went well then it should now be possible to compile and run the one of the C++ examples. Head over to Examples/Emscripten/Cxx/Cone/README.md and test the simple Cone example.

9.5 Cross-compiling for Mobile devices

Tip: For complete build instructions see here.

VTK supports mobile devices in its build. These are triggered by a top-level flag which then exposes some settings for a cross-compiled VTK that is controlled from the top-level build.

iOS builds may be enabled by setting the `VTK_IOS_BUILD` option. The following settings than affect the iOS build:

- `IOS_SIMULATOR_ARCHITECTURES`
- `IOS_DEVICE_ARCHITECTURES`
- `IOS_DEPLOYMENT_TARGET`
- `IOS_EMBED_BITCODE`

Android builds may be enabled by setting the `VTK_ANDROID_BUILD` option. The following settings affect the Android build:

- `ANDROID_NDK`
- `ANDROID_NATIVE_API_LEVEL`
- `ANDROID_ARCH_ABI`

9.6 Building documentation

This section outlines how to locally build both the user and developer guides and the C++ API documentation for VTK.

9.6.1 User and developer guides

VTK's user and developer guides are automatically built and deployed to https://docs.vtk.org every time the `master` branch is updated by leveraging the integration with the *Read the Docs* service.

To locally build the documentation:

Without VTK build tree

1. Download the VTK sources.
2. Create and activate a virtual environment.

Linux/macOS

```
cd Documentation/docs

python3 -m venv .venv
source .venv/bin/activate
```

Windows

```
cd Documentation\docs

py -m venv .venv
.\.venv\Scripts\activate
```

`py -m venv` executes venv using the latest Python interpreter you have installed. For more details, read the Python Windows launcher docs.

3. Install dependencies using pip.

Linux/macOS

```
python3 -m pip install -r requirements.txt
```

Windows

```
py -m pip install -r requirements.txt
```

4. Build the documentation as web pages.

```
make html
```

5. Open `_build/html/index.html` in a web browser.

Linux

```
xdg-open _build/html/index.html
```

macOS

```
open _build/html/index.html
```

Windows

```
start _build\html\index.html
```

With VTK build tree

Important: In order to successfully build the VTK documentation using the instructions below, you will need to install the required Python packages.

To ensure that you have the correct dependencies installed in the Python environment associated with the VTK build tree, please run `pip install -r Documentation\docs\requirements.txt` or `pip install --user -r Documentation\docs\requirements.txt`.

If updating your system installation of Python is not feasible or you prefer not to do so, we recommend following the `Without VTK build tree` approach instead.

1. Download VTK sources.
2. Configure VTK by setting the `VTK_BUILD_SPHINX_DOCUMENTATION` option to `ON`.
3. Build the `SphinxDoc` target.

Hint: Automatic build of preview documentation each time a merge request is submitted is not yet supported due to limitation of the *Read The Docs* service that does not yet support self-hosted GitLab deployment.

Solutions to address this are being discussed in https://github.com/readthedocs/readthedocs.org/issues/9464.

9.6.2 C++ API documentation

The C++ API documentation is built and uploaded to https://vtk.org/doc/nightly/html/index.html when the `master` branch is updated.

To locally build the documentation:

1. Install Doxygen
2. Download the VTK sources.
3. Configure VTK by setting the `VTK_BUILD_DOCUMENTATION` option to `ON`.
4. Build the `DoxygenDoc` target.

9.6.3 Targets

After configuring the VTK using CMake, the following targets may be used to build documentation for VTK:

- `DoxygenDoc` - build the doxygen documentation from VTK's C++ source files (`VTK_BUILD_DOCUMENTATION` needs to be `ON` for the target to exist).
- `SphinxDoc` - build the sphinx documentation for VTK. (`VTK_BUILD_SPHINX_DOCUMENTATION` needs to be `ON` for the target to exist).

9.7 Marshalling Hints

9.7.1 Classes

VTK auto generates (de)serialization code in C++ for classes annotated by the `VTK_MARSHALAUTO` wrapping hint.

On the other hand, the `VTK_MARSHALMANUAL` macro is used to indicate that a class will take part in marshalling, but it cannot trivially (de)serialize it's properties. This is because one or more of the class's properties may not have an appropriate setter/getter function that is recognized by the VTK property parser.

For such classes, a developer is expected to provide the code to serialize and deserialize the class in `vtkClassNameSerDes.cxx`. This file must satisfy three conditions:

1. It must live in the same module as `vtkClassName`.
2. It must export a function `int RegisterHandlers_vtkClassNameSerDesHelper(void*, void*)` with C linkage.
3. It must define and declare these three C++ functions:
   1. A serializer function

      ```
      nlohmann:json Serialize_vtkClassName(vtkObjectBase*, vtkSerializer*)
      ```

   2. A deserializer function

      ```
      void Deserialize_vtkClassName(const nlohmann::json&, vtkObjectBase*,␣
      ↪vtkDeserializer*)
      ```

   3. A registrar function

      ```
      int RegisterHandlers_vtkClassNameSerDesHelper(void* ser, void* deser)
      ```

      that registers:

      - a serializer function with a serializer instance
      - a deserializer function with a deserializer instance
      - a constructor of the VTK class with a deserializer instance

9.7.2 Properties

Excluding properties

You can exclude certain properties of a class by simply annotating the relevant setter/getter functions with `VTK_MARSHALEXCLUDE(reason)`, where reason is one of `VTK_MARSHAL_EXCLUDE_REASON_IS_INTERNAL` or `VTK_MARSHAL_EXCLUDE_REASON_NOT_SUPPORTED`. This reason will be printed in the generated C++ source code explaining why the property was not serialized.

9.7.3 Custom get/set functions

Some properties may not be correctly recognized by the property parser because they have different names for their get and set functions. You can override this by annotating the get function with the `VTK_MARSHALGETTER(property)` macro. Doing so will ensure that the function gets recognized as a getter for `property`. `VTK_MARSHALSETTER(property)` serves a similar purpose.

9.8 Object manager

9.8.1 Serialization

You can register objects with a `vtkObjectManager` instance and call `UpdateStatesFromObjects`, `GetState(identifier)` to obtain a serialized state of the registered objects and all their dependency objects that are serializable.

9.8.2 Deserialization

You can register a json state (stringified) with a `vtkObjectManager` instance and call `UpdateObjectsFromStates`, `GetObjectAtId(identifier)` to deserialize and retrieve the objects.

9.8.3 Blobs

All `vtkDataArray` are hashed and stored as unique blobs to prevent multiple copies of the same data within the state. The contents of a data array within a state are represented with a hash string. You can fetch and register blobs using `GetBlob` and `RegisterBlob`.

9.8.4 Dependencies

You can retrieve all dependent object identifiers using `vtkObjectManager::GetAllDependencies(identifier)`

9.9 Auto serialization

Modules which have `INCLUDE_MARSHAL` in their `vtk.module` will opt their headers into the automated code generation of (de)serializers. Only classes which are annotated by the `VTK_MARSHALAUTO` wrapping hint will have generated serialization code.

9.9.1 Automated code generation

The `vtkWrapSerDes` executable makes use of the `WrappingTools` package to automatically generate

1. A serializer function with signature `nlohmann:json Serialize_vtkClassName(vtkObjectBase*, vtkSerializer*)`
2. A deserializer function with signature `void(const nlohmann::json&, vtkObjectBase*, vtkDeserializer*)`
3. A registrar function that registers
    - the serializer function with a serializer instance

- the deserializer function with a deserializer instance
- the constructor of the VTK class with a deserializer instance
- It's signature is `int RegisterHandlers_vtkClassNameSerDes(void* ser, void* deser)`
- It more or less looks like:

```
int RegisterHandlers_vtkObjectSerDes(void* ser, void* deser)
{
  int success = 0;
  if (auto* asObjectBase = static_cast<vtkObjectBase*>(ser))
  {
    if (auto* serializer = vtkSerializer::SafeDownCast(asObjectBase))
    {
      serializer->RegisterHandler(typeid(vtkObject), Serialize_vtkObject);
      success = 1;
    }
  }
  if (auto* asObjectBase = static_cast<vtkObjectBase*>(deser))
  {
    if (auto* deserializer = vtkDeserializer::SafeDownCast(asObjectBase))
    {
      deserializer->RegisterHandler(typeid(vtkObject), Deserialize_vtkObject);
      deserializer->RegisterConstructor("vtkObject", []() { return␣
↪vtkObject::New(); });
      success = 1;
    }
  }
  return success;
}
```

9.9.2 Marshal hint macro

1. Classes which are annotated with `VTK_MARSHALAUTO` are considered by the `vtkWrapSerDes` executable.
2. Classes annotated with `VTK_MARSHALMANUAL` are hand coded in the same module. Here are some examples:
   - `Common/Core/vtkCollectionSerDesHelper.cxx` for `Common/Core/vtkCollection.h`
   - `Common/DataModel/vtkCellArraySerDesHelper.cxx` for `Common/DataModel/vtkCellArray.h`

9.9.3 Convenient script to annotate headers and module

- The Utilities/Marshalling/marshal_macro_annotate_headers.py script annotates headers for automatic or manual serialization. It is fed and driven by the accompanying Utilities/Marshalling/VTK_MARSHALAUTO.txt, Utilities/Marshalling/VTK_MARSHALMANUAL.txt and Utilities/Marshalling/ignore.txt.
- When the `-u, --update` argument is used, headers are in-place edited to use the `VTK_MARSHAL(AUTO|MANUAL)` wrapping hint. Files that already have this hint are untouched.
- When the `-t, --test` argument is used, the source tree is checked for inconsistent use of marshal macros.

9.10 Python Wrappers

9.10.1 Introduction

This document is a reference for using VTK from Python. It is not a tutorial and provides very little information about VTK itself, but instead describes in detail the features of the Python wrappers and how using VTK from Python differs from using VTK from C++. It assumes that the reader is already somewhat familiar with both Python and VTK.

9.10.2 Background

The Python wrappers are automatically generated from the VTK source code, and for the most part, there is a one-to-one mapping between the VTK classes and methods that you can use from Python and the ones that you can use from C++. More specifically, the wrappers are a package of Python extension modules that interface directly to the VTK C++ libraries. When you use VTK through the wrappers, you are actually executing compiled C++ code, and there is very little performance difference between VTK/C++ and VTK/Python.

9.10.3 Installation

VTK for Python can be installed via either conda or pip, where the conda packages is maintained on conda-forge, while the pip packages are maintained by the VTK developers themselves. If you are first getting started, then pip is probably the most convenient way to install VTK for Python:

```
pip install vtk
```

This will provide a basic installation of VTK that includes all core functionality, but which will not include some of the specialized VTK modules that rely on external libraries. Binary packages for VTK can also be downloaded directly from https://www.vtk.org/download/.

Instructions for building VTK from source code are given in the file Documentation/dev/build.md within the source repository.

9.10.4 Importing

VTK is comprised of over one hundred individual modules. Programs can import just the modules that are needed, in order to reduce load time.

```
from vtkmodules.vtkCommonCore import vtkObject
from vtkmodules.vtkFiltersSources import vtkConeSource, vtkSphereSource
from vtkmodules.vtkRenderingCore import (
    vtkActor,
    vtkDataSetMapper,
    vtkRenderer,
    vtkRenderWindow
)
import vtkmodules.vtkRenderingOpenGL2
```

When getting started, however, it is hard to know what modules you will need. So if you are experimenting with VTK in a Python console, or writing a quick and dirty Python script, it is easiest to simply import everything. There is a special module called ‘`all`’ that allows this to be done:

```
from vtkmodules.all import *
```

After importing the VTK classes, you can check to see which module each of the classes comes from:

```
for c in vtkObject, vtkConeSource, vtkRenderWindow:
    print(f"from {c.__module__} import {c.__name__}")
```

The output is as follows:

```
from vtkmodules.vtkCommonCore import vtkObject
from vtkmodules.vtkFiltersSources import vtkConeSource
from vtkmodules.vtkRenderingCore import vtkRenderWindow
```

Factories and Implementation Modules

In the first ‘import’ example above, you might be wondering about this line:

```
import vtkmodules.vtkRenderingOpenGL2
```

This import is needed because `vtkRenderingOpenGL2` provides the OpenGL implementations of the classes in `vtkRenderingCore`. To see this in action, open a new Python console and do the following:

```
>>> from vtkmodules.vtkRenderingCore import vtkRenderWindow
>>> renwin = vtkRenderWindow()
>>> type(renwin)
<class 'vtkmodules.vtkRenderingCore.vtkRenderWindow'>
>>>
>>> import vtkmodules.vtkRenderingOpenGL2
>>> renwin2 = vtkRenderWindow()
>>> type(renwin2)
<class 'vtkmodules.vtkRenderingOpenGL2.vtkXOpenGLRenderWindow'>
```

After `vtkRenderingOpenGL2` has been imported, the `vtkRenderWindow()` constructor magically starts returning a different type of object. This occurs because `vtkRenderWindow` is a *factory* class, which means that the kind of object it produces can be overridden by an *implementation* class. In order for the implementation class to do the override, all that is necessary is that its module is imported. To make things even more confusing, `vtkRenderingOpenGL2` is not the only module that contains implementations for the factory classes in `vtkRenderingCore`. The following modules are often needed, as well:

```
import vtkmodules.vtkInteractionStyle
import vtkmodules.vtkRenderingFreeType
```

Although you only need implementations for the factory classes that you use, it can be hard to know which classes are factory classes, or what modules contain implementations for them. Also, it can be difficult to even know what classes you are using, since many VTK classes make use of other VTK classes. An example of this is `vtkDataSetMapper`, which internally uses `vtkPolyDataMapper` to do the rendering. So even though `vtkDataSetMapper` is not a factory class, it needs an OpenGL implementation for `vtkPolyDataMapper`.

The simplest approach is to import all the important implementation modules into your program, even if you are not certain that you need them.

- For `vtkRenderingCore`, `import vtkRenderingOpenGL2, vtkRenderingFreeType, vtkInteractionStyle`
- For `vtkRenderingVolume`, `import vtkRenderingVolumeOpenGL2`

- For vtkCharts, import vtkContextOpenGL2

Classic VTK Import

There are many VTK programs that still import the ‘vtk’ module, which has been available since VTK 4.0, rather than using the ‘vtkmodules’ package that was introduced in VTK 8.2:

```
import vtk
```

The advantage (and disadvantage) of this is that it imports everything. It requires just one import statement for all of VTK, but it can be slow because VTK has grown to be very large over the years.

Also note that, between VTK 8.2 and VTK 9.2.5, the use of the vtk module would confuse the auto-completion features of IDEs such as PyCharm. This was fixed in VTK 9.2.6. For 9.2.5 and earlier, the following can be used:

```
import vtkmodules.all as vtk
```

From the programmer’s perspective, this is equivalent to ‘import vtk’.

9.10.5 VTK Classes and Objects

Classes Derived from vtkObjectBase

In C++, classes derived from vtkObjectBase are instantiated by calling New(). In Python, these classes are instantiated by simply calling the constructor:

```
o = vtkObject()
```

For factory classes, the returned object’s type might be a subtype of the class. This occurs because the Python wrappers are actually calling New() for you, which allows the VTK factory overrides to occur:

```
>>> a = vtkActor()
>>> type(a)
<class 'vtkmodules.vtkRenderingOpenGL2.vtkOpenGLActor'>
```

When you create a VTK object in Python, you are in fact creating two objects: a C++ object, and a Python object that holds a pointer to the C++ object. The repr() of the object shows the memory address of the C++ object (in parentheses) and of the Python object (after the ‘at’):

```
>>> a = vtkFloatArray()
>>> a
<vtkmodules.vtkCommonCore.vtkFloatArray(0x5653a6a6f700) at 0x7f0e7aecf5e0>
```

If you call str() or print() on these objects, the wrappers will call the C++ PrintSelf() method. The printed information can be useful for debugging:

```
>>> o = vtkObject()
>>> print(o)
vtkObject (0x55858308a210)
  Debug: Off
  Modified Time: 85
  Reference Count: 1
  Registered Events: (none)
```

Other Classes (Special Types)

VTK also uses several classes that aren't derived from `vtkObjectBase`. The most important of these is `vtkVariant`, which can hold any type of object:

```
>>> v1 = vtkVariant('hello')
>>> v1
vtkmodules.vtkCommonCore.vtkVariant('hello')
>>> v2 = vtkVariant(3.14)
>>> v2
vtkmodules.vtkCommonCore.vtkVariant(3.14)
```

The wrapping of these classes is fully automatic, but is done in a slightly different manner than `vtkObjectBase`-derived classes. First, these classes have no `New()` method, and instead the public C++ constructors are wrapped to create an equivalent Python constructor. Second, the Python object contains its own copy of the C++ object, rather than containing just a pointer to the C++ object. The vast majority of these classes are lightweight containers and numerical types. For example, `vtkQuaterniond`, `vtkRectf`, `vtkColor4ub`, etc. Many of them are actually class templates, which are discussed below.

When you apply `print()` or `str()` to these objects, the `operator<<` of the underlying C++ object is used to print them. For `repr()`, the name of the type name is printed, followed by the `str()` output in prentheses. The result looks similar to a constructor, though it might look strange depending on what `operator<<` produces.

```
>> v = vtkVariant()
>> print(repr(v))
vtkmodules.vtkCommonCore.vtkVariant((invalid))
```

Class Templates

There are several C++ templates in VTK, which can be tricky to use from the wrappers since the Python language has no real concept of templates. The wrappers wrap templates as dictionary-like objects that map the template parameters to template instantiations:

```
>>> vtkSOADataArrayTemplate
<template vtkCommonCorePython.vtkSOADataArrayTemplate>
>>> vtkSOADataArrayTemplate.keys()
['char', 'int8', 'uint8', 'int16', 'uint16', 'int32', 'uint32', 'int',
'uint', 'int64', 'uint64', 'float32', 'float64']
>>> c = vtkSOADataArrayTemplate['float64']
>>> c
<class 'vtkmodules.vtkCommonCore.vtkSOADataArrayTemplate_IdE'>
```

The wrappers instantiate the C++ template for a few useful types, as indicated by the `keys()` of the template. The Python type name also has a suffix (the '`IdE`') that indicates the template parameters in a compressed form according to IA64 C++ ABI name mangling rules, even when VTK is built with a compiler that does not use the IA64 ABI natively.

Objects are created by first instantiating the template, and then instantiating the class:

```
>>> a = vtkSOADataArrayTemplate['float32']()
>>> a.SetNumberOfComponents(3)
```

In the case of multiple template parameters, the syntax can look rather complicated, but really it isn't all that bad. For example, constructing a `vtkTuple<double,4>` in Python looks like this, with the template args in square brackets and the constructor args in parentheses:

```
>>> vtkTuple['float64',4]([1.0, 2.0, 3.0, 4.0])
vtkmodules.vtkCommonMath.vtkTuple_IdLi4EE([1.0, 2.0, 3.0, 4.0])
```

The type names are the same as numpy's dtypes: `bool`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `float32`, and `float64`. Since `int64` is '`long long`', `int` is used for `long`. Also see *Template Keys* in *Advanced Topics*.

9.10.6 Method Calls

When VTK methods are called from Python, conversion of all parameters from Python to C++ occurs automatically. That is, if the C++ method signature expects an integral type, you can pass a Python `int`, and if C++ expects a floating-point type, you can pass a Python `float` (or any type that allows implicit conversion to `float`).

For C++ '`char`' parameters, which are rarely used in VTK, you must pass a string with a length of 1 or 0 bytes. For unicode, the code must fit into eight bits (either ASCII, or within the Latin-1 Supplement block). An empty string signifies a null byte, and '\0' can also be used.

A Python `tuple`, `list`, or any other Python sequence can be passed to a VTK method that requires an array or `std::vector` in C++:

```
>>> a = vtkActor()
>>> p = (100.0, 200.0, 100.0)
>>> a.SetPosition(p)
```

If the method is going to modify the array that you pass as a parameter, then you must pass a Python `list` that has the correct number of slots to accept the returned values. If you try this with a `tuple`, you will get a `TypeError` because `tuple` is immutable.

```
>>> z = [0.0, 0.0, 0.0]
>>> vtkMath.Cross((1,0,0),(0,1,0),z)
>>> print(z)
[0.0, 0.0, 1.0]
```

For multi-dimensional array parameters, you can either use a nested list, or you can use numpy array with the correct shape.

If the C++ method returns a pointer to an array, then in Python the method will return a tuple if the wrappers know the size of the array. In most cases, the size is hinted in the header file.

```
>>> a = vtkActor()
>>> print(a.GetPosition())
(0.0, 0.0, 0.0)
```

Finally, Python `None` is treated the same as C++ `nullptr`, which allows you to pass null objects and null strings:

```
>>> a = vtkActor()
>>> a.SetMapper(None)
>>> print(a.GetMapper())
None
```

Wrappable and Unwrappable Methods

A method cannot be used from Python if its C++ parameters or return type cannot be converted to or from Python by the wrappers, or if the method is templated. Common non-convertible types include `std::ostream`, `std::istream`, and all STL container types except for `std::vector` (see below), and any non-trivial pointer type or any pointer to an object whose class is not derived from `vtkObjectBase`.

The wrappable parameter types are:

- `char`, wrapped as a single ASCII character in a Python `str`
- `signed char` and `unsigned char`, wrapped as Python `int`
- `short`, `int`, `long` and `long long`, wrapped as Python `int`
- `unsigned short` to `unsigned long long`, wrapped as Python `int`
- `float` and `double`, wrapped as Python `float`
- `size_t` and `ssize_t`, wrapped as Python `int`
- `std::string`, wrapped as Python `str` via utf-8 encoding/decoding
- typedefs of all the above, for any typedef defined in a VTK header file
- `std::vector<T>` where `T` is one of the above, as Python `tuple` or `list`
- `const T&` where `T` is any of the above, wrapped as described above
- `T[N]` where `T` is a fundamental type, as Python `tuple` or `list`
- `T[N][M]` where `T` is a fundamental type, as nested `tuple` or `list`
- `T*` where `T` is a fundamental type, as `tuple` or `list`
- `vtkObjectBase*` and derived types, as their respective Python type
- `vtkSmartPointer<T>` as the Python vtkObjectBase-derived type `T`
- `std::vector<vtkSmartPointer<T>>` as a sequence of objects of type `T`
- `const std::vector<vtkSmartPointer<T>>` as a sequence of objects of type `T`
- other wrapped classes (like `vtkVariant`), but not pointers to these types
- `char*`, as Python `str` via utf-8 encoding/decoding
- `void*`, as Python buffer (e.g. `bytes` or `bytearray`)
- the parameter list `(void (*f)(void*), void*)` as a Python callable type

References like `int&` and `std::string&` are wrapped via a reference proxy type as described in the *Pass by Reference* section below. Non-const references to `std::vector<T>` and other mutable types do not use a proxy, but instead require that a mutable Python object is passed, for example a `list` rather than a `tuple`.

A `void*` parameter can accept a pointer in two different ways: either from any Python object that supports the Python buffer protocol (this includes all numpy arrays along with the Python bytes and bytearray types), or from a string that contains a mangled pointer of the form '`_hhhhhhhhhhhh_p_void`' where '`hhhhhhhhhhhh`' is the hexadecimal address. Return-value `void*` will always be a string containing the mangled pointer.

Also, a `T*` parameter for fundamental type `T` can accept a buffer object, if and only if it is annotated with the `VTK_ZEROCOPY` hint in the header file. With this hint, a numpy array of `T` can be passed to a `T*` parameter and the VTK method will directly access the memory buffer of the array. Hence the name 'zerocopy', which indicates no copying is done, and that direct memory access is used.

The `vtkObject::AddObserver()` method has a special wrapping, as discussed in the *Observer Callbacks* section below.

Conversion Constructors

If a wrapped type has constructor that takes one parameter, and if that constructor is not declared '`explicit`', then the wrappers will automatically use that constructor for type conversion to the parameter type. The wrappers ensure that this conversion occurs in Python in the same manner that it is expected to occur in C++.

For example, `vtkVariantArray` has a method `InsertNextItem(v:vtkVariant)`, and `vtkVariant` has a constructor `vtkVariant(x:int)`. So, you can do this:

```
>>> variantArray.InsertNextItem(1)
```

The wrappers will automatically construct a `vtkVariant` from '1', and will then pass it as a parameter to `InsertNextItem()`. This is a feature that most C++ programmers will take for granted, but Python users might find it surprising.

Overloaded Methods

If you call a VTK method that is overloaded, the Python wrappers will choose the overload that best matches the supplied arguments. This matching takes into account all allowed implicit conversions, such as int to float or any conversion constructors that are defined for wrapped objects.

Some overloads will be unavailable (not wrapped) either because they are unwrappable as per the criteria described above, or because they are shadowed by another overload that is always preferable. A simple example of this is any methods that is overloaded on C++ `float` and `double`. The Python `float` type is a perfect match C++ `double`, therefore the `float` overload is not wrapped.

Static Methods

A static method can be called without an instance. For example,

```
vtkObject.SetGlobalWarningDisplay(1)
```

Some VTK classes, like vtkMath, consist solely of static methods. For others, like `vtkMatrix4x4`, most of the non-static methods have static overloads. Within Python, the only way to tell if a VTK method is static (other than trying it) is to look at its docstring.

Unbound Methods

When a non-static method is called on the class, rather than on an instance, it is called an unbound method call. An unbound method call must provide 'self' as the first argument, where 'self' is an instance of either the class or a subclass.

```
w = vtkRenderWindow()
vtkWindow.Render(w)
```

In other words, the wrappers translate Python unbound method calls into C++ unbound method calls. These are useful when deriving a Python class from a wrapped VTK class, since they allow you to call any base class methods that have been overridden in the subclass.

Operator Methods

For special classes (the ones not derived from `vtkObjectBase`), some useful C++ operators are wrapped in python. The '`[]`' operator is wrapped for indexing and item assignment, but because it relies on hints to guess which indices are out-of-bounds, it is only wrapped for `vtkVector` and related classes.

The comparison operators '<' '<=' '==' '>=' '>' are wrapped for all classes that have these operators in C++. These operators allow sorting of `vtkVariant` objects with Python.

The '<<' operator for printing is wrapped and is used by the python `print()` and `str()` commands.

Strings and Bytes

VTK uses both `char*` and `std::string` for strings. As far as the wrappers are concerned, these are equivalent except that the former can be `nullptr` (`None` in Python). For both, the expected encoding is ASCII or utf-8.

In Python, either `str` or `bytes` can be used to store strings, and both of these can be passed to VTK methods that require `char*` or `std::string` (or the legacy `vtkStdString`). A `str` object is passed to VTK as utf-8, while a `bytes` object is passed as-is.

When a VTK method returns a string, it is received in Python as a `str` object if it is valid utf-8, or as a `bytes` object if not. The caller should check the type of the returned object (`str`, `bytes`, or perhaps `None`) if there is any reason to suspect that non-utf-8 text might be present.

STL Containers

VTK provides conversion between `std::vector` and Python sequences such as `tuple` and `list`. If the C++ method returns a vector, the Python method will return a tuple:

```
C++: const std::vector<std::string>& GetPaths()
C++: std::vector<std::string> GetPaths()
Python: GetPaths() -> Tuple[str]
```

If the C++ method accepts a vector, then the Python method can be passed any sequence with compatible values:

```
C++: void SetPaths(const std::vector<std::string>& paths)
C++: void SetPaths(std::vector<std::string> paths)
Python: SetPaths(paths: Sequence[str]) -> None
```

Furthermore, if the C++ method accepts a non-const vector reference, then the Python method can be passed a mutable sequence (e.g. `list`):

```
C++: void GetPaths(std::vector<std::string>& paths)
Python: GetPaths(paths: MutableSequence[str]) -> None
```

The value type of the `std::vector<T>` must be `std::string` or a fundamental numeric type such as `double` or `int` (including `signed char` and `unsigned char` but excluding `char`).

Smart pointers

The wrappers will automatically convert between C++ `vtkSmartPointer<T>` and objects of type `T` (or `None`, if the smart pointer is empty):

```
C++: vtkSmartPointer<vtkObject> TakeObject()
Python: TakeObject() -> vtkObject
```

In other words, in Python the smart pointer doesn't look any different from the object it points to. Under the hood, however, the wrappers understand that the smart pointer carries a reference to the object and will take responsibility for deleting that reference.

A C++ method can return a vector of smart pointers, which will be seen in Python as a tuple of objects:

```
C++: std::vector<vtkSmartPointer<vtkObject>> GetObjects()
Python: GetObject() -> Tuple[vtkObject]
```

If a C++ method expects `std::vector<vtkSmartPointer<T>>` as a parameter, the wrappers will automatically construct the vector from any sequence that is passed from Python. The objects in the sequence must be of type `T` (or a subclass of `T`, or `None`). If not, a `TypeError` will be raised.

Pass by Reference

Many VTK methods use pass-by-reference to return values back to the caller. Calling these methods from Python requires special consideration, since Python's `str`, `tuple`, `int`, and `float` types are immutable. The wrappers provide a '`reference`' type, which is a simple container that allows pass-by-reference.

For example, consider the following C++ method that uses pass-by-reference:

```
void GetCellAtId(vtkIdType cellId, vtkIdType& cellSize, vtkIdType const*& cellPoints)
```

It requires a reference to `vtkIdType` (a Python `int`), and to `vtkIdType const*` (a tuple of `int`s). So we can call this method as follows:

```
>>> from vtkmodules.vtkCommonCore import reference
>>> from vtkmodules.vtkCommonDataModel import vtkCellArray
>>>
>>> # Build a cell array
>>> a = vtkCellArray()
>>> a.InsertNextCell(3, (1, 3, 0))
>>>
>>> # Create the reference objects
>>> n = reference(0)
>>> t = reference((0,))
>>>
>>> # Call the pass-by-reference method
>>> a.GetCellAtId(0, n, t)
>>>
>>> n.get()
3
>>> t.get()
(1, 3, 0)
```

Some important notes when using pass-by-reference:

1. The reference constructor must be given a value of the desired type. The method might use this value or might ignore it.
2. Calling the `get()` method of the reference is usually unnecessary, because the reference already supports the interface protocols of the object that it contains.

Preconditions

One very real concern when using VTK from Python is that the parameters that you pass to a method might cause the program to crash. In particular, it is very easy to pass an index that causes an out-of-bounds memory access, since the C++ methods don't do bounds checking. As a safety precaution, the wrappers perform the bounds check before the C++ method is called:

```
>>> a = vtkFloatArray()
>>> a.GetValue(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: expects 0 <= id && id < GetNumberOfValues()
```

All precondition checks raise a `ValueError` if they fail, since they are checks on the values of the parameters. The wrappers don't know if C++ is using the parameter as an index, so `IndexError` is not used.

Currently the only way to find out if a method has preconditions is to look at the declaration of the method in the C++ header file to see if it has a `VTK_EXPECTS` hint.

9.10.7 Observer Callbacks

Similar to what can be done in C++, a Python function can be called each time a VTK event is invoked on a given object. In general, the callback function should have the signature `func(obj:vtkObject, event:str)`, or `func(self, obj:vtkObject, event:str)` if it is a method of a class.

```
>>> def onObjectModified(object, event):
>>>     print('object: %s - event: %s' % (object.GetClassName(), event))
>>>
>>> o = vtkObject()
>>> o.AddObserver(vtkCommand.ModifiedEvent, onObjectModified)
1
>>> o.Modified()
object: vtkObject - event: ModifiedEvent
```

Call Data

In case there is a 'CallData' value associated with an event, in C++, you have to cast it from `void*` to the expected type using `reinterpret_cast`. The equivalent in python is to add a `CallDataType` attribute to the associated python callback method. The supported `CallDataType` values are `VTK_STRING`, `VTK_OBJECT`, `VTK_INT`, `VTK_LONG`, `VTK_DOUBLE`, and `VTK_FLOAT`.

The following example uses a function as a callback, but a method or any callable object can be used:

```
>>> from vtkmodules.vtkCommonCore import vtkCommand, VTK_INT
>>>
>>> def onError(object, event, calldata):
```

(continues on next page)

(continued from previous page)

```
>>>     print('object: %s - event: %s - msg: %s' % (object.GetClassName(), event,␣
↪calldata))
>>>
>>> onError.CallDataType = VTK_INT
>>>
>>> lt = vtkLookupTable()
>>> lt.AddObserver(vtkCommand.ErrorEvent, onError)
1
>>> lt.SetTableRange(2,1)
object: vtkLookupTable - event: ErrorEvent - msg: ERROR:
In /home/user/VTK/Common/Core/vtkLookupTable.cxx, line 122
vtkLookupTable (0x6b40b30): Bad table range: [2, 1]
```

For convenience, the `CallDataType` can also be specified where the function is first declared with the help of the `@calldata_type` decorator:

```
>>> from vtkmodules.util.misc import calldata_type
>>>
>>> @calldata_type(VTK_INT)
>>> def onError(object, event, calldata):
>>>     print('object: %s - event: %s - msg: %s' % (object.GetClassName(),
                                                    event, calldata))
```

9.10.8 Other Wrapped Entities

Constants

Most of the constants defined in the VTK header files are available in Python, and they can be accessed from the module in which they are defined. Many of these are found in the `vtkCommonCore` module, where they were defined as preprocessor macros.

```
>>> from vtkmodules.vtkCommonCore import VTK_DOUBLE_MAX
>>> VTK_DOUBLE_MAX
1.0000000000000001e+299
```

Others are defined as enums, often within a class namespace. If the enum is anonymous, then its values are `int`.

```
>>> vtkCommand.ErrorEvent
39
```

Constants in the header files are wrapped if they are enums, or if they are const variables of a wrappable scalar type, or if they are preprocessor symbols that evaluate to integer, floating-point, or string literal types.

Enum Types

Each named enum type is wrapped as a new Python type, and members of the enum are instances of that type. This allows type checking for enum types:

```
>>> from vtkmodules.vtkCommonColor import vtkColorSeries
>>> vtkColorSeries.COOL
2
>>> isinstance(vtkColorSeries.ColorSchemes, vtkColorSeries.COOL)
>>> cs = vtkColorSeries()
>>> cs.SetColorScheme(vtkColorSeries.COOL)
```

Enum classes are wrapped in a manner similar to named enums, except that the enum values are placed within the enum class namespace. For example, `vtkEventDataAction` is an enum class, with '`Press`' as a member:

```
>>> from vtkmodules.vtkCommonCore import vtkEventDataAction
>>> vtkEventDataAction.Press
1
>>> isinstance(vtkEventDataAction.Press, vtkEventDataAction)
True
```

In the first example, the `ColorSchemes` enum type and the `COOL` enum value were both defined in the `vtkColorSeries` namespace. In the second example, the `vtkEventDataAction` enum class was defined in the module namespace, and the `Press` value was defined in the enum class namespace.

Note that the VTK enum types behave like C++ enums, and not like the Python enums types provided by the Python '`enum`' module. In particular, all VTK enum values can be used anywhere that an `int` can be used.

Namespaces

Namespaces are currently wrapped in a very limited manner. The only namespace members that are wrapped are enum constants and enum types. There is no wrapping of namespaced classes or functions, or of nested namespaces. Currently, the wrappers implement namespaces as Python `module` objects.

9.10.9 Docstrings

The wrappers automatically generate docstrings from the doxygen comments in the header files. The Python `help()` command can be used to print the documentation to the screen, or the `__doc__` attributes of the classes and methods can be accessed directly.

Method Docstrings

The method docstrings are formatted with the method signatures first, followed by doxygen comments. The Python method signatures have type annotations, and are followed by the C++ method signatures for completeness.

```
    InvokeEvent(self, event:int, callData:Any) -> int
    C++: int InvokeEvent(unsigned long event, void* callData)
    InvokeEvent(self, event:str, callData:Any) -> int
    C++: int InvokeEvent(const char* event, void* callData)
    InvokeEvent(self, event:int) -> int
    C++: int InvokeEvent(unsigned long event)
    InvokeEvent(self, event:str) -> int
```

(continues on next page)

(continued from previous page)

```
C++: int InvokeEvent(const char* event)

This method invokes an event and returns whether the event was
aborted or not. If the event was aborted, the return value is 1,
otherwise it is 0.
```

Some Python IDEs will automatically show the docstring as soon as you type the name of the method.

Class Docstrings

The class docstrings include a brief description of the class, followed by the name of the superclass, and then the full doxygen documentation, including doxygen markup:

```
vtkMatrix4x4 - represent and manipulate 4x4 transformation matrices

Superclass: vtkObject

vtkMatrix4x4 is a class to represent and manipulate 4x4 matrices.
Specifically, it is designed to work on 4x4 transformation matrices
found in 3D rendering using homogeneous coordinates [x y z w]. Many
of the methods take an array of 16 doubles in row-major format. Note
that OpenGL stores matrices in column-major format, so the matrix
contents must be transposed when they are moved between OpenGL and
VTK.
@sa
vtkTransform
```

If the class is not derived from `vtkObjectBase`, then it will have one or more public constructors, and these will be included before the comments:

```
vtkSimpleCriticalSection() -> vtkSimpleCriticalSection
C++: vtkSimpleCriticalSection()
vtkSimpleCriticalSection(isLocked:int) -> vtkSimpleCriticalSection
C++: vtkSimpleCriticalSection(int isLocked)

vtkSimpleCriticalSection - Critical section locking class

vtkCriticalSection allows the locking of variables which are accessed
through different threads.
```

Template Docstrings

Class templates are documented similar to classes, except that they include a 'Provided Types' section that lists the available template instantiations and the C++ template arguments that they correspond to.

```
vtkSOADataArrayTemplate - Struct-Of-Arrays implementation of
vtkGenericDataArray.

Superclass: vtkGenericDataArray[vtkSOADataArrayTemplate[ValueTypeT],ValueTypeT]

vtkSOADataArrayTemplate is the counterpart of vtkAOSDataArrayTemplate.
```

(continues on next page)

(continued from previous page)

```
    Each component is stored in a separate array.

    @sa
    vtkGenericDataArray vtkAOSDataArrayTemplate


    Provided Types:

      vtkSOADataArrayTemplate[char] => vtkSOADataArrayTemplate<char>
      vtkSOADataArrayTemplate[int8] => vtkSOADataArrayTemplate<signed char>
      vtkSOADataArrayTemplate[uint8] => vtkSOADataArrayTemplate<unsigned char>
      vtkSOADataArrayTemplate[int16] => vtkSOADataArrayTemplate<short>
      vtkSOADataArrayTemplate[uint16] => vtkSOADataArrayTemplate<unsigned short>
      vtkSOADataArrayTemplate[int32] => vtkSOADataArrayTemplate<int>
      vtkSOADataArrayTemplate[uint32] => vtkSOADataArrayTemplate<unsigned int>
      vtkSOADataArrayTemplate[int] => vtkSOADataArrayTemplate<long>
      vtkSOADataArrayTemplate[uint] => vtkSOADataArrayTemplate<unsigned long>
      vtkSOADataArrayTemplate[int64] => vtkSOADataArrayTemplate<long long>
      vtkSOADataArrayTemplate[uint64] => vtkSOADataArrayTemplate<unsigned long long>
      vtkSOADataArrayTemplate[float32] => vtkSOADataArrayTemplate<float>
      vtkSOADataArrayTemplate[float64] => vtkSOADataArrayTemplate<double>
```

Unlike classes, the template documentation is formatted similarly regardless of whether the the class template derives from `vtkObjectBase` or not:

```
    vtkVector - templated base type for storage of vectors.

    Superclass: vtkTuple[T,Size]

    This class is a templated data type for storing and manipulating fixed
    size vectors, which can be used to represent two and three dimensional
    points. The memory layout is a contiguous array of the specified type,
    such that a float[2] can be cast to a vtkVector2f and manipulated. Also
    a float[6] could be cast and used as a vtkVector2f[3].


    Provided Types:

      vtkVector[float64,4] => vtkVector<double, 4>
      vtkVector[float32,4] => vtkVector<float, 4>
      vtkVector[int32,4] => vtkVector<int, 4>
      vtkVector[float64,2] => vtkVector<double, 2>
      vtkVector[float32,2] => vtkVector<float, 2>
      vtkVector[int32,2] => vtkVector<int, 2>
      vtkVector[float64,3] => vtkVector<double, 3>
      vtkVector[float32,3] => vtkVector<float, 3>
      vtkVector[int32,3] => vtkVector<int, 3>
```

9.10.10 Internals and Advanced Topics

Special Attributes

Classes and objects derived from `vtkObjectBase` have special attributes, which are only used in very special circumstances.

The `__vtkname__` attribute of the class provides the same string that the GetClassName() method returns. With the exception of classes that are template instantiations, it is identical to the `__name__` attribute. For template instantiations, however, `GetClassName()` and `__vtkname__` return the result of calling `typeid(cls).name()` from C++, which provides a platform specific result:

```
>>> vtkSOADataArrayTemplate['float32'].__vtkname__
'23vtkSOADataArrayTemplateIfE'
```

This can be used to get the VTK `ClassName` when you don't have an instantiation to call `GetClassName()` on. It is useful for checking the type of a C++ VTK object against a Python VTK class.

The `__this__` attribute of the objects is a bit less esoteric, it provides a pointer to the C++ object as a mangled string:

```
>>> a = vtkFloatArray()
>>> a.__this__
'_00005653a6a6f700_p_vtkFloatArray'
```

The string provides the hexadecimal address of '`this`', followed by '`p`' (shorthand for *pointer*), and the type of the pointer. You can also construct a Python object directly from the C++ address, if the address is formatted as described above:

```
>>> a = vtkFloatArray('_00005653a6a6f700_p_vtkFloatArray')
>>> a
<vtkmodules.vtkCommonCore.vtkFloatArray(0x5653a6a6f700) at 0x7f0e7aecf5e0>
```

If you call the constructor on the string provided by `__this__`, you will get exactly the same Python object back again, rather than a new object. But this constructor can be useful if you have some VTK code that has been wrapped with a different wrapper tool, for example with SWIG. If you can get the VTK pointer from SWIG, you can use it to construct Python object that can be used with the native VTK wrappers.

Wrapper Hints

A wrapper hint is an attribute that can be added to a class, method, or parameter declaration in a C++ header file to give extra information to the wrappers. These hints are defined in the `vtkWrappingHints.h` header file.

The following hints can appear before a method declaration:

- `VTK_WRAPEXCLUDE` excludes a method from the wrappers
- `VTK_NEWINSTANCE` passes ownership of a method's return value to the caller

For convenience, `VTK_WRAPEXCLUDE` can also be used to exclude a whole class. The `VTK_NEWINSTANCE` hint is used when the return value is a `vtkObjectBase*` and the caller must not increment the reference count upon acceptance of the object (but must still decrement the reference count when finished with the object).

The following hints can appear after a method declaration:

- `VTK_EXPECTS(cond)` provides preconditions for the method call
- `VTK_SIZEHINT(expr)` marks the array size of a return value
- `VTK_SIZEHINT(name, expr)` marks the array size of a parameter

For `VTK_EXPECTS(cond)`, the precondition must be valid C++ code, and can use any of the parameter names or `this`. Even without `this`, any public names in the class namespace (including method names) will be resolved. See the *Preconditions* section for additional information.

`VTK_SIZEHINT(expr)` is used for methods that return an array as type `T*`, where `T` is a numeric data type. The hint allows the wrappers to convert the array to a tuple of the correct size. Without the size hint, the wrappers will return the pointer as a string that provides a mangled memory address of the form '`_hhhhhhhhhhhh_p_void`' where '`hhhhhhhhhhhh`' is address expressed in hexadecimal.

`VTK_SIZEHINT(parameter_name, expr)` is used to hint parameters of type `T*` or `T&*` (with `T` as a numeric data type) so that the wrappers know the size of the array that the pointer is pointing to. The `expr` can be any expression that evaluates to an integer, and it can include parameter names, public class members and method calls, or the special name `_` (underscore) which indicates the method's return value. In the absence of a size hint, the wrappers cannot check that the length of the sequence passed from Python matches the size of the array required by the method. If the method requires a larger array than it receives, a buffer overrun will occur.

The following hints can appear before a parameter declaration:

- `VTK_FILEPATH` marks a parameter that accepts a pathlib.Path object
- `VTK_ZEROCOPY` marks a parameter that accepts a buffer object

More specifically, `VTK_FILEPATH` is used with `char*` and `std::string` parameters to indicate that the method also accepts any object with a `__fspath__()` method that returns a path string. And `VTK_ZEROCOPY` is used with `T*` parameters, for basic integer or float type `T`, to indicate that the Python buffer protocol will be used to access the values, rather than the Python sequence protocol that is used by default.

Deprecation Warnings

In addition to the wrapping hints, the Python wrappers are also aware of the deprecation attributes that have been applied to classes and methods. When a deprecated method is called, a `DeprecationWarning` is generated and information about the deprecation is printed, including the VTK version for the deprecation.

To ignore these warnings, use the following code:

```
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

To see each deprecation warning just once per session,

```
warnings.filterwarnings('once', category=DeprecationWarning)
```

Template Keys

The following is a table of common template key names, which are the same as the numpy dtype names. Note that you can actually use numpy dtypes as keys, as well as the native Python types `bool`, `int`, and `float`. There is some danger in using `int`, however, because it maps to C++ `long` which has a platform-dependent size (either 32 bits or 64 bits). Finally, the char codes from the Python `array` module can be used as keys, but they should be avoided since more programmers are familiar with numpy than with the much older `array` module.

C++ Type	Template Key	Type Key	Char Key	IA64 ABI Code
bool	'bool'	bool	'?'	IbE
char	'char'		'c'	IcE
signed char	'int8'		'b'	IaE
unsigned char	'uint8'		'B'	IhE
short	'int16'		'h'	IsE
unsigned short	'uint16'		'H'	ItE
int	'int32'		'i'	IiE
unsigned int	'uint32'		'I'	IjE
long	'int'	int	'l'	IlE
unsigned long	'uint'		'L'	ImE
long long	'int64'		'q'	IxE
unsigned long long	'uint64'		'Q'	IyE
float	'float32'		'f'	IfE
double	'float64'	float	'd'	IdE

Since the size of '`long`' and '`unsigned long`' is platform-dependent, these types should generally be avoided.

Exception Handling

There are times when an observer might generate a Python exception. Since the observers are called from C++, there is no good way to catch these exceptions from within Python. So, instead, the wrappers simply print a traceback to stderr and then clear the error indicator. The Python program will continue running unless the exception was a `KeyboardInterrupt` (Ctrl-C), in which case the program will exit with an error code of 1.

Deleting a vtkObject

There is no direct equivalent of VTK's `Delete()` method, since Python does garbage collection automatically. The Python object will be deleted when there are no references to it within Python, and the C++ object will be deleted when there are no references to it from within either Python or C++. Note that references can hide in unexpected places, for example if a method of an object is used as an observer callback, the object will not be deleted until the observer is disconnected.

The `DeleteEvent` can be used to detect object deletion, but note that the observer will receive a None for the object, since the observer is called after (not before) the deletion occurs:

```
>>> o = vtkObject()
>>> o.AddObserver('DeleteEvent', lambda o,e: print(e, o))
1
```

(continues on next page)

(continued from previous page)

```
>>> del o
DeleteEvent None
```

If you need to know what object is deleted, the identifying information must be extracted before the deletion occurs:

```
>>> o = vtkObject()
>>> o.AddObserver('DeleteEvent',lambda x,e,r=repr(o): print(e, r))
1
>>> del o
DeleteEvent <vtkmodules.vtkCommonCore.vtkObject(0x55783870f970) at 0x7f1e61678be0>
```

In cases where you need to track down tricky memory issues, you might find it useful to call the `GetReferenceCount()` method of the object directly.

Ghosts

A wrapped VTK object (derived from `vtkObjectBase`) is a Python object that holds a pointer to a C++ object (specifically, a `vtkObjectBase*`). The Python object can have attributes that the C++ object knows nothing about. So, what happens to these attributes if the Python object is deleted, but the C++ object lives on? Consider this simple example of storing the C++ object in an array and then deleting the Python object:

```
obj = vtkObject()
obj.tag = 'FirstObject'
va = vtkVariantArray()
va.InsertNextValue(obj)
del obj
```

When we retrieve the object from the array, we want it to have the '`tag`' attributes that it had we stored it. But you might wonder, aren't all Python-specific attributes deleted along with the Python object? The answer is, no they aren't, they're saved until until the C++ object itself is deleted.

The wrappers have a special place, which we will call the graveyard, where 'ghosts' of objects are stored when the objects are deleted. The ghost is not an object, but rather a container for the Python attributes of a deceased object. If the object ever reappears within Python, usually as a return value from a C++ method call, then the ghost is resurrected as a new Python object that has all the attributes of the original Python object.

The graveyard is only used for objects that have unfinished business. If a Python object has an empty dict and no other special attributes, then it will not go to the graveyard. Also, if the C++ object is deleted at the same time as the Python object, then the graveyard will not be used. Each ghost in the graveyard holds a weak pointer to its C++ object and will vanish when the C++ object is deleted (not immediately, but the next time the graveyard garbage collector runs).

Subclassing a VTK Class

It is possible to subclass a VTK class from within Python, but this is of limited use because the C++ virtual methods are not hooked to the Python methods. In other words, if you make a subclass of `vtkPolyDataAlgorithm` and override override the `Execute()` method, it will not be automatically called by the VTK pipeline. Your `Execute()` method will only be called if the call is made from Python.

The addition of virtual method hooks to the wrappers has been proposed, but currently the only way for Python methods to be called from C++ code is via callbacks. The `vtkProgrammableSource` and `vtkProgrammableFilter` are examples of VTK algorithm classes that use callbacks for execution, while `vtkInteractionStyleUser` can use observer callbacks for event handling.

Wrapping External VTK Modules

If you have your own C++ classes that are based on VTK, and if they are placed with a VTK module with a vtk.module file, then they can be wrapped as shown in the Module Wrapping Example. You will also find the cmake documentation on VTK modules to be useful.

9.10.11 Experimental Features

Python Class Overrides

VTK now supports overriding wrapped classes with Python subclasses. This enables developers to provide more Python friendly interfaces for certain classes. Here is a trivial example of an override:

```
from vtkmodules.vtkCommonCore import vtkPoints
@vtkPoints.override
class CustomPoints(vtkPoints):
    pass
```

Once the override is in place, any future `vtkPoints` Python object instances will be instances of the override class. This behavior is global.

```
points = vtk.vtkPoints() # returns an instance of CustomPoints
```

The override can be reversed by setting an override of `None`, but this will not impact instantiations that have already occurred.

```
vtkPoints.override(None)
```

If the class has already been overridden in C++ via VTK's object factory mechanism, then directly applying a Python override to that class will not work. Instead, the Python override must be applied to the C++ factory override. For example, on Windows,

```
@vtkWin32OpenGLRenderWindow.override
class CustomRenderWindow(vtkWin32OpenGLRenderWindow):
    ...
window = vtkRenderWindow() # creates a CustomRenderWindow
```

Please see *Subclassing a VTK Class* for restrictions on subclassing VTK classes through Python.

Stub Files for Type Hinting

VTK includes a script called `generate_pyi.py` that will generate pyi stub files for each wrapped VTK module. The purpose of these files, as explained in PEP 484, is to provide type information for all constants, classes, and methods in the modules. Each of these files contain blocks like this:

```
VTK_DOUBLE:int
VTK_DOUBLE_MAX:float
VTK_DOUBLE_MIN:float
...

class vtkObject(vtkObjectBase):
    def AddObserver(self, event:int, command:Callback, priority:float=0.0) -> int: ...
```

(continues on next page)

(continued from previous page)

```
    def GetMTime(self) -> int: ...
    @staticmethod
    def GetNumberOfGenerationsFromBaseType(type:str) -> int: ...
    @overload
    def HasObserver(self, event:int, __b:'vtkCommand') -> int: ...
    @overload
    def HasObserver(self, event:str, __b:'vtkCommand') -> int: ...

class vtkAbstractArray(vtkObject):
    class DeleteMethod(int): ...
    VTK_DATA_ARRAY_ALIGNED_FREE:'DeleteMethod'
    VTK_DATA_ARRAY_DELETE:'DeleteMethod'
    VTK_DATA_ARRAY_FREE:'DeleteMethod'
    VTK_DATA_ARRAY_USER_DEFINED:'DeleteMethod'
    def Allocate(self, numValues:int, ext:int=1000) -> int: ...
```

Python consoles like ipython and IDEs like PyCharm can use the information in these files to provide hints while you edit the code. These files are included in the Python packages for VTK, but they can also be built by executing the `generate_pyi.py` script. To do so, execute the script with the `vtkpython` executable (or with the regular python executable, if its paths are set for VTK):

```
vtkpython -m vtkmodules.generate_pyi
```

This will place build the pyi files and place them inside the `vtkmodules` package, where ipython and PyCharm should automatically find them. The help for this script is as follows:

```
usage: python generate_pyi.py [-p package] [-o output_dir] [module ...]
options:
  -p NAME        Package name [vtkmodules by default].
  -o OUTPUT      Output directory [package directory by default].
  -e EXT         Output file suffix [.pyi by default].
  module         Module or modules to process [all by default].
```

The pyi files are syntactically correct python files, so it is possible to load them as such in order to test them and inspect them.

9.11 Wrapping Tools

The wrapping tools consist of executables that pull information from C++ header files, and produce wrapper code that allows the C++ interfaces to be used from other programming languages (Python and Java). One can think of the wrappers as having a front-end that parses C++ header files, and a back-end that produces language-specific glue code.

All of the code in this directory is C, rather than C++. One might think this is silly, since the front-end parses C++ .h files and the back-end generates .cxx files. The original reason for this is that the parser uses lex and yacc, which are written in C and previously could not easily be linked into C++ programs.

9.11.1 The C++ Parser

vtkParse

The header vtkParse.h provides a C API for the C++ parser that wrappers use to read the VTK header files. The parser consists of three critical pieces: a preprocessor (see below), a lex-based lexical analyzer (lex.yy.c, generated from vtkParse.l) and a bison-based glr parser (vtkParse.tab.c, generated from vtkParse.y). Instructions on rebuilding the parser are provided at the end of this document.

vtkParsePreprocess

This is a preprocessor that can run independently of the parser. In general, the parser does not recursively parse `#include` files, but it does recursively preprocess them in order to gather all of the macro definitions within them.

vtkParseString

This provides low-level string handling routines that are used by the parser and the preprocessor. Most importantly, it contains a C++ tokenizer. It also contains a cache for storing strings (type names, etc.) that are encountered during the parse.

vtkParseSystem

This contains utilities for file system access. One of its functionalities is to manage a cache of where header files are located on the file system, so that header file lookups can be done inexpensively even on slow file systems.

vtkParseType

This is a header file that defines numerical constants that we use to identify C++ types, type qualifiers, and specifiers. These constants are used in the vtkParseData data structures described below.

vtkParseAttributes

This is a header file that defines numerical constants for wrapper-specific attributes that can be added to declarations in the VTK header files. For example, `[[vtk::wrapexclude]]` and `[[vtk::deprecated]]`. These attribute constants are stored in the vtkParseData data structures.

vtkParseData

The data structures defined in vtkParseData.h are used for the output of the parser. This header provides data structures for namespaces, classes, methods, typedefs, and for other entities that can be declared in a C++ file. The wrappers convert this data into wrapper code.

9.11.2 Parser Utilities

vtkParseExtras

This file provides routines for managing certain abstractions of the data that is produced by the parser. Most specifically, it provides facilities for expanding typedefs and for instantiating templates. Its code is not pretty.

vtkParseMerge

This provides methods for dealing with method resolution order. It defines a data structure for managing a class along with all the classes it derives from. It is needed for managing tricky details relating to inheritance, such as “using” declarations, overrides, virtual methods, etc.

vtkParseMangle

The Python wrappers rely on name-mangling routines to convert C++ names into names that can be used in Python. The mangling is done according to the rules of the IA64 ABI (this same mangling is used to convert C++ APIs into C APIs)

vtkParseHierarchy

A hierarchy file is a text file that lists information about all the types defined in a VTK module. The wrappers use these files to look up types from names. Through the use of vtkParseHierarchy, the wrappers can get detailed information about a type even if the header file only contains a forward reference, as long as the type is defined somewhere in another header.

vtkParseMain

A common main() function for use by wrapper tool executables. It provides a standard set of command-line options as well as response-file handling. It also invokes the parser.

9.11.3 Wrapper Utilities

vtkWrap

This has functions that are common to the wrapper tools for all the wrapper languages. Unlikely vtkParse, it deals with the generation of code, rather than the parsing of code.

vtkWrapText

This has functions for automatically generating documentation from the header files that are parsed. It produces the Python docstrings.

9.11.4 Python-Specific Utilities

These are named according to the pieces of wrapper code they produce.

- **vtkWrapPythonClass** creates type objects for vtkObjectBase classes
- **vtkWrapPythonType** creates type objects for other wrapped classes
- **vtkWrapPythonMethod** for calling C++ methods from Python
- **vtkWrapPythonOverload** maps a Python method to multiple C++ overloads
- **vtkWrapPythonMethodDef** generates the method tables for wrapped classes
- **vtkWrapPythonTemplate** for wrapping of C++ class templates
- **vtkWrapPythonNamespace** for wrapping namespaces
- **vtkWrapPythonEnum** creates type objects for enum types
- **vtkWrapPythonConstant** adds C++ constants to Python classes, namespaces

9.11.5 Python Wrapper Executables

vtkWrapPython

This executable will parse the C++ declarations from a header file and produce wrapper code that can be linked into a Python extension module.

vtkWrapPythonInit

This will produce the PyInit entry point for a Python extension module, as well as code for loading all the dependent modules. The .cxx file produced by vtkWrapPythonInit is linked together to the .cxx files that are produced by vtkWrapPython to create the module.

9.11.6 Java Wrapper Executables

- **vtkWrapJava** produces C++ wrapper code that uses the JNI
- **vtkParseJava** produces Java code that sits on top of the C++ code

9.11.7 Other Executables

vtkWrapHierarchy

This will slurp up all the header files in a VTK module and produce a “hierarchy.txt” file that provides information about all of the types that are defined in that module. In other words, it provides a summary of the module’s contents. The Python and Java wrapper executables rely on these “hierarchy.txt” files in order to look up types by name.

vtkWrapSerDes

This generates C++ code to serialize a VTK object into json and deserialize the object back from json. This relies upon the property parser from `vtkParseProperties`

9.11.8 Rebuilding the Parser

The code for the C++ parser is generated from the files vtkParse.l and vtkParse.y with the classic compiler-generator tools lex and yacc (or, more specifically, with their modern incarnations flex and bison). These tools are readily available on macOS and Linux systems, and they can be installed (with some difficulty) on Windows systems.

The C code that flex and bison generate is not styled according to VTK standards, and must be cleaned up in order to compile without warnings and in order to satisfy VTK's git hooks and style checks.

vtkParse.l

The file vtkParse.l contains regular expressions for tokenizing a C++ header file. It is used to generate the file lex.yy.c, which is directly included (i.e. as a C file) by the main parser file, vtkParse.tab.c.

To generate lex.yy.c from vtkParse.l, use the following steps.

1. Get a copy of flex, version 2.6.4 or later
2. Run `flex --nodefault --noline -olex.yy.c vtkParse.l`
3. In an editor, remove blank lines from the top and bottom of lex.yy.c
4. Replace all tabs with two spaces (e.g. `:%s/\t/  /g` in vi)
5. Remove spaces from the ends of lines (e.g. `:%s/ *$//` in vi)
6. Remove `struct yy_trans_info`, which is used nowhere in the code
7. Add the following code at line 23 (after "`end standard C headers`")

   #ifndef __cplusplus extern int isatty(int); #endif /* __cplusplus */

Finally, if you have clang-format installed, you can use it to re-style the code.

vtkParse.y

The file vtkParse.y contains the rules for parsing a C++ header file. Many of the rules in this file have the same names as in the description of the grammar in the official ISO standard. The file vtkParse.y is used to generate the file vtkParse.tab.c, which contains the parser.

1. Get a copy of bison 3.2.3 or later, it has a yacc-compatible front end.
2. Run `bison --no-lines -b vtkParse vtkParse.y`, to generate vtkParse.tab.c
3. In an editor, replace every `static inline` in vtkParse.tab.c with `static`
4. Replace `#if ! defined lint || defined __GNUC__` with `#if 1`
5. remove `YY_ATTRIBUTE_UNUSED` from `yyfillin`, `yyfill`, and `yynormal`
6. comment out the `break;` after `return yyreportAmbiguity`
7. replace `(1-yyrhslen)` with `(1-(int)yyrhslen)`
8. replace `sizeof yynewStates[0]` and `sizeof yyset->yystates[0]` with `sizeof (yyGLRState*)`

9. replace `sizeof yynewLookaheadNeeds[0]` and `sizeof yyset->yylookaheadNeeds[0]` with `sizeof (yybool)`
10. replace `sizeof yynewItems[0]` and `sizeof yystackp->yynextFree[0]` with `sizeof (yyGLRStackItem)`

If you are familiar with "diff" and "patch" and if you have clang-format, you can automate these code changes as follows. For this, you must use exactly version 3.2.3 of bison to ensure that the code that is produced is as similar as possible to what is currently in the VTK repository.

1. Run bison (as above) on the vtkParse.y from the master branch
2. Use clang-format-8 to re-style vtkParse.tab.c to match VTK code style
3. Use "git diff -R vtkParse.tab.c" to produce a patch file

If done correctly, this will produce a patch file that contains all the changes above (steps 3 through 9 in the original list). Load the patch file into a text editor to verify that this is so, and remove any superfluous changes from the patch file.

Then, switch to your new vtkParse.y (the one you have modified). Repeat steps 1 and 2 (generate vtkParse.tab.c and reformat it with clang-format). Now you can apply the patch file to automate the original steps 3 through 9. Note that as you continue to edit vtkParse.y and regenerate vtkParse.tab.c, you can continue to use the same patch. Just remember to run clang-format every time that you run bison.

Debugging the Parser

When bison is run, it should not report any shift/reduce or reduce/reduce warnings. If modifications to the rules cause these warnings to occur, you can run bison with the `--debug` and `--verbose` options:

```
bison --debug --verbose -b vtkParse vtkParse.y
```

This will cause bison to produce a file called "vtkParse.output" that will show which rules conflict with other rules.

9.12 Migration Guides

9.12.1 Module Migration from VTK 8.2 to 9+

VTK 8.2 and older contained a module system which was based on variables and informed CMake's migration to target-based properties and interactions. This was incompatible with the way VTK ended up doing it. With VTK 9, its module system has been reworked to use CMake's targets.

This document may be used as a guide to updating code using old VTK modules into code using new VTK modules.

Using modules

If your project is just using VTK's modules and not declaring any of your own modules, porting involves a few changes to the way VTK is found and used.

The old module system made variables available for using VTK.

```
find_package(VTK
  REQUIRED
  COMPONENTS
    vtkCommonCore
```

(continues on next page)

(continued from previous page)

```
    vtkRenderingOpenGL2)
include(${VTK_USE_FILE})

add_library(usesvtk ...)
target_link_libraries(usesvtk ${visibility} ${VTK_LIBRARIES})
target_include_directories(usesvtk ${visibility} ${VTK_INCLUDE_DIRS})

# Pass any VTK autoinit defines to the target.
target_compile_definitions(usesvtk PRIVATE ${VTK_DEFINITIONS})
```

This causes problems if VTK is found multiple times within a source tree with different components. The new pattern is:

```
find_package(VTK
  #9.0 # Compatibility support is not provided if 9.0 is requested.
  REQUIRED
  COMPONENTS
    # Old component names are OK, but deprecated.
    #vtkCommonCore
    #vtkRenderingOpenGL2
    # New names reflect the target names in use.
    CommonCore
    RenderingOpenGL2)
# No longer needed; warns or errors depending on the version requested when
# finding VTK.
#include(${VTK_USE_FILE})

add_library(usesvtk ...)
# VTK_LIBRARIES is provided for compatibility, but not recommended.
#target_link_libraries(usesvtk ${visibility} ${VTK_LIBRARIES})
target_link_libraries(usesvtk ${visibility} VTK::CommonCore VTK::RenderingOpenGL2)

# Rather than defining a single `VTK_DEFINITIONS` for use by all relevant
# targets, the definitions are made as needed with the exact set needed for the
# listed modules.
vtk_module_autoinit(
  TARGETS usesvtk
  #MODULES ${VTK_LIBRARIES} # Again, works, but is not recommended.
  MODULES VTK::CommonCore VTK::RenderingOpenGL2)
```

Module declaration

The old module system had CMake code declare modules in `module.cmake` files. This allowed logic and other things to happen within them which could cause module dependencies to be hard to follow. The new module system now provides facilities for disabling modules in certain configurations (using `CONDITION`) and for optionally depending on modules (using `OPTIONAL_DEPENDS`).

```
if (NOT SOME_OPTION)
  set(depends)
  if (SOME_OTHER_OPTION)
    list(APPEND depends vtkSomeDep)
```

(continues on next page)

(continued from previous page)

```
  endif ()
  vtk_module(vtkModuleName
    GROUPS
      # groups the module belongs to
    KIT
      # the kit the module belongs to
    IMPLEMENTS
      # modules containing vtkObjectFactory instances that are implemented here
    DEPENDS
      # public dependencies
      #${depends} # no analogy in the new system
    PRIVATE_DEPENDS
      # private dependencies
      ${depends}
    COMPILE_DEPENDS
      # modules which must be built before this one but which are not actually
      # linked.
    TEST_DEPENDS
      # test dependencies
    TEST_OPTIONAL_DEPENDS
      # optional test dependencies
      ${depends}
    #EXCLUDE_FROM_WRAPPING
      # present for modules which cannot be wrapped
  )
endif ()
```

This is now replaced with a declarative file named `vtk.module`. This file is not CMake code and is instead parsed as an argument list in CMake (variable expansions are also not allowed). The above example would translate into:

```
MODULE
  vtkModuleName
CONDITION
  SOME_OPTION
GROUPS
  # groups the module belongs to
KIT
  # the kit the module belongs to
#IMPLEMENTABLE # Implicit in the old build system. Now explicit.
IMPLEMENTS
  # modules containing vtkObjectFactory instances that are implemented here
DEPENDS
  # public dependencies
PRIVATE_DEPENDS
  # private dependencies
OPTIONAL_DEPENDS
  vtkSomeDep
ORDER_DEPENDS
  # modules which must be built before this one but which are not actually
  # linked.
TEST_DEPENDS
  # test dependencies
```

(continues on next page)

(continued from previous page)

```
TEST_OPTIONAL_DEPENDS
  # optional test dependencies
  vtkSomeDep
#EXCLUDE_WRAP
  # present for modules which cannot be wrapped
```

Modules may also now be provided by the current project or by an external project found by `find_package` as well.

Declaring sources

Sources used to be listed just as `.cxx` files. The module system would then search for a corresponding `.h` file, then add it to the list. Some source file properties could be used to control header-only or private headers.

In this example, we have a module with the following sources:

- `vtkPublicClass.cxx` and `vtkPublicClass.h`: Public VTK class meant to be wrapped and its header installed.
- `vtkPrivateClass.cxx` and `vtkPrivateClass.h`: Private VTK class not meant for use outside of the module.
- `helper.cpp` and `helper.h`: Private API, but not following VTK's naming conventions.
- `public_helper.cpp` and `public_helper.h`: Public API, but not following VTK's naming conventions.
- `vtkImplSource.cxx`: A source file without a header.
- `public_header.h`: A public header without a source file.
- `template.tcc` and `template.h`: Public API, but not following VTK's naming conventions.
- `private_template.tcc` and `private_template.h`: Private API, but not following VTK's naming conventions.
- `vtkPublicTemplate.txx` and `vtkPublicTemplate.h`: Public template sources. Wrapped and installed.
- `vtkPrivateTemplate.txx` and `vtkPrivateTemplate.h`: Private template sources.
- `vtkOptional.cxx` and `vtkOptional.h`: Private API which requires an optional dependency.

The old module's way of building these sources is:

```
set(Module_SRCS
  vtkPublicClass.cxx
  vtkPrivateClass.cxx
  helper.cpp
  helper.h
  public_helper.cpp
  public_helper.h
  public_header.h
  vtkImplSource.cxx
  vtkPublicTemplate.txx
  vtkPrivateTemplate.txx
  template.tcc # Not detected as a template, so not installed.
  template.h
  private_template.tcc
  private_template.h
)
```

(continues on next page)

(continued from previous page)

```
# Mark some files as only being header files.
set_source_files_properties(
  public_header.h
  HEADER_FILE_ONLY
)

# Mark some headers as being private.
set_source_files_properties(
  helper.h
  private_template.h
  public_header.h
  template.h
  vtkImplSource.cxx # no header
  vtkPrivateTemplate.h
  PROPERTIES SKIP_HEADER_INSTALL 1
)

set(${vtk-module}_HDRS # Magic variable
  public_helper.h
  template.h
  #helper.h # private headers just go ignored.
)

# Optional dependencies are detected through variables.
if (Module_vtkSomeDep)
  list(APPEND Module_SRCS
    # Some optional file.
    vtkOptional.cxx)
endif ()

vtk_module_library(vtkModuleName ${Module_SRCS})
```

While with the new system, source files are explicitly declared using argument parsing.

```
set(classes
  vtkPublicClass)
set(private_classes
  vtkPrivateClass)
set(sources
  helper.cpp
  public_helper.cpp
  vtkImplSource.cxx)
set(headers
  public_header.h
  public_helper.h
  template.h)
set(private_headers
  helper.h
  private_template.h)

set(template_classes
  vtkPublicTemplate)
```

(continues on next page)

(continued from previous page)

```
set(private_template_classes
  vtkPrivateTemplate)
set(templates
  template.tcc)
set(private_templates
  private_template.tcc)

# Optional dependencies are detected as targets.
if (TARGET vtkSomeDep)
  # Optional classes may not be public (though there's no way to actually
  # enforce it, optional dependencies are always treated as private.
  list(APPEND private_classes
    vtkOptional)
endif ()

vtk_module_add_module(vtkModuleName
  # File pairs which follow VTK's conventions. The headers will be wrapped and
  # installed.
  CLASSES ${classes}
  # File pairs which follow VTK's conventions, but are not for use outside the
  # module.
  PRIVATE_CLASSES ${private_classes}
  # Standalone sources (those without headers or which do not follow VTK's
  # conventions).
  SOURCES ${sources}
  # Standalone headers (those without sources or which do not follow VTK's
  # conventions). These will be installed.
  HEADERS ${public_headers}
  # Standalone headers (those without sources or which do not follow VTK's
  # conventions), but are not for use outside the module.
  PRIVATE_HEADERS ${private_headers}

  # Templates are also supported.

  # Template file pairs which follow VTK's conventions. Both files will be
  # installed (only the headers will be wrapped).
  TEMPLATE_CLASSES ${template_classes}
  # Template file pairs which follow VTK's conventions, but are not for use
  # outside the module.
  PRIVATE_TEMPLATE_CLASSES ${private_template_classes}
  # Standalone template files (those without headers or which do not follow
  # VTK's conventions). These will be installed.
  TEMPLATES ${templates}
  # Standalone template files (those without headers or which do not follow
  # VTK's conventions), but are not for use outside the module.
  PRIVATE_TEMPLATES ${private_templates}
)
```

Note that the arguments with `CLASSES` in their name expand to pairs of files with the `.h` and either `.cxx` or `.txx` extension based on whether it is a template or not. Projects not using this convention may use the `HEADERS`, `SOURCES`, and `TEMPLATES` arguments instead.

Object Factories

Previously, object factories were made using implicit variable declaration magic behind the scenes. This is no longer the case and proper CMake APIs for them are available.

```
set(sources
  vtkObjectFactoryImpl.cxx
  # This path is made by `vtk_object_factory_configure` later.
  "${CMAKE_CURRENT_BINARY_DIR}/${vtk-module}ObjectFactory.cxx")

# Make a list of base classes we will be overriding.
set(overrides vtkObjectFactoryBase)
# Make a variable declaring what the override for the class is.
set(vtk_module_vtkObjectFactoryBase_override "vtkObjectFactoryImpl")
# Generate a source using the list of base classes overridden.
vtk_object_factory_configure("${overrides}")

vtk_module_library("${vtk-module}" "${sources}")
```

This is now handled using proper APIs instead of variable lookups.

```
set(classes
  vtkObjectFactoryImpl)

# Explicitly declare the override relationship.
vtk_object_factory_declare(
  BASE      vtkObjectFactoryBase
  OVERRIDE  vtkObjectFactoryImpl)
# Collects the set of declared overrides and writes out a source file.
vtk_object_factory_declare(
  # The path to the source is returned as a variable.
  SOURCE_FILE factory_source
  # As is its header file.
  HEADER_FILE factory_header
  # The export macro is now explicitly passed (instead of assumed based on the
  # current module context).
  EXPORT_MACRO MODULE_EXPORT)

vtk_module_add_module(vtkModuleName
  CLASSES ${classes}
  SOURCES "${factory_source}"
  PRIVATE_HEADERS "${factory_header}")
```

Building a group of modules

This was not well supported in the old module system. Basically, it involved setting up the source tree like VTK expects and then including the `vtkModuleTop` file. This is best just rewritten using the following CMake APIs:

- *vtk_module_find_modules()*
- *vtk_module_find_kits()*
- *vtk_module_scan()*
- *vtk_module_build()*

CHAPTER

TEN

DESIGN DOCUMENTS

10.1 VTK File Formats

A lot of this material is taken from The VTK User's Guide.

The *Visualization Toolkit* provides a number of source and writer objects to read and write popular data file formats. The *Visualization Toolkit* also provides some of its own file formats. The main reason for creating yet another data file format is to offer a consistent data representation scheme for a variety of dataset types, and to provide a simple method to communicate data between software. Whenever possible, we recommend that you use formats that are more widely used. But if this is not possible, the *Visualization Toolkit* formats described here can be used instead. Note that these formats may not be supported by many other tools.

There are three different styles of file formats available in VTK:

1. Legacy

It's a serial formats that are easy to read and write either by hand or programmatically.

1. XML

More flexible but more complex than the legacy file format, it supports random access, parallel I/O, and portable data compression and are preferred to the serial VTK file formats whenever possible.

2. VTKHDF

This is a file format using the same concepts as the XML formats described above but relying on HDF5 for actual storage. It is simpler than the XML. It provides good I/O performance as well as robust and flexible parallel I/O capabilities and may to replace others file formats once it will be complete. It can be read/written using either hdf5 directly or the vtkhdf implementation in VTK.

10.1.1 Simple Legacy Formats

The legacy VTK file formats consist of five basic parts.

1. The first part is the file version and identifier. This part contains the single line: `vtk DataFile Version x.x.` This line must be exactly as shown with the exception of the version number x.x, which will vary with different releases of VTK. (Note: the current version number is 3.0. Version 1.0 and 2.0 files are compatible with version 3.0 files.)
2. The second part is the header. The header consists of a character string terminated by end-of-line character \n. The header is 256 characters maximum. The header can be used to describe the data and include any other pertinent information.
3. The next part is the file format. The file format describes the type of file, either ASCII or binary. On this line the single word ASCII or BINARY must appear.

4. The fourth part is the dataset structure. The geometry part describes the geometry and topology of the dataset. This part begins with a line containing the keyword *DATASET* followed by a keyword describing the type of dataset.Then, depending upon the type of dataset, other keyword/data combinations define the actual data.
5. The final part describes the dataset attributes. This part begins with the keywords *POINT_DATA* or *CELL_DATA*, followed by an integer number specifying the number of points or cells, respectively. (It doesn't matter whether *POINT_DATA* or *CELL_DATA* comes first.) Other keyword/data combinations then define the actual dataset attribute values (i.e., scalars, vectors, tensors, normals, texture coordinates, or field data).

An overview of the file format is shown in Figure 1:

Figure 1: Overview of five parts of VTK data file format.

The first three parts are mandatory, but the other two are optional. Thus you have the flexibility of mixing and matching dataset attributes and geometry, either by operating system file manipulation or using VTK filters to merge data. Keywords are case insensitive, and may be separated by whitespace. Before describing the data file formats please note the following.

- *dataType* is one of the types *bit*, *unsigned_char*, *char*, *unsigned_short*, *short*, *unsigned_int*, *int*, *unsigned_long*, *long*, *float*, or *double*. These keywords are used to describe the form of the data, both for reading from file, as well as constructing the appropriate internal objects. Not all data types are supported for all classes.
- All keyword phrases are written in ASCII form whether the file is binary or ASCII. The binary section of the file (if in binary form) is the data proper; i.e., the numbers that define points coordinates, scalars, cell indices, and so forth.
- Indices are 0-offset. Thus the first point is point id 0.
- If both the data attribute and geometry/topology part are present in the file, then the number of data values defined in the data attribute part must exactly match the number of points or cells defined in the geometry/topology part.
- Cell types and indices are of type *int*.
- Binary data must be placed into the file immediately after the "newline" *(\n)* character from the previous ASCII keyword and parameter sequence.
- The geometry/topology description must occur prior to the data attribute description.

Binary Files

Binary files in VTK are portable across different computer systems as long as you observe two conditions. First, make sure that the byte ordering of the data is correct, and second, make sure that the length of each data type is consistent.

Most of the time VTK manages the byte ordering of binary files for you. When you write a binary file on one computer and read it in from another computer, the bytes representing the data will be automatically swapped as necessary. For example, binary files written on a Sun are stored in big endian order, while those on a PC are stored in little endian order. As a result, files written on a Sun workstation require byte swapping when read on a PC. (See the class vtkByteSwap for implementation details.) The VTK data files described here are written in big endian form.

Some file formats, however, do not explicitly define a byte ordering form. You will find that data read or written by external programs, or the classes vtkVolume16Reader, vtkMCubesReader, and vtkMCubesWriter may have a different byte order depending on the system of origin. In such cases, VTK allows you to specify the byte order by using the methods

```
SetDataByteOrderToBigEndian()
SetDataByteOrderToLittleEndian()
```

Another problem with binary files is that systems may use a different number of bytes to represent an integer or other native type. For example, some 64-bit systems will represent an integer with 8-bytes, while others represent an integer with 4-bytes. Currently, the *Visualization Toolkit* cannot handle transporting binary files across systems with incompatible data length. In this case, use ASCII file formats instead.

Dataset Format

The *Visualization Toolkit* supports five different dataset formats: structured points, structured grid, rectilinear grid, unstructured grid, and polygonal data. Data with implicit topology (structured data such as vtkImageData and vtkStructuredGrid) are ordered with x increasing fastest, then y, then z. These formats are as follows.

- **Structured Points**. The file format supports 1D, 2D, and 3D structured point datasets. The dimensions nx, ny, nz must be greater than or equal to 1. The data spacing sx, sy, sz must be greater than 0. (Note: in the version 1.0 data file, spacing was referred to as "aspect ratio". ASPECT_RATIO can still be used in version 2.0 data files, but is discouraged.) DATASET STRUCTURED_POINTS DIMENSIONS nx ny nz ORIGIN x y z SPACING sx sy yz
- **Structured Grid**. The file format supports 1D, 2D, and 3D structured grid datasets. The dimensions nx, ny, nz must be greater than or equal to 1. The point coordinates are defined by the data in the *POINTS* section. This consists of x-y-z data values for each point. DATASET STRUCTURED_GRID DIMENSIONS nx ny nz POINTS n dataType p0x p0y p0z p1x p1y p1z … p(n-1)x p(n-1)y p(n-1)z
- **Rectilinear Grid**. A rectilinear grid defines a dataset with regular topology, and semi-regular geometry aligned along the x-y-z coordinate axes. The geometry is defined by three lists of monotonically increasing coordinate values, one list for each of the x-y-z coordinate axes. The topology is defined by specifying the grid dimensions, which must be greater than or equal to 1. DATASET RECTILINEAR_GRID DIMENSIONS nx ny nz X_COORDINATES nx dataType x0 x1 … x(nx-1) Y_COORDINATES ny dataType y0 y1 … y(ny-1) Z_COORDINATES nz dataType z0 z1 … z(nz-1)
- **Polygonal Data**. The polygonal dataset consists of arbitrary combinations of surface graphics primitives vertices (and polyvertices), lines (and polylines), polygons (of various types), and triangle strips. Polygonal data is defined by the *POINTS*, *VERTICES*, *LINES*, *POLYGONS*, or *TRIANGLE_STRIPS* sections. The *POINTS* definition is the same as we saw for structured grid datasets. The *VERTICES*, *LINES*, *POLYGONS*, or *TRIANGLE_STRIPS* keywords define the polygonal dataset topology. Each of these keywords requires two parameters: the number of cells n and the size of the cell list size. The cell list size is the total number of integer values required to represent the list (i.e., sum of numPoints and connectivity indices over each cell). None of the keywords *VERTICES*, *LINES*, *POLYGONS*, or *TRIANGLE_STRIPS* is required. DATASET POLYDATA POINTS n dataType p0x p0y p0z p1x p1y p1z … p(n-1)x p(n-1)y p(n-1)z VERTICES n size numPoints0, i0, j0, k0, … numPoints1, i1, j1, k1, … … numPointsn-1, in-1, jn-1, kn-1, … LINES n size numPoints0, i0, j0, k0, … numPoints1, i1, j1, k1, … … numPointsn-1, in-1, jn-1, kn-1, … POLYGONS n size numPoints0, i0, j0, k0, … numPoints1, i1, j1, k1, … … numPointsn-1, in-1, jn-1, kn-1, … TRIANGLE_STRIPS n size numPoints0, i0, j0, k0, … numPoints1, i1, j1, k1, … … numPointsn-1, in-1, jn-1, kn-1, …
- **Unstructured Grid**. The unstructured grid dataset consists of arbitrary combinations of any possible cell type. Unstructured grids are defined by points, cells, and cell types. The CELLS keyword requires two parameters: the number of cells n and the size of the cell list size. The cell list size is the total number of integer values required to represent the list (i.e., sum of numPoints and connectivity indices over each cell). The CELL_TYPES keyword requires a single parameter: the number of cells n. This value should match the value specified by the CELLS keyword. The cell types data is a single integer value per cell that specified cell type (see vtkCell.h or Figure 2). DATASET UNSTRUCTURED_GRID POINTS n dataType p0x p0y p0z p1x p1y p1z … p(n-1)x p(n-1)y p(n-1)z CELLS n size numPoints0, i0, j0, k0, … numPoints1, i1, j1, k1, … numPoints2, i2, j2, k2, … … numPointsn-1, in-1, jn-1, kn-1, … CELL_TYPES n type0 type1 type2 … typen-1
- **Field**. Field data is a general format without topological and geometric structure, and without a particular dimensionality. Typically field data is associated with the points or cells of a dataset. However, if the FIELD type is specified as the dataset type (see Figure1), then a general VTK data object is defined. Use the format described in the next section to define a field. Also see "Working With Field Data" on page 249 and the fourth example in this chapter *Legacy File Examples*.

Dataset Attribute Format

The *Visualization Toolkit* supports the following dataset attributes: scalars (one to four components), vectors, normals, texture coordinates (1D, 2D, and 3D), tensors, and field data. In addition, a lookup table using the RGBA color specification, associated with the scalar data, can be defined as well. Dataset attributes are supported for both points and cells.

Each type of attribute data has a dataName associated with it. This is a character string (without embedded whitespace) used to identify a particular data. The dataName is used by the VTK readers to extract data. As a result, more than one attribute data of the same type can be included in a file. For example, two different scalar fields defined on the dataset points, pressure and temperature, can be contained in the same file. (If the appropriate dataName is not specified in the VTK reader, then the first data of that type is extracted from the file.)

- **Scalars**. Scalar definition includes specification of a lookup table. The definition of a lookup table is optional. If not specified, the default VTK table will be used (and tableName should be "default"). Also note that the numComp variable is optional—by default the number of components is equal to one. (The parameter numComp must range between 1 and 4 inclusive; in versions of VTK prior to 2.3 this parameter was not supported.) SCALARS dataName dataType numComp LOOKUP_TABLE tableName s0 s1 … sn-1 The definition of color scalars (i.e., unsigned char values directly mapped to color) varies depending upon the number of values (nValues) per scalar. If the file format is ASCII, the color scalars are defined using nValues float values between (0,1). If the file format is BINARY, the stream of data consists of nValues unsigned char values per scalar value. COLOR_SCALARS dataName nValues c00 c01 … c0(nValues-1) c10 c11 … c1(nValues-1) … c(n-1)0 c(n-1)1 … c(n-1)(nValues-1)
- **Lookup Table**. The *tableName* field is a character string (without embedded white space) used to identify the lookup table. This label is used by the VTK reader to extract a specific table. Each entry in the lookup table is a rgba[4] (red-green-blue-alpha) array (alpha is opacity where alpha=0 is transparent). If the file format is ASCII, the lookup table values must be float values between (0,1). If the file format is BINARY, the stream of data must be four unsigned char values per table entry. LOOKUP_TABLE tableName size r0 g0 b0 a0 r1 g1 b1 a1 … rsize-1 gsize-1 bsize-1 asize-1
- **Vectors**. VECTORS dataName dataType v0x v0y v0z v1x v1y v1z … v(n-1)x v(n-1)y v(n-1)z
- **Normals**. Normals are assumed normalized |n| = 1. NORMALS dataName dataType n0x n0y n0z n1x n1y n1z … n(n-1)x n(n-1)y n(n-1)z
- **Texture Coordinates**. Texture coordinates of 1, 2, and 3 dimensions are supported. TEXTURE_COORDINATES dataName dim dataType t00 t01 … t0(dim-1) t10 t11 … t1(dim-1) … t(n-1)0 t(n-1)1 … t(n-1)(dim-1)
- **Tensors**. Currently only real-valued, symmetric tensors are supported. TENSORS dataName dataType t000 t001 t002 t010 t011 t012 t020 t021 t022

  t100 t101 t102 t110 t111 t112 t120 t121 t122 …

  tn - 100 tn - 101 tn - 102 tn - 110 tn - 111 tn - 112 tn - 120 tn - 121 tn - 122
- **Field Data**. Field data is essentially an array of data arrays. Defining field data means giving a name to the field and specifying the number of arrays it contains. Then, for each array, the name of the array arrayName(i), the number of components of the array, numComponents, the number of tuples in the array, numTuples, and the data type, dataType, are defined. FIELD dataName numArrays arrayName0 numComponents numTuples dataType f00 f01 … f0(numComponents-1) f10 f11 … f1(numComponents-1) … f(numTuples-1)0 f(numTuples-1)1 … f(numTuples-1)(numComponents-1) arrayName1 numComponents numTuples dataType f00 f01 … f0(numComponents-1) f10 f11 … f1(numComponents-1) … f(numTuples-1)0 f(numTuples-1)1 … f(numTuples-1)(numComponents-1) … arrayName(numArrays-1) numComponents numTuples dataType f00 f01 … f0(numComponents-1) f10 f11 … f1(numComponents-1) … f(numTuples-1)0 f(numTuples-1)1 … f(numTuples-1)(numComponents-1)

Legacy File Examples

The first example is a cube represented by six polygonal faces. We define a single-component scalar, normals, and field data on the six faces. There are scalar data associated with the eight vertices. A lookup table of eight colors, associated with the point scalars, is also defined.

```
# vtk DataFile Version 2.0
Cube example
ASCII
DATASET POLYDATA
POINTS 8 float
0.0 0.0 0.0
1.0 0.0 0.0
1.0 1.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
1.0 0.0 1.0
1.0 1.0 1.0
0.0 1.0 1.0
POLYGONS 6 30
4 0 1 2 3
4 4 5 6 7
4 0 1 5 4
4 2 3 7 6
4 0 4 7 3
4 1 2 6 5
CELL_DATA 6
SCALARS cell_scalars int 1
LOOKUP_TABLE default
0
1
2
3
4
5
NORMALS cell_normals float
0 0 -1
0 0 1
0 -1 0
0 1 0
-1 0 0
1 0 0
FIELD FieldData 2
cellIds 1 6 int
0 1 2 3 4 5
faceAttributes 2 6 float
0.0 1.0 1.0 2.0 2.0 3.0 3.0 4.0 4.0 5.0 5.0 6.0
POINT_DATA 8
SCALARS sample_scalars float 1
LOOKUP_TABLE my_table
0.0
1.0
2.0
```

(continues on next page)

(continued from previous page)

```
3.0
4.0
5.0
6.0
7.0
LOOKUP_TABLE my_table 8
0.0 0.0 0.0 1.0
1.0 0.0 0.0 1.0
0.0 1.0 0.0 1.0
1.0 1.0 0.0 1.0
0.0 0.0 1.0 1.0
1.0 0.0 1.0 1.0
0.0 1.0 1.0 1.0
1.0 1.0 1.0 1.0
```

The next example is a volume of dimension 3 by 4 by 6. Since no lookup table is defined, either the user must create one in VTK, or the default lookup table will be used.

```
# vtk DataFile Version 2.0
Volume example
ASCII
DATASET STRUCTURED_POINTS
DIMENSIONS 3 4 6
ASPECT_RATIO 1 1 1
ORIGIN 0 0 0
POINT_DATA 72
SCALARS volume_scalars char 1
LOOKUP_TABLE default
0 0 0 0 0 0 0 0 0 0 0 0
0 5 10 15 20 25 25 20 15 10 5 0
0 10 20 30 40 50 50 40 30 20 10 0
0 10 20 30 40 50 50 40 30 20 10 0
0 5 10 15 20 25 25 20 15 10 5 0
0 0 0 0 0 0 0 0 0 0 0 0
```

The third example is an unstructured grid containing twelve of the nineteen VTK cell types (see Figure 2 and Figure 3). Figure 2 shows all 16 of the linear cell types and was generated with the LinearCellDemo.

Figure 3 shows 16 of the non-linear cells and was generated with the IsoparametricCellsDemo.

The file contains scalar and vector data. Figure 4 shows a presentation of this file generated by ReadLegacyUnstructuredGrid.

```
# vtk DataFile Version 2.0
Unstructured Grid Example
ASCII
DATASET UNSTRUCTURED_GRID

POINTS 27 float
0 0 0  1 0 0  2 0 0  0 1 0  1 1 0  2 1 0
0 0 1  1 0 1  2 0 1  0 1 1  1 1 1  2 1 1
0 1 2  1 1 2  2 1 2  0 1 3  1 1 3  2 1 3
0 1 4  1 1 4  2 1 4  0 1 5  1 1 5  2 1 5
```

(continues on next page)

(continued from previous page)

```
0 1 6  1 1 6  2 1 6

CELLS 11 60
8 0 1 4 3 6 7 10 9
8 1 2 4 5 7 8 10 11
4 6 10 9 12
4 11 14 10 13
6 15 16 17 14 13 12
6 18 15 19 16 20 17
4 22 23 20 19
3 21 22 18
3 22 19 18
2 26 25
1 24

CELL_TYPES 11
12
11
10
8
7
6
9
5
4
3
1

POINT_DATA 27
SCALARS scalars float 1
LOOKUP_TABLE default
0.0 1.0 2.0 3.0 4.0 5.0
6.0 7.0 8.0 9.0 10.0 11.0
12.0 13.0 14.0 15.0 16.0 17.0
18.0 19.0 20.0 21.0 22.0 23.0
24.0 25.0 26.0

VECTORS vectors float
1 0 0  1 1 0  0 2 0  1 0 0  1 1 0  0 2 0
1 0 0  1 1 0  0 2 0  1 0 0  1 1 0  0 2 0
0 0 1  0 0 1  0 0 1  0 0 1  0 0 1  0 0 1
0 0 1  0 0 1  0 0 1  0 0 1  0 0 1  0 0 1
0 0 1  0 0 1  0 0 1

CELL_DATA 11
SCALARS scalars float 1
LOOKUP_TABLE CellColors
0.0 1.0 2.0 3.0 4.0 5.0
6.0 7.0 8.0 9.0 10.0

LOOKUP_TABLE CellColors 11
.4 .4 1 1
```

(continues on next page)

(continued from previous page)

```
.4 1 .4 1
.4 1 1 1
1 .4 .4 1
1 .4 1 1
1 1 .4 1
1 1 1 1
1 .5 .5 1
.5 1 .5 1
.5 .5 .5 1
1 .5 .4 1
```

The fourth and final example is data represented as a field. You may also wish to see "Working With Field Data" on page 249 to see how to manipulate this data. The data file shown below can be found in its entirety here. The example FinanceFieldData generated Figure 5.

```
# vtk DataFile Version 2.0
Financial data in vtk field format
ASCII
FIELD financialData 6
TIME_LATE 1 3188 float
29.14 0.00 0.00 11.71 0.00 0.00 0.00 0.00
...(more stuff - 3188 total values)...
MONTHLY_PAYMENT 1 3188 float
7.26 5.27 8.01 16.84 8.21 15.75 10.62 15.47
...(more stuff)...
UNPAID_PRINCIPLE 1 3188 float
430.70 380.88 516.22 1351.23 629.66 1181.97 888.91 1437.83
...(more stuff)...
LOAN_AMOUNT 1 3188 float
441.50 391.00 530.00 1400.00 650.00 1224.00 920.00 1496.00
...(more stuff)...
INTEREST_RATE 1 3188 float
13.875 13.875 13.750 11.250 11.875 12.875 10.625 10.500
...(more stuff)...
MONTHLY_INCOME 1 3188 unsigned_short
39 51 51 38 35 49 45 56
...(more stuff)...
```

In this example, a field is represented using six arrays. Each array has a single component and 3,188 tuples. Five of the six arrays are of type float, while the last array is of type unsigned_short. Additional examples are available in the data directory.

10.1.2 XML File Formats

VTK provides another set of data formats using XML syntax. While these formats are much more complicated than the original VTK format described previously (see *Simple Legacy Formats*), they support many more features. The major motivation for their development was to facilitate data streaming and parallel I/O. Some features of the format include support for compression, portable binary encoding, random access, big endian and little endian byte order, multiple file representation of piece data, and new file extensions for different VTK dataset types. XML provides many features as well, especially the ability to extend a file format with application specific tags.

There are two types of VTK XML data files: parallel and serial as described in the following.

- **Serial**. File types designed for reading and writing by applications of only a single process. All of the data are contained within a single file.
- **Parallel**. File types designed for reading and writing by applications with multiple processes executing in parallel. The dataset is broken into pieces. Each process is assigned a piece or set of pieces to read or write. An individual piece is stored in a corresponding serial file type. The parallel file type does not actually contain any data, but instead describes structural information and then references other serial files containing the data for each piece.

In the XML format, VTK datasets are classified into one of two categories.

- **Structured**. The dataset is a topologically regular array of cells such as pixels and voxels (e.g., image data) or quadrilaterals and hexahedra (e.g., structured grid). Rectangular subsets of the data are described through extents. The structured dataset types are vtkImageData, vtkRectilinearGrid, and vtkStructuredGrid.
- **Unstructured**. The dataset forms a topologically irregular set of points and cells. Subsets of the data are describedusing pieces. The unstructured dataset types are vtkPolyData and vtkUnstructuredGrid.

By convention, each data type and file type is paired with a particular file extension. The types and corresponding extensions are

- ImageData (*.vti*) — Serial vtkImageData (structured).
- PolyData (*.vtp*) — Serial vtkPolyData (unstructured).
- RectilinearGrid (*.vtr*) — Serial vtkRectilinearGrid (structured).
- StructuredGrid (*.vts*) — Serial vtkStructuredGrid (structured).
- UnstructuredGrid (*.vtu*) — Serial vtkUnstructuredGrid (unstructured).
- PImageData (*.pvti*) — Parallel vtkImageData (structured).
- PPolyData (*.pvtp*) — Parallel vtkPolyData (unstructured).
- PRectilinearGrid (*.pvtr*) — Parallel vtkRectilinearGrid (structured).
- PStructuredGrid (*.pvts*) — Parallel vtkStructuredGrid (structured).
- PUnstructuredGrid (*.pvtu*) — Parallel vtkUnstructuredGrid (unstructured).

All of the VTK XML file types are valid XML documents.

Note:

There is one case in which the file is not a valid XML document. When the AppendedData section is not encoded as base64, raw binary data is present that may violate the XML specification. This is not default behavior, and must be explicitly enabled by the user.

The document-level element is *VTKFile*:

```
<VTKFile type="ImageData" version="0.1" byte_order="LittleEndian">
...
</VTKFile>
```

The attributes of the element are:

type — The type of the file (the bulleted items in the previous list)..

version — File version number in "major.minor" format.

byte_order — Machine byte order in which data are stored. This is either "BigEndian" or "LittleEndian".

compressor — Some data in the file may be compressed. This specifies the subclass of vtkDataCompressor that was used to compress the data.

Nested inside the *VTKFile* element is an element whose name corresponds to the type of the data format (i.e., the *type* attribute). This element describes the topology the dataset, and is different for the serial and parallel formats, which are described as follows.

Serial XML File Formats

The *VTKFile* element contains one element whose name corresponds to the type of dataset the file describes. We refer to this as the dataset element, which is one of *ImageData*, *RectilinearGrid*, *StructuredGrid*, *PolyData*, or *Unstructured-Grid*. The dataset element contains one or more *Piece* elements, each describing a portion of the dataset. Together, the dataset element and *Piece* elements specify the entire dataset.

Each piece of a dataset must specify the geometry (points and cells) of that piece along with the data associated with each point or cell. Geometry is specified differently for each dataset type, but every piece of every dataset contains *PointData* and *CellData* elements specifying the data for each point and cell in the piece.

The general structure for each serial dataset format is as follows:

ImageData

Each ImageData piece specifies its extent within the dataset's whole extent. The points and cells are described implicitly by the extent, origin, and spacing. Note that the origin and spacing are constant across all pieces, so they are specified as attributes of the *ImageData* XML element as follows.

```
<VTKFile type="ImageData" ...>
  <ImageData WholeExtent="x1 x2 y1 y2 z1 z2"
   Origin="x0 y0 z0" Spacing="dx dy dz">
   <Piece Extent="x1 x2 y1 y2 z1 z2">
      <PointData>...</PointData>
      <CellData>...</CellData>
   </Piece>
   </ImageData>
</VTKFile>
```

RectilinearGrid

Each RectilinearGrid piece specifies its extent within the dataset's whole extent. The points are described by the *Coordinates* element. The cells are described implicitly by the extent.

```
<VTKFile type="RectilinearGrid" ...>
  <RectilinearGrid WholeExtent="x1 x2 y1 y2 z1 z2">
    <Piece Extent="x1 x2 y1 y2 z1 z2">
    <PointData>...</PointData>
    <CellData>...</CellData>
    <Coordinates>...</Coordinates>
    </Piece>
  </RectilinearGrid>
</VTKFile>
```

StructuredGrid

Each StructuredGrid piece specifies its extent within the dataset's whole extent. The points are described explicitly by the Points element. The cells are described implicitly by the extent.

```
<VTKFile type="StructuredGrid" ...>
  <StructuredGrid WholeExtent="x1 x2 y1 y2 z1 z2">
    <Piece Extent="x1 x2 y1 y2 z1 z2">
    <PointData>...</PointData>
    <CellData>...</CellData>
    <Points>...</Points>
    </Piece>
  </StructuredGrid>
</VTKFile>
```

PolyData

Each PolyData piece specifies a set of points and cells independently from the other pieces. The points are described explicitly by the Points element. The cells are described explicitly by the Verts, Lines, Strips, and Polys elements.

```
<VTKFile type="PolyData" ...>
  <PolyData>
    <Piece NumberOfPoints="#" NumberOfVerts="#" NumberOfLines="#"
      NumberOfStrips="#" NumberOfPolys="#">
    <PointData>...</PointData>
    <CellData>...</CellData>
    <Points>...</Points>
    <Verts>...</Verts>
    <Lines>...</Lines>
    <Strips>...</Strips>
    <Polys>...</Polys>
   </Piece>
  </PolyData>
</VTKFile>
```

UnstructuredGrid

Each UnstructuredGrid piece specifies a set of points and cells independently from the other pieces. The points are described explicitly by the Points element. The cells are described explicitly by the Cells element.

```
<VTKFile type="UnstructuredGrid" ...>
  <UnstructuredGrid>
    <Piece NumberOfPoints="#" NumberOfCells="#">
    <PointData>...</PointData>
    <CellData>...</CellData>
    <Points>...</Points>
    <Cells>...</Cells>
    </Piece>
  </UnstructuredGrid>
</VTKFile>
```

Every dataset describes the data associated with its points and cells with PointData and CellData XML elements as follows:

```
  <PointData Scalars="Temperature" Vectors="Velocity">
    <DataArray Name="Velocity" .../>
    <DataArray Name="Temperature" .../>
    <DataArray Name="Pressure" .../>
  </PointData>
```

VTK allows an arbitrary number of data arrays to be associated with the points and cells of a dataset. Each data array is described by a DataArray element which, among other things, gives each array a name. The following attributes of PointData and CellData are used to specify the active arrays by name:

Scalars — The name of the active scalars array, if any.

Vectors — The name of the active vectors array, if any.

Normals — The name of the active normals array, if any.

Tensors — The name of the active tensors array, if any.

TCoords — The name of the active texture coordinates array, if any.

Some datasets describe their points and cells using different combinations of the following common elements:

- **Points** — The *Points* element explicitly defines coordinates for each point individually. It contains one *DataArray* element describing an array with three components per value, each specifying the coordinates of one point.

```
    <Points>
      <DataArray NumberOfComponents="3" .../>
    </Points>
```

- **Coordinates** — The *Coordinates* element defines point coordinates for an extent by specifying the ordinate along each axis for each integer value in the extent's range. It contains three *DataArray* elements describing the ordinates along the x-y-z axes, respectively.

```
    <Coordinates>
      <DataArray .../>
      <DataArray .../>
      <DataArray .../>
    </Coordinates>
```

- **Verts**, **Lines**, **Strips**, and **Polys** — The *Verts*, *Lines*, *Strips*, and *Polys* elements define cells explicitly by specifying point connectivity. Cell types are implicitly known by the type of element in which they are specified. Each element contains two *DataArray* elements. The first array specifies the point connectivity. All the cells' point lists are concatenated together. The second array specifies the offset into the connectivity array for the end of each cell.

```
<Verts>
  <DataArray type="Int32" Name="connectivity" .../>
  <DataArray type="Int32" Name="offsets" .../>
</Verts>
```

- **Cells** — The *Cells* element defines cells explicitly by specifying point connectivity and cell types. It contains three *DataArray* elements. The first array specifies the point connectivity. All the cells' point lists are concatenated together. The second array specifies the offset into the connectivity array for the end of each cell. The third array specifies the type of each cell. (Note: the cell types are defined in Figure 2 and Figure 3.)

```
<Cells>
  <DataArray type="Int32" Name="connectivity" .../>
  <DataArray type="Int32" Name="offsets" .../>
  <DataArray type="UInt8" Name="types" .../>
</Cells>
```

All of the data and geometry specifications use *DataArray* elements to describe their actual content as follows:

- **DataArray** — The *DataArray* element stores a sequence of values of one type. There may be one or more components per value.

```
<DataArray type="Float32" Name="vectors" NumberOfComponents="3"
           format="appended" offset="0"/>
<DataArray type="Float32" Name="scalars" format="binary">
           bAAAAAAAAAAAAIA/AAAAQAAAQEAAAIBA... </DataArray>
<DataArray type="Int32" Name="offsets" format="ascii">
           10 20 30 ... </DataArray>
```

The attributes of the *DataArray* elements are described as follows: type — The data type of a single component of the array. This is one of Int8, UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Float32, Float64. Note: the 64-bit integer types are only supported if VTK_USE_64BIT_IDS is on (a CMake variable—see "CMake" on page 10) or the platform is 64-bit.

Name — The name of the array. This is usually a brief description of the data stored in the array.

NumberOfComponents — The number of components per value in the array.

format — The means by which the data values themselves are stored in the file. This is "ascii", "binary", or "appended".

offset — If the format attribute is "appended", this specifies the offset from the beginning of the appended data section to the beginning of this array's data.

The *format* attribute chooses among the three ways in which data values can be stored:

format="ascii" — The data are listed in ASCII directly inside the *DataArray* element. Whitespace is used for separation.

format="binary" — The data are encoded in base64 and listed contiguously inside the *DataArray* element. Data may also be compressed before encoding in base64. The byte-order of the data matches that specified by the byte_order attribute of the *VTKFile* element.

format="appended" — The data are stored in the appended data section. Since many *DataArray* elements may store their data in this section, the offset attribute is used to specify where each DataArray's data begins. This format is the default used by VTK's writers.

The appended data section is stored in an *AppendedData* element that is nested inside *VTKFile* after the dataset element:

```
<VTKFile ...>
  ...
  <AppendedData encoding="base64">
                _QMwEAAAAAAAAA...
  </AppendedData>
</VTKFile>
```

The appended data section begins with the first character after the underscore inside the *AppendedData* element. The underscore is not part of the data, but is always present. Data in this section is always in binary form, but can be compressed and/or base64 encoded. The byte-order of the data matches that specified by the byte_order attribute of the *VTKFile* element. Each *DataArray*'s data are stored contiguously and appended immediately after the previous *DataArray*'s data without a separator. The *DataArray*'s *offset* attribute indicates the file position offset from the first character after the underscore to the beginning its data.

Parallel File Formats

The parallel file formats do not actually store any data in the file. Instead, the data are broken into pieces, each of which is stored in a serial file of the same dataset type.

The *VTKFile* element contains one element whose name corresponds to the type of dataset the file describes, but with a "P" prefix. We refer to this as the parallel dataset element, which is one of *PImageData*, *PRectilinearGrid*, *PStructuredGrid*, *PPolyData*, or *PUnstructuredGrid*.

The parallel dataset element and those nested inside specify the types of the data arrays used to store points, pointn data, and cell data (the type of arrays used to store cells is fixed by VTK). The element does not actually contain any data, but instead includes a list of *Piece* elements that specify the source from which to read each piece. Individual pieces are stored in the corresponding serial file format. The parallel file needs to specify the type and structural information so that readers can update pipeline information without actually reading the pieces' files.

The general structure for each parallel dataset format is as follows:

PImageData

The *PImageData* element specifies the whole extent of the dataset and the number of ghost-levels by which the extents in the individual pieces overlap. The Origin and Spacing attributes implicitly specify the point locations. Each *Piece* element describes the extent of one piece and the file in which it is stored.

```
<VTKFile type="PImageData" ...>
  <PImageData WholeExtent="x1 x2 y1 y2 z1 z2"
              GhostLevel="#" Origin="x0 y0 z0" Spacing="dx dy dz">
    <PPointData>...</PPointData>
    <PCellData>...</PCellData>
    <Piece Extent="x1 x2 y1 y2 z1 z2" Source="imageData0.vti"/>
    ...
 </PImageData>
</VTKFile>
```

PRectilinearGrid

The *PRectilinearGrid* element specifies the whole extent of the dataset and the number of ghost-levels by which the extents in the individual pieces overlap. The *PCoordinates* element describes the type of arrays used to specify the point ordinates along each axis, but does not actually contain the data. Each *Piece* element describes the extent of one piece and the file in which it is stored.

```
<VTKFile type="PRectilinearGrid" ...>
  <PRectilinearGrid WholeExtent="x1 x2 y1 y2 z1 z2"
                    GhostLevel="#">
    <PPointData>...</PPointData>
    <PCellData>...</PCellData>
    <PCoordinates>...</PCoordinates>
    <Piece Extent="x1 x2 y1 y2 z1 z2"
           Source="rectilinearGrid0.vtr"/>
    ...
  </PRectilinearGrid>
</VTKFile>
```

PStructuredGrid

The *PStructuredGrid* element specifies the whole extent of the dataset and the number of ghost-levels by which the extents in the individual pieces overlap. The *PPoints* element describes the type of array used to specify the point locations, but does not actually contain the data. Each *Piece* element describes the extent of one piece and the file in which it is stored.

```
<VTKFile type="PStructuredGrid" ...>
  <PStructuredGrid WholeExtent="x1 x2 y1 y2 z1 z2"
                   GhostLevel="#">
    <PPointData>...</PPointData>
    <PCellData>...</PCellData>
    <PPoints>...</PPoints>
    <Piece Extent="x1 x2 y1 y2 z1 z2"
           Source="structuredGrid0.vts"/>
    ...
  </PStructuredGrid>
</VTKFile>
```

PPolyData

The *PPolyData* element specifies the number of ghost-levels by which the individual pieces overlap. The *PPoints* element describes the type of array used to specify the point locations, but does not actually contain the data. Each *Piece* element specifies the file in which the piece is stored.

```
<VTKFile type="PPolyData" ...>
  <PPolyData GhostLevel="#">
    <PPointData>...</PPointData>
    <PCellData>...</PCellData>
    <PPoints>...</PPoints>
    <Piece Source="polyData0.vtp"/>
```

(continues on next page)

(continued from previous page)

```
    ...
  </PPolyData>
</VTKFile>
```

PUnstructuredGrid

The *PUnstructuredGrid* element specifies the number of ghost-levels by which the individual pieces overlap. The *PPoints* element describes the type of array used to specify the point locations, but does not actually contain the data. Each *Piece* element specifies the file in which the piece is stored.

```
<VTKFile type="PUnstructuredGrid" ...>
  <PUnstructuredGrid GhostLevel="0">
    <PPointData>...</PPointData>
    <PCellData>...</PCellData>
    <PPoints>...</PPoints>
    <Piece Source="unstructuredGrid0.vtu"/>
    ...
  </PUnstructuredGrid>
</VTKFile>
```

Every dataset uses *PPointData* and *PCellData* elements to describe the types of data arrays associated with its points and cells.

- **PPointData** and **PCellData** — These elements simply mirror the *PointData* and *CellData* elements from the serial file formats. They contain *PDataArray* elements describing the data arrays, but without any actual data.

```
  <PPointData Scalars="Temperature" Vectors="Velocity">
    <PDataArray Name="Velocity" .../>
    <PDataArray Name="Temperature" .../>
    <PDataArray Name="Pressure" .../>
 </PPointData>
```

For datasets that need specification of points, the following elements mirror their counterparts from the serial file format:

- **PPoints** — The *PPoints* element contains one *PDataArray* element describing an array with three components. The data array does not actually contain any data.

```
  <PPoints>
    <PDataArray NumberOfComponents="3" .../>
  </PPoints>
```

- **PCoordinates** — The *PCoordinates* element contains three *PDataArray* elements describing the arrays used to specify ordinates along each axis. The data arrays do not actually contain any data.

```
  <PCoordinates>
    <PDataArray .../>
    <PDataArray .../>
    <PDataArray .../>
 </PCoordinates>
```

All of the data and geometry specifications use *PDataArray* elements to describe the data array types:

- **PDataArray** — The *PDataArray* element specifies the type, Name, and optionally the NumberOfComponents attributes from the *DataArray* element. It does not contain the actual data. This can be used by readers to create the data array in their output without needing to read any real data, which is necessary for efficient pipeline updates in some cases.

```
<PDataArray type="Float32" Name="vectors" NumberOfComponents="3"/>
```

XML File Example

The following is a complete example specifying a vtkPolyData representing a cube with some scalar data on its points and faces. 1

```
<?xml version="1.0"?>
<VTKFile type="PPolyData" version="0.1" byte_order="LittleEndian">
  <PPolyData GhostLevel="0">
    <PPointData Scalars="my_scalars">
      <PDataArray type="Float32" Name="my_scalars"/>
    </PPointData>
      <PCellData Scalars="cell_scalars" Normals="cell_normals">
        <PDataArray type="Int32" Name="cell_scalars"/>
         <PDataArray type="Float32" Name="cell_normals" NumberOfComponents="3"/>
      </PCellData>
      <PPoints>
        <PDataArray type="Float32" NumberOfComponents="3"/>
      </PPoints>
      <Piece Source="polyEx0.vtp"/>
  </PPolyData>
</VTKFile>


<?xml version="1.0"?>
  <VTKFile type="PolyData" version="0.1" byte_order="LittleEndian">
    <PolyData>
      <Piece NumberOfPoints="8" NumberOfVerts="0" NumberOfLines="0"
             NumberOfStrips="0" NumberOfPolys="6">
      <Points>
        <DataArray type="Float32" NumberOfComponents="3" format="ascii">
          0 0 0 1 0 0 1 1 0 0 1 0 0 0 1 1 0 1 1 1 1 0 1 1
        </DataArray>
      </Points>
      <PointData Scalars="my_scalars">
        <DataArray type="Float32" Name="my_scalars" format="ascii">
          0 1 2 3 4 5 6 7
       </DataArray>
      </PointData>
      <CellData Scalars="cell_scalars" Normals="cell_normals">
        <DataArray type="Int32" Name="cell_scalars" format="ascii">
         0 1 2 3 4 5
        </DataArray>
        <DataArray type="Float32" Name="cell_normals"
                   NumberOfComponents="3" format="ascii">
          0 0 -1 0 0 1 0 -1 0 0 1 0 -1 0 0 1 0 0
```

(continues on next page)

(continued from previous page)

```
          </DataArray>
        </CellData>
        <Polys>
          <DataArray type="Int32" Name="connectivity" format="ascii">
             0 1 2 3 4 5 6 7 0 1 5 4 2 3 7 6 0 4 7 3 1 2 6 5
          </DataArray>
          <DataArray type="Int32" Name="offsets" format="ascii">
             4 8 12 16 20 24
          </DataArray>
        </Polys>
      </Piece>
    </PolyData>
  </VTKFile>
```

10.1.3 VTKHDF File Format

The VTKHDF file format is a file format relying on HDF5. It is meant to provide good I/O performance as well as robust and flexible parallel I/O capabilities.

It currently supports: PolyData, UnstructuredGrid, ImageData, OverlappingAMR, MultiBlockDataSet and the PartitionedDataSetCollection.

The current file format version is the **2.2**.

Note: This development is iterative and the format is expected to grow in its support for more and more use cases.

Changelog

VTKHDF - 2.2

- add support for temporal `OverlappingAMR`
- add official support for ignored data outside of `VTKHDF`

VTKHDF - 2.1

- add specification in the format for `PartitionedDataSetCollection` and `MultiBlockDataSet`

VTKHDF - 2.0

- extends the specification for `PolyData`.
- add support for `Temporal` dataset for `PolyData`, `ImageData` and `UnstructuredGrid`.

VTKHDF - 1.0

- add specification for these vtk data types:
  - `UnstructuredGrid`
  - `ImageData`
  - `Overlapping AMR`

Extension

The `VTKHDF` format generally uses the `.vtkhdf` extension. The `.hdf` extension is also supported but is not preferred. There are no specific extensions to differentiate between different types of dataset, serial vs. distributed data or static vs. temporal data.

General Specification

VTK HDF files start with a group called `VTKHDF` with two attributes: `Version`, an array of two integers and `Type`, a string showing the VTK dataset type stored in the file. Additional attributes can follow depending on the dataset type. Currently, `Version` is the array [2, 2] and `Type` can be `ImageData`, `PolyData`, `UnstructuredGrid`, `OverlappingAMR`, `PartitionedDataSetCollection` or `MultiBlockDataSet`.

Top-level groups outside of /VTKHDF do not contain any information related to VTK data model and are outside of the scope of this specification. They can be useful to store meta-information that could be read and written by custom VTKHDF implementations.

The data type for each HDF dataset is part of the dataset and it is determined at write time. The reader matches the type of the dataset with a `H5T_NATIVE_` type and creates the VTK array of that type. Consequently, the type at writing might be different than the type at reading even on the same machine because for instance `long` can be the same type as `long long` or `int` can be the same as `long` on certain platforms. Also, `vtkIdType` is read as the C++ type it represents (`long` or `long long`). Endianness conversions are done automatically.

In the diagrams that follow, showing the HDF file structure for VTK datasets, the rounded blue rectangles are HDF groups and the gray rectangles are HDF datasets. Each rectangle shows the name of the group or dataset in bold font and the attributes underneath with regular font.

Image data

The format for image data is detailed in the Figure 6 where the `Type` attribute of the `VTKHDF` group is `ImageData`. An ImageData (regular grid) is not split into partitions for parallel processing. We rely on the writer to chunk the data to optimize reading for a certain number of MPI ranks. Attribute data is stored in a PointData or CellData array using hyper slabs. `WholeExtent`, `Origin`, `Spacing` and `Direction` attributes have the same meaning as the corresponding attributes for the vtkImageData dataset. `Scalars`, `Vectors`, ... string attributes for the `PointData` and `CellData` groups specify the active attributes in the dataset.

Unstructured grid

The format for unstructured grid is shown in Figure 7. In this case the `Type` attribute of the `VTKHDF` group is `UnstructuredGrid`. The unstructured grid is split into partitions, with a partition for each MPI rank. This is reflected in the HDF5 file structure. Each HDF dataset is obtained by concatenating the data for each partition. The offset O(i) where we store the data for partition i is computed using:

O(i) = S(0) + … + S(i-1), i > 1 with O(0) = 0.

where S(i) is the size of partition i.

We describe the split into partitions using HDF5 datasets `NumberOfConnectivityIds`, `NumberOfPoints` and `NumberOfCells`. Let n be the number of partitions which usually correspond to the number of the MPI ranks. `NumberOfConnectivityIds` has size n where NumberOfConnectivityIds[i] represents the size of the `Connectivity` array for partition i. `NumberOfPoints` and `NumberOfCells` are arrays of size n, where NumberOfPoints[i] and NumberOfCells[i] are the number of points and number of cells for partition i. The `Points` array contains the points of the VTK dataset. `Offsets` is an array of size (S(i) + 1), where S(i) is the number of cells in partition i, indicating the index in the `Connectivity` array where each cell's points start. `Connectivity` stores the lists of point ids for each cell, and `Types` contain the cell information stored as described in vtkCellArray documentation. Data for each partition is appended in a HDF dataset for `Points`, `Connectivity`, `Offsets`, `Types`, `PointData` and `CellData`. We can compute the size of partition i using the following formulas:

	Size of partition i
Points	NumberOfPoints[i] * 3 * sizeof(Points[0][0])
Connectivity	NumberOfConnectivityIds[i] * sizeof(Connectivity[0])
Offsets	(NumberOfCells[i] + 1) * sizeof(Offsets[0])
Types	NumberOfCells[i] * sizeof(Types[i])
PointData	NumberOfPoints[i] * sizeof(point_array_k[0])
CellData	NumberOfCells[i] * sizeof(cell_array_k[0])

To read the data for its rank a node reads the information about all partitions, compute the correct offset and then read data from that offset.

Poly data

The format for unstructured grid is shown in Figure 8. In this case the `Type` attribute of the `VTKHDF` group is `PolyData`. The poly data is split into partitions, with a partition for each MPI rank. This is reflected in the HDF5 file structure. Each HDF dataset is obtained by concatenating the data for each partition. The offset O(i) where we store the data for partition i is computed using:

O(i) = S(0) + … + S(i-1), i > 1 with O(0) = 0.

where S(i) is the size of partition i. This is very similar to and completely inspired by the `UnstructuredGrid` format.

The split into partitions of the point coordinates is exactly the same as in the `UnstructuredGrid` format above. However, the split into partitions of each of the category of cells (`Vertices`, `Lines`, `Polygons` and `Strips`) using HDF5 datasets `NumberOfConnectivityIds` and `NumberOfCells`. Let n be the number of partitions which usually correspond to the number of the MPI ranks. `{CellCategory}/NumberOfConnectivityIds` has size n where NumberOfConnectivityIds[i] represents the size of the `{CellCategory}/Connectivity` array for partition i. `NumberOfPoints` and `{CellCategory}/NumberOfCells` are arrays of size n, where NumberOfPoints[i] and {CellCategory}/NumberOfCells[i] are the number of points and number of cells for partition i. The `Points` array contains the points of the VTK dataset. `{CellCategory}/Offsets` is an array of size (S(i) + 1), where S(i) is the number of cells in partition i, indicating the index in the `{CellCategory}/Connectivity` array where each cell's points start. `{CellCategory}/Connectivity` stores the lists of point ids for each cell. Data for each partition is appended in a

HDF dataset for `Points`, `Connectivity`, `Offsets`, `PointData` and `CellData`. We can compute the size of partition i using the following formulas:

	Size of partition i
Points	NumberOfPoints[i] * 3 * sizeof(Points[0][0])
{CellCategory}/Connectivity	{CellCategory}/NumberOfConnectivityIds[i] * sizeof({CellCategory}/Connectivity[0])
{CellCategory}/Offsets	({CellCategory}/NumberOfCells[i] + 1) * sizeof({CellCategory}/Offsets[0])
PointData	NumberOfPoints[i] * sizeof(point_array_k[0])
CellData	(j {CellCategory_j}/NumberOfCells[i]) * sizeof(cell_array_k[0])

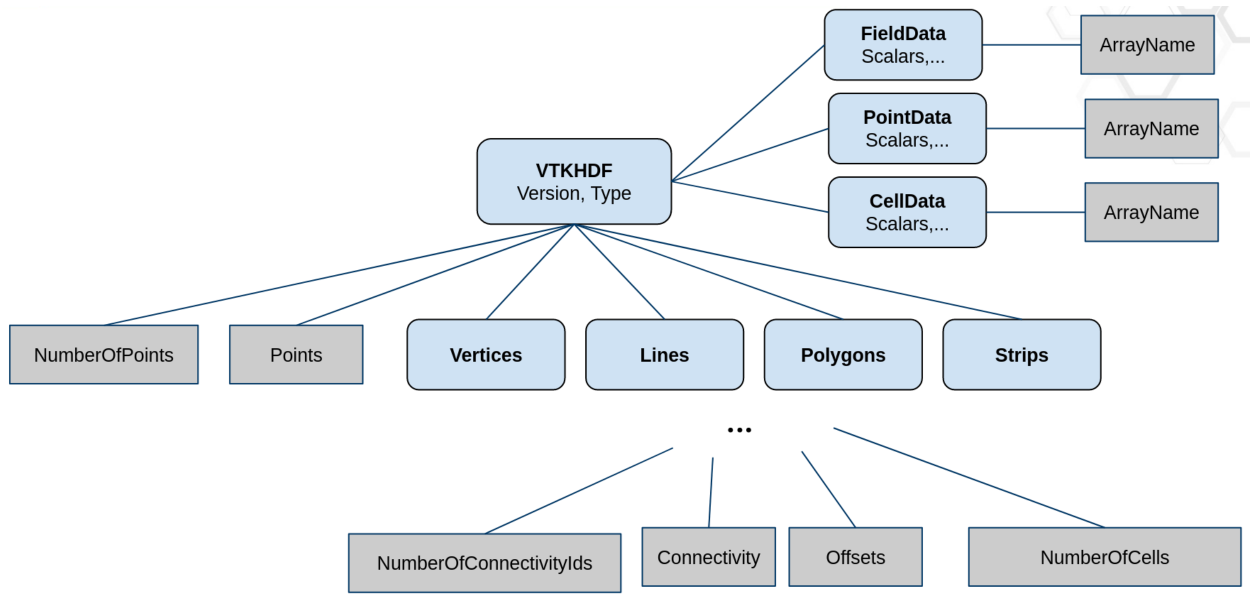


Fig. 1: Figure 8. - Poly Data VTKHDF File Format

To read the data for its rank a node reads the information about all partitions, compute the correct offset and then read data from that offset.

Overlapping AMR

The format for Overlapping AMR is shown in Figure 9. In this case the `Type` attribute of the `VTKHDF` group is `OverlappingAMR`. The mandatory `Origin` parameter is a double triplet that defines the global origin of the AMR data set. Each level in an overlapping AMR file format (and data structure) consists of a list of uniform grids with the same spacing from the Spacing attribute. The Spacing attribute is a list a three doubles describing the spacing in each x/y/z direction. The AMRBox dataset contains the bounding box for each of these grids. Each line in this dataset is expected to contain 6 integers describing the the indexed bounds in i, j, k space (imin/imax/jmin/jmax/kmin/kmax). The points and cell arrays for these grids are stored serialized in one dimension and stored in a dataset in the PointData or CellData group.

PartitionedDataSetCollection and MultiBlockDataSet

The general VTKHDF format for vtkPartitionedDataSetCollection (PDC) and vtkMultiBlockDataSet (MB) is shown in Figure 10.

Both VTK data types share a common structure, with a few notable differences. The `Type` attribute of the `VTKHDF` group for them should be `PartitionedDataSetCollection` or `MultiBlockDataSet`. The most important element in this design is the `Assembly` group, direct child of the `VTKHDF` group. This group describes the composite data hierarchy. The elements of the Assembly group do not contain the data directly. Instead, the data blocks are stored as direct children of the `VTKHDF` group, without any hierarchy, and any node in the Assembly group can use an HDF5 symbolic link to the top-level datasets.

Here lies the main distinction between the PDC and MB formats. For PDC, a group in the assembly that is not a softlink represents a node in the vtkAssembly associated to it, and a softlink represents a dataset index associated to its parent node (similar to what the function `AddDataSetIndex` does in `vtkDataAssembly`). This way, a single dataset can be used multiple times in the assembly without any additional storage cost. Top-level datasets need to set an `Index` attribute to specify their index in the PDC flat structure. On the other hand, MB structures work a little differently. First, they don't need no index for their datasets, and secondly, an assembly node that is not a softlink represents a nested `vtkMultiBlockDataSet`. A softlink in the assembly represents a dataset nested in its parent `vtkMultiBlockDataSet`. Again, this MB format can save space when a block is referenced multiple times.

Some additional detail about the format:

- The data blocks should not be composite themselves : they can only be simple or partitioned types, but not using an assembly.
- The Assembly group and its children need to track creation order to be able to keep subtrees ordered. For this, you need to set H5G properties `H5P_CRT_ORDER_TRACKED` and `H5P_CRT_ORDER_INDEXED` on each group when writing the Assembly.
- For PDC, the assembly structure only needs to be traversed once at the beginning of the reading procedure (and can potentially be read and broadcasted only by the main process in a distributed context) to optimize file meta-data reading.
- The block wise reading implementation and composite level implementation can be managed independently from each other.
- It would be doable for each block to have its own time range and time steps in a temporal context with the full composite data set able to collect and expose a combined range and set of time values, but for now we only allow reading datasets that have all the same number of timesteps.
- Reading performance can scale linearly with the number of blocks even in a distributed context.

Temporal Data

The generic format for all `VTKHDF` temporal data is shown in Figure 11. The general idea is to take the static formats described above and use them as a base to append all the time dependent data. As such, a file holding static data has a very similar structure to a file holding dynamic data. An additional `Steps` subgroup is added to the `VTKHDF` main group holding offset information for each of the time steps as well as the time values. The choice to include offset information as HDF5 datasets was made to reduce the quantity of meta-data in the file to improve performance. This `Steps` group has one integer like attribute `NSteps` indicating the number of steps in the temporal dataset.

The `Steps` group is structured as follows:

- `Values` [dim = (NSteps)]: each entry indicates the time value for the associated time step.
- `PartOffsets` [dims = (NSteps)]: each entry indicates at which part offset to start reading the associated time step (relevant for `Unstructured Grids` and `Poly Data`).

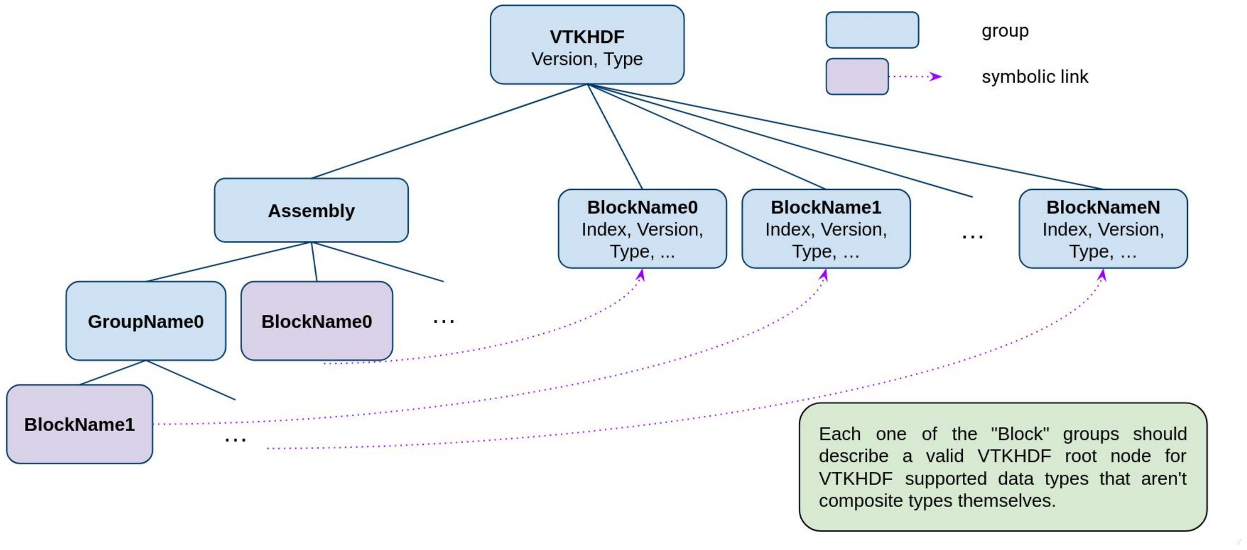

VTKHDF
Version, Type
group
symbolic link
Assembly
BlockName0
Index, Version, Type, ...
BlockName1
Index, Version, Type, ...
...
BlockNameN
Index, Version, Type, ...
GroupName0
BlockName0
...
BlockName1
...
Each one of the "Block" groups should describe a valid VTKHDF root node for VTKHDF supported data types that aren't composite types themselves.

Fig. 2: Figure 10. - PartitionedDataSetCollection/MultiBlockDataset VTKHDF File Format

- `NumberOfParts` [dims = (NSteps)]: each entry indicates how many parts the associated time step has (relevant for `Unstructured Grids` and `Poly Data`). This information is optional if there is a constant number of parts per time steps and the length of `VTKHDF/NumberOfPoints` is equal to `NumberOfPartsPerTimeStep x NSteps`.
- `PointOffsets` [dims = (NSteps)]: each entry indicates where in the `VTKHDF/Points` data set to start reading point coordinates for the associated time step (relevant for `Unstructured Grid` and `Poly Data`).
- `CellOffsets` [dims = (NSteps, NTopologies)]: each entry indicates by how many cells to offset reading into the connectivity offset structures for the associated time step (relevant for `Unstructured Grid` and `Poly Data`).
  - `Unstructured Grids` only have one set of connectivity data and NTopologies = 1.
  - `Poly Data`, however, have `Vertices`,`Lines`, `Polygons` and `Strips` in that order and therefore NTopologies = 4.
- `ConnectivityIdOffsets` [dims = (NSteps, NTopologies)]: each entry indicates by how many values to offset reading into the connectivity indexing structures for the associated time step (relevant for `Unstructured Grid` and `Poly Data`).
  - `Unstructured Grids` only have one set of connectivity data and NTopologies = 1.
  - `Poly Data`, however, have `Vertices`,`Lines`, `Polygons` and `Strips` in that order and therefore NTopologies = 4.
- `{Point,Cell,Field}DataOffsets/{ArrayName}` [dims = (NSteps)]: each entry indicates by how many values to offset reading into the given array for the associated time step. In the absence of a data set, the appropriate geometry offsetting for the time step is used in its place.
- `FieldDataSizes/{ArrayName}` [dims = (NSteps, 2)]: each entry indicates the field data component and tuple size. In the absence of a data set, the maximum number of components and one tuple per step are considered.

Writing incrementally to `VTKHDF` temporal datasets is relatively straightforward using the appending functionality of `HDF5` chunked data sets (Chunking in HDF5).

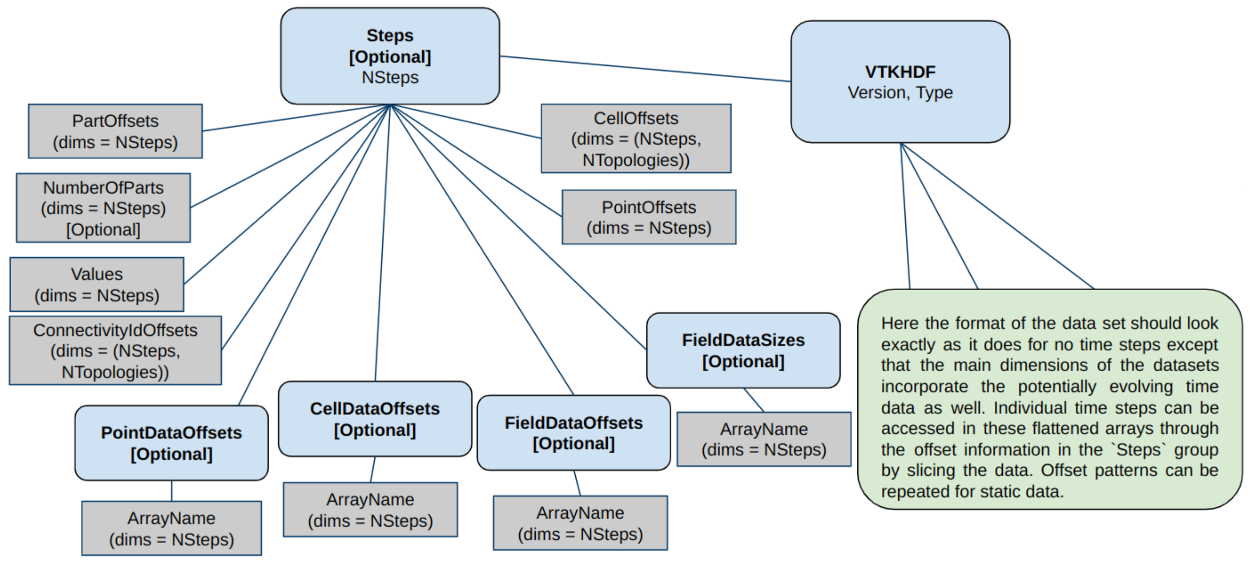

Steps
[Optional]
NSteps
VTKHDF
Version, Type
PartOffsets
(dims = NSteps)
NumberOfParts
(dims = NSteps)
[Optional]
Values
(dims = NSteps)
ConnectivityIdOffsets
(dims = (NSteps, NTopologies))
CellOffsets
(dims = (NSteps, NTopologies))
PointOffsets
(dims = NSteps)
FieldDataSizes
[Optional]
ArrayName
(dims = NSteps)
PointDataOffsets
[Optional]
ArrayName
(dims = NSteps)
CellDataOffsets
[Optional]
ArrayName
(dims = NSteps)
FieldDataOffsets
[Optional]
ArrayName
(dims = NSteps)
Here the format of the data set should look exactly as it does for no time steps except that the main dimensions of the datasets incorporate the potentially evolving time data as well. Individual time steps can be accessed in these flattened arrays through the offset information in the `Steps` group by slicing the data. Offset patterns can be repeated for static data.

Fig. 3: Figure 11. - Temporal Data VTKHDF File Format

Particularity regarding ImageData

A particularity of temporal `Image Data` in the format is that the reader expects an additional prepended dimension considering the time to be the first dimension in the multidimensional arrays. As such, arrays described in temporal `Image Data` should have dimensions ordered as `(time, z, y, x)`.

Particularity regarding OverlappingAMR

Currently only `AMRBox` and `Point/Cell/Field data` can be temporal, not the `Spacing`. Due to the structure of the OverlappingAMR format, the format specify an intermediary group between the `Steps` group and the `Point/Cell/FieldDataOffset` group named `LevelX` for each level where `X` is the number of level. These `Level` groups will also contain 2 other datasets to retrieve the `AMRBox`:

- `AMRBoxOffsets` : each entry indicates by how many AMR box to offset reading into the `AMRBox`.
- `NumberOfAMRBox` : the number of boxes contained in the `AMRBox` for each timestep.

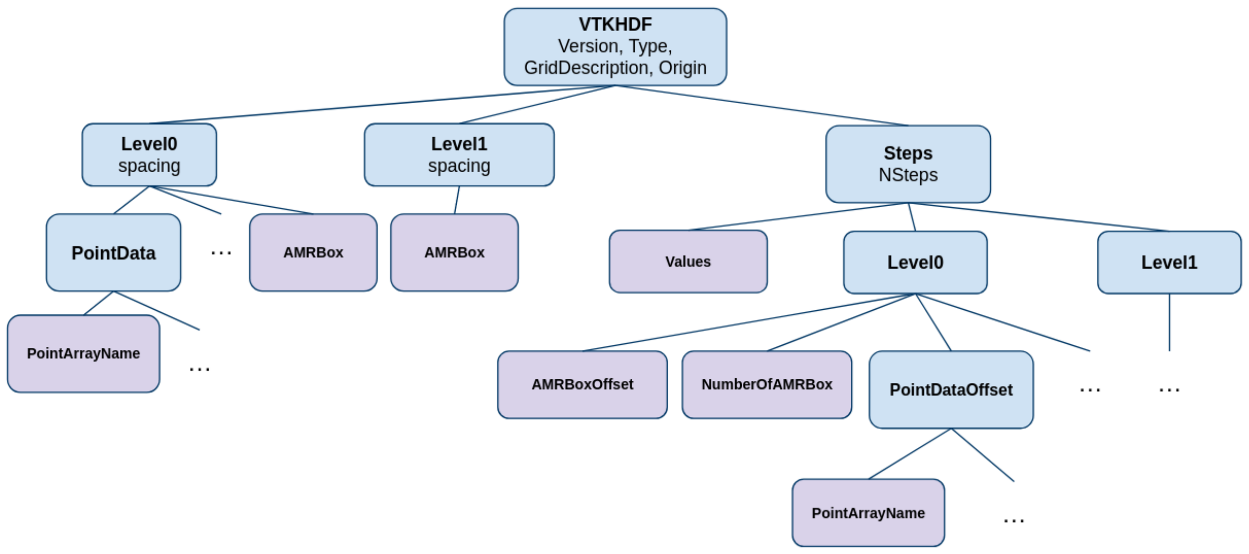


Fig. 4: Figure 12. - Temporal OverlappingAMR VTKHDF File Format

Limitations

This specification and the reader available in VTK currently only supports ImageData, UnstructuredGrid, PolyData, Overlapping AMR, MultiBlockDataSet and Partitioned DataSet Collection. Other dataset types may be added later depending on interest and funding.

Examples

We present three examples of VTK HDF files, shown using h5dump -A one image file, one unstructured grid and one overlapping AMR. These files can be examined in the VTK source code, by building VTK and enabling testing (`VTK_BUILD_TESTING`). The two files are in the build directory ExternalData at `Testing/Data/mandelbrot-vti.hdf` for the ImageData and at `Testing/Data/can-pvtu.hdf` for the partitioned UnstructuredGrid and `Testing/Data/amr_gaussian_pulse.hdf` for the overlapping AMR.

ImageData

The image data file is a wavelet source produced in ParaView. Note that we don't partition image data, so the same format is used for serial and parallel processing.

```
HDF5 "ExternalData/Testing/Data/mandelbrot-vti.hdf" {
GROUP "/" {
   GROUP "VTKHDF" {
      ATTRIBUTE "Direction" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 9 ) / ( 9 ) }
         DATA {
         (0): 1, 0, 0, 0, 1, 0, 0, 0, 1
         }
      }
      ATTRIBUTE "Origin" {
```

(continues on next page)

(continued from previous page)

```
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
         DATA {
         (0): -1.75, -1.25, 0
         }
      }
      ATTRIBUTE "Spacing" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
         DATA {
         (0): 0.131579, 0.125, 0.0952381
         }
      }
      ATTRIBUTE "Type" {
         DATATYPE  H5T_STRING {
            STRSIZE 9;
            STRPAD H5T_STR_NULLPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SCALAR
         DATA {
         (0): "ImageData"
         }
      }
      ATTRIBUTE "Version" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         DATA {
         (0): 1, 0
         }
      }
      ATTRIBUTE "WholeExtent" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 6 ) / ( 6 ) }
         DATA {
         (0): 0, 19, 0, 20, 0, 21
         }
      }
      GROUP "PointData" {
         ATTRIBUTE "Scalars" {
            DATATYPE  H5T_STRING {
               STRSIZE 18;
               STRPAD H5T_STR_NULLPAD;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SCALAR
            DATA {
            (0): "IterationsGradient"
            }
         }
```

(continues on next page)

(continued from previous page)

```
         DATASET "Iterations" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
         DATASET "IterationsGradient" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 22, 21, 20, 3 ) / ( 22, 21, 20, 3 ) }
         }
         DATASET "Iterations_double" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
         DATASET "point_index_llong" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
         DATASET "xextent_int" {
            DATATYPE  H5T_STD_I32LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
         DATASET "xextent_long" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
         DATASET "xextent_uint" {
            DATATYPE  H5T_STD_U32LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
         DATASET "xextent_ulong" {
            DATATYPE  H5T_STD_U64LE
            DATASPACE  SIMPLE { ( 22, 21, 20 ) / ( 22, 21, 20 ) }
         }
      }
   }
}
}
```

UnstructuredGrid

The unstructured grid is the can example (only the can, not the brick) from ParaView, partitioned in three:

```
HDF5 "ExternalData/Testing/Data/can-pvtu.hdf" {
GROUP "/" {
   GROUP "VTKHDF" {
      ATTRIBUTE "Type" {
         DATATYPE  H5T_STRING {
            STRSIZE 16;
            STRPAD H5T_STR_NULLPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
```

(continues on next page)

(continued from previous page)

```
         DATASPACE  SCALAR
         DATA {
         (0): "UnstructuredGrid"
         }
      }
      ATTRIBUTE "Version" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         DATA {
         (0): 1, 0
         }
      }
      GROUP "CellData" {
         DATASET "EQPS" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 5480 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "vtkGhostType" {
            DATATYPE  H5T_STD_U8LE
            DATASPACE  SIMPLE { ( 5480 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "vtkOriginalCellIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 5480 ) / ( H5S_UNLIMITED ) }
         }
      }
      DATASET "Connectivity" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 43840 ) / ( H5S_UNLIMITED ) }
      }
      GROUP "FieldData" {
         DATASET "ElementBlockIds" {
            DATATYPE  H5T_STD_I32LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
         DATASET "Info_Records" {
            DATATYPE  H5T_STD_I8LE
            DATASPACE  SIMPLE { ( 4, 81 ) / ( 4, 81 ) }
         }
         DATASET "KE" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
         }
         DATASET "NSTEPS" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
         }
         DATASET "QA_Records" {
            DATATYPE  H5T_STD_I8LE
            DATASPACE  SIMPLE { ( 24, 33 ) / ( 24, 33 ) }
         }
         DATASET "TMSTEP" {
```

(continues on next page)

(continued from previous page)

```
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
         }
         DATASET "TimeValue" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
         DATASET "Title" {
            DATATYPE  H5T_STRING {
               STRSIZE H5T_VARIABLE;
               STRPAD H5T_STR_NULLTERM;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
         DATASET "XMOM" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
         }
         DATASET "YMOM" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
         }
         DATASET "ZMOM" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
         }
      }
      DATASET "NumberOfCells" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
      }
      DATASET "NumberOfConnectivityIds" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
      }
      DATASET "NumberOfPoints" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
      }
      DATASET "Offsets" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 5483 ) / ( H5S_UNLIMITED ) }
      }
      GROUP "PointData" {
         DATASET "ACCL" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 8076, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
         DATASET "DISPL" {
            DATATYPE  H5T_IEEE_F64LE
```

(continues on next page)

(continued from previous page)

```
            DATASPACE  SIMPLE { ( 8076, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
         DATASET "GlobalNodeId" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 8076 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "PedigreeNodeId" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 8076 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "VEL" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 8076, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
         DATASET "vtkGhostType" {
            DATATYPE  H5T_STD_U8LE
            DATASPACE  SIMPLE { ( 8076 ) / ( H5S_UNLIMITED ) }
         }
      }
      DATASET "Points" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 8076, 3 ) / ( H5S_UNLIMITED, 3 ) }
      }
      DATASET "Types" {
         DATATYPE  H5T_STD_U8LE
         DATASPACE  SIMPLE { ( 5480 ) / ( H5S_UNLIMITED ) }
      }
   }
}
}
```

PolyData

The poly data is the `test_poly_data.hdf` from the VTK testing data:

```
HDF5 "ExternalData/Testing/Data/test_poly_data.hdf" {
GROUP "/" {
   GROUP "VTKHDF" {
      ATTRIBUTE "Type" {
         DATATYPE  H5T_STRING {
            STRSIZE 8;
            STRPAD H5T_STR_NULLPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SCALAR
      }
      ATTRIBUTE "Version" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
      }
```

(continues on next page)

(continued from previous page)

```
      GROUP "CellData" {
         DATASET "Materials" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 816 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Lines" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 0 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
      }
      DATASET "NumberOfPoints" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
      }
      GROUP "PointData" {
         DATASET "Normals" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 412, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
         DATASET "Warping" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 412, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
      }
      DATASET "Points" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 412, 3 ) / ( H5S_UNLIMITED, 3 ) }
      }
      GROUP "Polygons" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2448 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
```

(continues on next page)

(continued from previous page)

```
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 818 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Strips" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 0 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Vertices" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 0 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( H5S_UNLIMITED ) }
         }
      }
   }
}
}
```

Overlapping AMR

The Overlapping AMR data file is an AMR Guaussian Pulse source with two levels (0 and 1), describing one Point Data, several Cell Data and a Field Data. Actual `Data` are not displayed for readability.

```
HDF5 "ExternalData/Testing/Data/amr_gaussian_pulse.hdf" {
GROUP "/" {
   GROUP "VTKHDF" {
      ATTRIBUTE "Origin" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
         DATA {
         (0): -2, -2, 0
         }
      }
      ATTRIBUTE "Type" {
         DATATYPE  H5T_STRING {
            STRSIZE 14;
            STRPAD H5T_STR_NULLPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SCALAR
         DATA {
         (0): "OverlappingAMR"
         }
      }
      ATTRIBUTE "Version" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         DATA {
         (0): 1, 0
         }
      }
      GROUP "Level0" {
         ATTRIBUTE "Spacing" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
            DATA {
            (0): 0.5, 0.5, 0.5
            }
         }
         DATASET "AMRBox" {
            DATATYPE  H5T_STD_I32LE
            DATASPACE  SIMPLE { ( 1, 6 ) / ( 1, 6 ) }
            DATA {
            (0,0): 0, 4, 0, 4, 0, 4
            }
         }
         GROUP "CellData" {
            DATASET "Centroid" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 125, 3 ) / ( 125, 3 ) }
```

(continues on next page)

(continued from previous page)

```
            }
            DATASET "Gaussian-Pulse" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 125 ) / ( 125 ) }
            }
            DATASET "vtkGhostType" {
               DATATYPE  H5T_STD_U8LE
               DATASPACE  SIMPLE { ( 125 ) / ( 125 ) }
            }
         }
         GROUP "FieldData" {
            DATASET "KE" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 44 ) / ( 44 ) }
            }
         }
         GROUP "PointData" {
            DATASET "Coord Result" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 216 ) / ( 216 ) }
            }
         }
      }
      GROUP "Level1" {
         ATTRIBUTE "Spacing" {
            DATATYPE  H5T_IEEE_F64LE
            DATASPACE  SIMPLE { ( 3 ) / ( 3 ) }
            DATA {
            (0): 0.25, 0.25, 0.25
            }
         }
         DATASET "AMRBox" {
            DATATYPE  H5T_STD_I32LE
            DATASPACE  SIMPLE { ( 2, 6 ) / ( 2, 6 ) }
            DATA {
            (0,0): 0, 3, 0, 5, 0, 9,
            (1,0): 6, 9, 4, 9, 0, 9
            }
         }
         GROUP "CellData" {
            DATASET "Centroid" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 480, 3 ) / ( 480, 3 ) }
            }
            DATASET "Gaussian-Pulse" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 480 ) / ( 480 ) }
            }
            DATASET "vtkGhostType" {
               DATATYPE  H5T_STD_U8LE
               DATASPACE  SIMPLE { ( 480 ) / ( 480 ) }
            }
```

(continues on next page)

(continued from previous page)

```
         }
         GROUP "FieldData" {
            DATASET "KE" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 88 ) / ( 88 ) }
            }
         }
         GROUP "PointData" {
            DATASET "Coord Result" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 770 ) / ( 770 ) }
            }
         }
      }
   }
}
}
```

PartitionedDataSetCollection

This partitioned dataset collection has 2 blocks, one unstructured grid (Block1) and one polydata (Block0). Its assembly has 3 elements and no nesting, referencing one of the 2 blocks using symbolic links

```
HDF5 "composite.hdf" {
GROUP "/" {
   GROUP "VTKHDF" {
      ATTRIBUTE "Type" {
         DATATYPE  H5T_STRING {
            STRSIZE 28;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SCALAR
      }
      ATTRIBUTE "Version" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
      }
      GROUP "Assembly" {
         GROUP "blockName0" {
            SOFTLINK "Block0" {
               LINKTARGET "/VTKHDF/Block0"
            }
         }
         GROUP "blockName2" {
            SOFTLINK "Block1" {
               LINKTARGET "/VTKHDF/Block1"
            }
         }
         GROUP "groupName0" {
```

(continues on next page)

(continued from previous page)

```
            GROUP "blockName1" {
               SOFTLINK "Block1" {
                  LINKTARGET "/VTKHDF/Block1"
               }
            }
         }
      }
      GROUP "Block0" {
         ATTRIBUTE "Index" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SCALAR
         }
         ATTRIBUTE "Type" {
            DATATYPE  H5T_STRING {
               STRSIZE 8;
               STRPAD H5T_STR_NULLTERM;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SCALAR
         }
         ATTRIBUTE "Version" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         }
         GROUP "CellData" {
            DATASET "Materials" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 96 ) / ( 96 ) }
            }
         }
         GROUP "Lines" {
            DATASET "Connectivity" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 0 ) / ( 0 ) }
            }
            DATASET "NumberOfCells" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "NumberOfConnectivityIds" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "Offsets" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
         }
         DATASET "NumberOfPoints" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
```

(continues on next page)

(continued from previous page)

```
         }
         GROUP "PointData" {
            DATASET "Normals" {
               DATATYPE  H5T_IEEE_F32LE
               DATASPACE  SIMPLE { ( 50, 3 ) / ( 50, 3 ) }
            }
            DATASET "Warping" {
               DATATYPE  H5T_IEEE_F32LE
               DATASPACE  SIMPLE { ( 50, 3 ) / ( 50, 3 ) }
            }
         }
         DATASET "Points" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 50, 3 ) / ( 50, 3 ) }
         }
         GROUP "Polygons" {
            DATASET "Connectivity" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 288 ) / ( 288 ) }
            }
            DATASET "NumberOfCells" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "NumberOfConnectivityIds" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "Offsets" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 97 ) / ( 97 ) }
            }
         }
         GROUP "Strips" {
            DATASET "Connectivity" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 0 ) / ( 0 ) }
            }
            DATASET "NumberOfCells" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "NumberOfConnectivityIds" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "Offsets" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
         }
         GROUP "Vertices" {
```

(continues on next page)

(continued from previous page)

```
            DATASET "Connectivity" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 0 ) / ( 0 ) }
            }
            DATASET "NumberOfCells" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "NumberOfConnectivityIds" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
            DATASET "Offsets" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
            }
         }
      }
      GROUP "Block1" {
         ATTRIBUTE "Index" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SCALAR
         }
         ATTRIBUTE "Type" {
            DATATYPE  H5T_STRING {
               STRSIZE 16;
               STRPAD H5T_STR_NULLTERM;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
            DATASPACE  SCALAR
         }
         ATTRIBUTE "Version" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         }
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 8 ) / ( 8 ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
         DATASET "NumberOfPoints" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
```

(continues on next page)

(continued from previous page)

```
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
         }
         DATASET "Points" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 8, 3 ) / ( 8, 3 ) }
         }
         DATASET "Types" {
            DATATYPE  H5T_STD_U8LE
            DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         }
      }
   }
}
}
```

Temporal Poly Data

The poly data is the `test_transient_poly_data.hdf` from the VTK testing data:

```
HDF5 "ExternalData/Testing/Data/test_transient_poly_data.hdf" {
GROUP "/" {
   GROUP "VTKHDF" {
      ATTRIBUTE "Type" {
         DATATYPE  H5T_STRING {
            STRSIZE 8;
            STRPAD H5T_STR_NULLPAD;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
         DATASPACE  SCALAR
      }
      ATTRIBUTE "Version" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 2 ) / ( 2 ) }
      }
      GROUP "CellData" {
         DATASET "Materials" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 8160 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Lines" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 0 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
```

(continues on next page)

(continued from previous page)

```
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
      }
      DATASET "NumberOfPoints" {
         DATATYPE  H5T_STD_I64LE
         DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
      }
      GROUP "PointData" {
         DATASET "Normals" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 4120, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
         DATASET "Warping" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 4120, 3 ) / ( H5S_UNLIMITED, 3 ) }
         }
      }
      DATASET "Points" {
         DATATYPE  H5T_IEEE_F32LE
         DATASPACE  SIMPLE { ( 2060, 3 ) / ( H5S_UNLIMITED, 3 ) }
      }
      GROUP "Polygons" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 12240 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 4090 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Steps" {
         ATTRIBUTE "NSteps" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SCALAR
         }
         GROUP "CellDataOffsets" {
```

(continues on next page)

(continued from previous page)

```
            DATASET "Materials" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
            }
         }
         DATASET "CellOffsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10, 4 ) / ( H5S_UNLIMITED, 4 ) }
         }
         DATASET "ConnectivityIdOffsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10, 4 ) / ( H5S_UNLIMITED, 4 ) }
         }
         DATASET "NumberOfParts" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "PartOffsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         GROUP "PointDataOffsets" {
            DATASET "Normals" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
            }
            DATASET "Warping" {
               DATATYPE  H5T_STD_I64LE
               DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
            }
         }
         DATASET "PointOffsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Values" {
            DATATYPE  H5T_IEEE_F32LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Strips" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 0 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
```

(continues on next page)

(continued from previous page)

```
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
      }
      GROUP "Vertices" {
         DATASET "Connectivity" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 0 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfCells" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "NumberOfConnectivityIds" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
         DATASET "Offsets" {
            DATATYPE  H5T_STD_I64LE
            DATASPACE  SIMPLE { ( 10 ) / ( H5S_UNLIMITED ) }
         }
      }
   }
}
}
```

10.2 Parallel Processing with VTK's SMP Framework

August 2022

10.2.1 Contributors

Berk Geveci wrote the initial version of this document in 2013. The design and implementation of vtkSMPTools was strongly influenced by the KAAPI thread scheduling system and an associated Inria research report: *VtkSMP: Task-based Parallel Operators for Accelerating VTK Filters*. Later contributors to this document include:

- Timothee Couble
- Charles Gueunet
- Will Schroeder
- Spiros Tsalikis

Also note that several blog posts have been written about vtkSMPTools:

- Simple, Parallel Computing with vtkSMPTools
- VTK Shared Memory Parallelism Tools, 2021 updates
- Ongoing VTK / ParaView Performance Improvements

- VTK/ParaView Filters: Performance Improvements

10.2.2 Introduction

The overarching objective of vtkSMPTools, the SMP (symmetric multiprocessing) framework, is to provide an infrastructure to simplify the development of shared memory parallel algorithms in VTK. In addition, vtkSMPTools defines a simple, abstract API that drives several threading backends such as std::thread, TBB (i.e., Intel's Threading Building Blocks template library); and OpenMP; as well as supporting a sequential backend for testing and debugging. To achieve these objectives, we have developed three simple constructs to support basic SMP functionality:

- Parallel building blocks / functions
- Thread local storage
- Atomic integers and associated operations. (Note, since C++11 this has been superseded by `std::atomic<>`. Also, `std::mutex` and `vtkAtomicMutex` are options.)

vtkSMPTools is extremely easy to use, ensuring that the major challenge of creating parallel algorithms is not one of implementation, but rather the design of good, threaded algorithms. In the next sections we describe the basic concepts used in vtkSMPTools, and then demonstrate these concepts through example code. Of course, there are hundreds of vtkSMPTools implementations found in VTK which provide an excellent source of more complex examples. In the final section of this document we provide tips on how to design and implement vtkSMPTools-based algorithms.

10.2.3 Concepts

The following are several high-level concepts that will help you understand and use vtkSMPTools.

The Age of Abundant Computing Cores

Many early computational algorithms were designed and implemented in an era of limited computing resources: typically a single CPU was available with rudimentary memory models. Such limitations typically led to a frugal approach to writing algorithms, in particular approaches that minimized CPU utilization. However modern computing architectures commonly have many cores with multiple execution threads per core, and memory models have expanded to include a hierarchy of data caches to retrieve frequently used data more quickly. Also, many developers are inclined to think in terms of *sequential* algorithmic operations, partly due to the way in which we were trained but also because managing multiple simultaneous processes can take a lot of work and programmers are often pressed for time. But with growing data sizes, increasing computational demands, and the abundance of computing threads; it's clear that parallel approaches are essential to creating responsive and impactful software tools. It's important that VTK developers conceive and implement performant parallel algorithms to ensure that the system remain vital into the future.

There are a variety of approaches to parallel computing, but two approaches - distributed computing and shared memory computing - are particularly relevant to VTK. In distributed computing, computational tasks are carried out in separate memory space and exchange information through message passing communication. In shared memory computing, information is exchanged through variables in shared memory space. Typically a flavor of MPI is used by VTK for distributed computing, plus VTK provides a variety of software constructs to support distributed computing. vtkSMPTools is used to implement shared memory computing with symmetric multiprocessing (SMP) approaches; i.e., where multiple processors are connected to a single, shared memory space. Distributed computing is more complex and scales best for extremely large data, while shared memory computing is simpler and works cell on single computers (desktop, laptop, mobile). Note that it is possible to combine distributed and shared computing in a VTK application.

Besides MPI (for distributed computing) and vtkSMPTools (shared memory parallelism, typically on CPUs), be aware that VTK leverages another parallel processing toolkit for computing accelerators (e.g., GPUs). vtk-m is a toolkit of scientific visualization algorithms for emerging processor architectures, supporting fine-grained concurrency for data analysis and visualization algorithms. Depending on the application, vtk-m may be a preferred solution for extreme scale computing. It is possible to mix all three forms of parallel computing frameworks into a single VTK application.

Fine- and Coarse-Grained Parallelism

When parallelizing an algorithm, it is important to first consider the “dimension” (i.e., the way in which data is accessed via threads) over which to parallelize it. For example, VTK’s Imaging modules parallelize many algorithms by assigning subsets of the input image (VOIs) to a thread safe function which processes them in parallel. Another example is parallelizing over blocks of a composite dataset (such as an AMR dataset). We refer to these examples as coarse-grained parallelism. On the other hand, we can choose points or cells as a dimension over which to parallelize access to a VTK dataset. Many algorithms simply loop over cells or points and are relatively trivial to parallelize this way. Here we refer to this approach as fine-grained parallelism. Note that some algorithms fall into a gray area. For example, if we parallelize streamline generation over seeds, is it fine- or coarse-grained parallelism?

Backends

The SMP framework provides a thin abstraction over a number of threading backends. Currently, we support four backends: Sequential (serial execution); C++ std::thread referred to as STDThread; TBB (based on Intel’s TBB); and OpenMP. Note that the Sequential backend is useful for debugging but is typically not used unless no other backend can be made to work on the target platform. As discussed in the following, it’s possible to build VTK with multiple backends, and switch between them at run-time.

Backends are configured via CMake during the build process. Setting the CMake variables `VTK_SMP_ENABLE_OPENMP`, `VTK_SMP_ENABLE_SEQUENTIAL`, `VTK_SMP_ENABLE_STDTHREAD`, and `VTK_SMP_ENABLE_TBB` enables the inclusion of the appropriate SMP backend(s), and `VTK_SMP_IMPLEMENTATION_TYPE` can be used to select one of Sequential, OpenMP, TBB, and STDThread (this selects the default backend when VTK runs). Once VTK is built, setting the environment variable `VTK_SMP_BACKEND_IN_USE` can be used to select from multiple backends. (Note: `vtkSMPTools::SetBackend()` can be used from within a C++ application to select the backend as well – for example `vtkSMPTools::SetBackend("TBB")` will select TBB.)

Thread Safety in VTK

Probably the most important thing in parallelizing shared-memory algorithms is to make sure that all operations that occur in a parallel region are performed in a thread-safe way (i.e., avoid race conditions). Note that there is much in the VTK core functionality that is not thread-safe. The VTK community has an ongoing effort of cleaning this up and marking APIs that are thread-safe to use. At this point, probably the best approach is to double check by looking at the implementation. Also, we highly recommend using analysis tools such as ThreadSanitizer or Valgrind (with the Helgrind tool) to look for race conditions in problematic code.

When coding parallel algorithms, be especially wary of insidious execution side effects. Such side effects typically result in simultaneous execution of code. For example, invoking `Update()` on a filter shared by multiple threads is a bad idea since simultaneous updates to that filter is likely doomed to fail. Also, some methods like `vtkPolyData::GetCellNeighbors()` internally invoke the one-time operation `BuildLinks()` in order to generate topological information. Similarly, the `BuildLocator()` method found in point and cell locators may be called as a side effect of a geoemtric query such as `vtkDataSet::FindCell()`. In such cases, prior to threaded execution, affected classes should be “primed” by explicitly invoking methods that produce side effects (e.g., call `BuildLinks()` directly on the `vtkPolyData`; or manually call `BuildLocator()` prior to using methods that require a locator).

Results Invariance

A significant challenge to writing good threaded algorithms is to insure that they produce the same output each time they execute. For example, a threaded sort operation may order *identical* set elements differently each time the sort is run depending on the order in which data is processed by different computing threads. (This is related to the C++ standard providing the `std::stable_sort` algorithm.) Even simple threaded operation such as summing a list of numbers can produce different results, since the order and partitioning of data during threading may result in round off effects. Since sequential algorithms implicitly order their operations, and threading typically does not do so (unless extensive use of locks, barriers, etc. are used), a sequential algorithm may produce different results than a threaded algorithm, and even across multiple runs threaded algorithms may produce results that vary across each run. Such behaviors are disturbing to users, and make testing difficult. In VTK, we aim to write algorithms that are results invariant.

Show Me the Code

The vtkSMPTools class defined in `VTK\Common\Core\vtkSMPTools.h` provides detailed documentation and further implementation details. To find examples of vtkSMPTools in use, simply search for VTK C++ classes that include this header file.

10.2.4 Implementation Overview

As mentioned previously, vtkSMPTools provides a few, simple programmatic building blocks; support for thread-local storage; and support for atomics. In this section we provide high-level descriptions of these building blocks. Then in the following section we provide implementation details.

Functional Building Blocks

The core, functional building blocks of vtkSMPTools are as follows. See `vtkSMPTools.h` for details.

- `For(begin, end, functor)` - a for loop over the range [begin,end) executing the functor each time.
- `Fill(begin, end, value)` - assign the given value to the elements in range [begin,end) (a drop in replacement for `std::fill()`).
- `Sort(begin,end)` and `Sort(begin,end,compare)` - sort the elements in range [begin,end) using the optional comparison function (a drop in replacement for `std::sort()`).
- `Transform()` - a drop in replacement for `std::transform()`.

Note that the ranges [begin,end) may be expressed via integral (`vtkIdType`) types for example point or cell ids, or C++ iterators.

Of special interest is the `functor` invoked in the `For()` loop. The functor is a class/struct which requires defining the `void operator()(begin,end)` method. Given a range defined by `[begin, end)` and the functor, `For()` will call the functor's `operator()`, usually in parallel, over a number of subranges of `[begin, end)`. The functor may also implement methods to initialize data associated with each thread (`void Initialize()`), and to composite the results of executing the `For()` loop into a final result (i.e., `void Reduce()`).

With these few building blocks, powerful threaded algorithms can easily be written. In many cases, the `For()` loop is all that is needed.

Thread Local Storage

Often times parallel algorithms produce intermediate results that are combined to produce a final result. For example, to sum a long list of numbers, each thread may sum just a subset of the numbers, and when completed the intermediate sums from each thread can be combined to produce a final summation. So the ability to maintain intermediate data associated with each thread is valuable. This is the purpose of thread local storage.

Thread local storage is generally referred to memory that is accessed by one thread only. In the SMP framework, vtkSMPThreadLocal and vtkSMPThreadLocalObject enable the creation of objects local to executing threads. The main difference between the two is that vtkSMPThreadLocalObject makes it easy to manage vtkObject and subclasses by allocating and deleting them appropriately. Thread local storage almost always requires definition of the `Initialize()` and `Reduce()` methods to initialize local storage, and then combine it once the `For()` loop completes.

One important performance trick with thread local storage, is that temporary variables may be defined and then used in the execution of `operator()`. For example, instantiating temporary objects such as vtkGenericCell, vtkIdList, and other C++ containers or classes can be relatively slow. Sometimes it's much faster to create and initialize them once (when the thread is created), and then "reset" them in each invocation of `operator()`.

Atomics

Another very useful tool when developing shared memory parallel algorithms is atomic integers. Atomic integers provide the ability to manipulate integer values in a way that can't be interrupted by other threads. A very common use case for atomic integers is implementing global counters. For example, in VTK, the modified time (MTime) global counter and vtkObject's reference count are implemented as atomic integers.

Prior to C++11, vtkSMPTools had an internal implementation for atomic integers. However, this implementation is now obsolete in favor of `std::atomic<>`. C++ also provides `std::mutex' and 'std::lock_guard`<>; and VTK provides a lightweight spinlock `vtkAtomicMutex` which may be faster than using mutexes.

10.2.5 Implementation Examples

In the subsections below, we describe the SMP framework in more detail and provide examples of how it can be used.

Functors and Parallel For

The `vtkSMPTools::For()` parallel for is the core computational construct of vtkSMPTools. It's use is as shown in the following example which evaluates points against a set of planes, and adjusts the planes to "bound" the points (see `vtkHull.cxx` and `VTK/Common/DataModel/Testing/Cxx/TestSMPFeatures.cxx`).

```
  vtkNew<vtkPoints> pts;
  pts->SetDataTypeToFloat();
  pts->SetNumberOfPoints(numPts);
  for ( auto i=0; i < numPts; ++i)
  {
    pts->SetPoint(i, vtkMath::Random(-1,1), vtkMath::Random(-1,1), vtkMath::Random(-1,
↪1));
  }
```

Now define the functor:

```
struct HullFunctor
{
  vtkPoints *InPts;
```

(continues on next page)

(continued from previous page)

```
  std::vector<double>& Planes;

  HullFunctor(vtkPoints *inPts, std::vector<double>& planes) : InPts(inPts),␣
↪Planes(planes) {}

  void operator()(vtkIdType ptId, vtkIdType endPtId)
  {
    vtkPoints *inPts = this->InPts;
    std::vector<double>& planes = this->Planes;
    auto numPlanes = planes.size() / 4;

    for (; ptId < endPtId; ++ptId)
    {
      double v, coord[3];
      inPts->GetPoint(ptId, coord);
      for (size_t j = 0; j < numPlanes; j++)
      {
        v = -(planes[j * 4 + 0] * coord[0] + planes[j * 4 + 1] * coord[1] +
          planes[j * 4 + 2] * coord[2]);
        // negative means further in + direction of plane
        if (v < planes[j * 4 + 3])
        {
          planes[j * 4 + 3] = v;
        }
      }
    }
 }
}; //HullFunctor
```

To use the functor and invoke `vtkSMPTools::For()`:

```
  HullFunctor hull(pts,planes);
  vtkSMPTools::For(0,numPts, hull);
```

Note that same code can be conveniently and compactly defined inline via a C++ lambda function. Lambdas are particularly useful when thread local storage and/or local variable are not required.

```
  vtkSMPTools::For(0, numPts, [&](vtkIdType ptId, vtkIdType endPtId) {
    for (; ptId < endPtId; ++ptId)
    {
      double v, coord[3];
      pts->GetPoint(ptId, coord);
      for (auto j = 0; j < numPlanes; j++)
      {
        v = -(planes[j * 4 + 0] * coord[0] + planes[j * 4 + 1] * coord[1] +
          planes[j * 4 + 2] * coord[2]);
        // negative means further in + direction of plane
        if (v < planes[j * 4 + 3])
        {
          planes[j * 4 + 3] = v;
        }
      }
```

(continues on next page)

(continued from previous page)

```
    }
  }); // end lambda
```

With alternative signatures for `For()` it is possible to provide a grain parameter. Grain is a hint to the underlying backend about the coarseness of the typical range when parallelizing a for loop. If you don't know what grain will work best for a particular problem, omit the grain specification and let the backend find a suitable grain. TBB in particular does a good job with this. Sometimes, you can eek out a little bit more performance by setting the grain just right. Too small, the task queuing overhead will be too much. Too little, load balancing will suffer.

Thread Local Storage

Thread local storage is generally referred to memory that is accessed by one thread only. In the SMP framework, vtkSMPThreadLocal and vtkSMPThreadLocalObject enable the creation objects local to executing threads. The main difference between the two is that vtkSMPThreadLocalObject makes it easy to manage vtkObject and subclasses by allocating and deleting them appropriately.

Below is an example of thread local objects in use. This example computes the bounds of a set of points represented by a vtkFloatArray. Note in particular the introduction of the `Initialize()` and `Reduce()` methods:

```
using BoundsArray = std::array<double,6>;
using TLS = vtkSMPThreadLocal<BoundsArray>;

struct BoundsFunctor
{
  vtkFloatArray* Pts;
  BoundsArray Bounds;
  TLS LocalBounds;

  BoundsFunctor(vtkFloatArray *pts) : Pts(pts) {}

  // Initialize thread local storage
  void Initialize()
  {
    // The first call to .Local() will create the array,
    // all others will return the same.
    std::array<double,6>& bds = this->LocalBounds.Local();
    bds[0] = VTK_DOUBLE_MAX;
    bds[1] = -VTK_DOUBLE_MAX;
    bds[2] = VTK_DOUBLE_MAX;
    bds[3] = -VTK_DOUBLE_MAX;
    bds[4] = VTK_DOUBLE_MAX;
    bds[5] = -VTK_DOUBLE_MAX;
  }

  // Process the range of points [begin,end)
  void operator()(vtkIdType begin, vtkIdType end)
  {
    BoundsArray& lbounds = this->LocalBounds.Local();
    float* x = this->Pts->GetPointer(3*begin);
    for (vtkIdType i=begin; i<end; i++)
    {
      lbounds[0] = (x[0] < lbounds[0] ? x[0] : lbounds[0]);
```

(continues on next page)

(continued from previous page)

```
      lbounds[1] = (x[0] > lbounds[1] ? x[0] : lbounds[1]);
      lbounds[2] = (x[1] < lbounds[2] ? x[1] : lbounds[2]);
      lbounds[3] = (x[1] > lbounds[3] ? x[1] : lbounds[3]);
      lbounds[4] = (x[2] < lbounds[4] ? x[2] : lbounds[4]);
      lbounds[5] = (x[2] > lbounds[5] ? x[2] : lbounds[5]);

      x += 3;
    }
  }

  // Composite / combine the thread local storage into a global result.
  void Reduce()
  {
    this->Bounds[0] = VTK_DOUBLE_MAX;
    this->Bounds[1] = -VTK_DOUBLE_MAX;
    this->Bounds[2] = VTK_DOUBLE_MAX;
    this->Bounds[3] = -VTK_DOUBLE_MAX;
    this->Bounds[4] = VTK_DOUBLE_MAX;
    this->Bounds[5] = -VTK_DOUBLE_MAX;

    using TLSIter = TLS::iterator;
    TLSIter end = this->LocalBounds.end();
    for (TLSIter itr = this->LocalBounds.begin(); itr != end; ++itr)
    {
       BoundsArray& lBounds = *itr;
       this->Bounds[0] = (this->Bounds[0] < lBounds[0] ? this->Bounds[0] : lBounds[0]);
       this->Bounds[1] = (this->Bounds[1] > lBounds[1] ? this->Bounds[1] : lBounds[1]);
       this->Bounds[2] = (this->Bounds[2] < lBounds[2] ? this->Bounds[2] : lBounds[2]);
       this->Bounds[3] = (this->Bounds[3] > lBounds[3] ? this->Bounds[3] : lBounds[3]);
       this->Bounds[4] = (this->Bounds[4] < lBounds[4] ? this->Bounds[4] : lBounds[4]);
       this->Bounds[5] = (this->Bounds[5] > lBounds[5] ? this->Bounds[5] : lBounds[5]);
    }
  }
}; // BoundsFunctor
```

Then to use the functor:

```
  vtkFloatArray* ptsArray = vtkFloatArray::SafeDownCast(pts->GetData());
  BoundsFunctor calcBounds(ptsArray);
  vtkSMPTools::For(0, numPts, calcBounds);
  std::array<double,6>& bds = calcBounds.Bounds;
```

A few things to note here:

- LocalBounds.Local() will return a new instance of a `std::vector<std::vector<double>>` per thread the first time it is called by that thread. All calls afterwards will return the same instance for that thread. Therefore, threads can safely access the local object over and over again without worrying about race conditions.
- The `Initialize()` method initializes the new instance of the thread local vector with invalid bound values.

So at the end of the threaded computation, the `LocalBounds` will contain a number of arrays, each that was populated by one thread during the parallel execution. These still need to be composited to produce the global bounds. This can be achieved by iterating over all thread local values and combining them in the `Reduce()` method as shown previously. Consequently the user can simply retrieve the final bounds by accessing `calcBounds.Bounds` once `vtkSMPTools::For()` completes execution. Note that, if the methods exist, `Initialize()` and `Reduce()` are invoked automatically by `vtkSMPTools::For()`.

Very important note: if you use more than one thread local storage object, don't assume that the iterators will traverse them in the same order. The iterator for one may return the value from thread i with begin() whereas the other may return the value form thread j. If you need to store and access values together, make sure to use a struct or class to group them.

Thread local objects are immensely useful. Often, visualization algorithms want to accumulate their output by appending to a data structure. For example, the contour filter iterates over cells and produces polygons that it adds to an output vtkPolyData. This is usually not a thread safe operation. One way to address this is to use locks that serialize writing to the output data structure.

However, mutexes have a major impact on the scalability of parallel operations. Another solution is to produce a different vtkPolyData for each execution of the functor. However, this can lead to hundreds if not thousands of outputs that need to be merged, which is a difficult operation to scale. The best option is to use one vtkPolyData per thread using thread local objects. Since it is guaranteed that thread local objects are accessed by one thread at a time (but possibly in many consecutive functor invocations), it is thread safe for functors to keep adding polygons to these objects. The result is that the parallel section will produce only a few vtkPolyData, usually the same as the number of threads in the pool. It is much easier to efficiently merge these vtkPolyData.

Atomic Integers

As mentioned previously, atomics should be represented by the C++ std::atomic<>. However, to provide a brief explanation of the importance of atomics we provide the following simple example.

```
int Total = 0;
std::atomic<vtkTypeInt32> TotalAtomic(0);
constexpr int Target = 1000000;
constexpr int NumThreads = 2;

VTK_THREAD_RETURN_TYPE MyFunction(void *)
{
  for (int i=0; i<Target/NumThreads; i++)
  {
    ++Total;
    ++TotalAtomic;
  }
  return VTK_THREAD_RETURN_VALUE;
}

// Now exercise atomics
vtkNew<vtkMultiThreader> mt;
mt->SetSingleMethod(MyFunction, NULL);
mt->SetNumberOfThreads(NumThreads);
mt->SingleMethodExecute();
std::cout << Total << " " << TotalAtomic.load() << endl;
```

When this program is executed, most of the time `Total` will be different (smaller) than `Target` whereas `TotalAtomic` will be exactly the same as `Target`. For example, a test run on a Mac prints: `999982 1000000`. This is because when the integer is not atomic, both threads can read the same value of `Total`, increment and write out the same value, which leads to losing one increment operation. Whereas, when ++ happens atomically, it is guaranteed that it will read, increment and write out `Total` all in one uninterruptible operation. When atomic operations are supported at hardware level, they are very fast.

10.2.6 Tips

In this section, we provide some tips that we hope will be useful to those that want to develop shared memory parallel algorithms.

Think about Thread Safety

First things first, it is essential to keep thread safety in mind. If the parallel section does not produce correct results consistently, there is not a lot of point in the performance improvement it produces. To create thread-safe algorithms, consider using common parallel design patterns. Also verify that the API you are using is thread safe under your particular application. While VTK continues to add additional thread-safe capabilities, there are still many booby traps to avoid.

Analysis Tools Are Your Friend

The LLVM/Clang-based ThreadSanitizer is widely used to detect data races. Valgrind's Helgrind is also a wonderful tool. Use these tools often. We developed the original backends mainly using Helgrind. Note that backends like TBB can produce many false positives; you may want to try different backends to reduce these. There are commercial tools with similar functionality, e.g., Intel's Parallel Studio has static and dynamic checking.

Debugging Tricks

Beyond using the analysis tools mentioned previously (e.g., ThreadSanitizer), there are some simple tricks that can be used to resolve programming issues relatively quickly. Firstly, switch between different backends. For example, if a program runs correctly when the backend is set to Sequential, but incorrectly when the backend is other than Sequential, it's likely that there is a race condition. Such broken code, when run repeatedly, while not always failing at the same point due to the variability of thread execution, will often fail at or near the same function, providing clues as to the location of the race. Also, empirically the STDThread backend seems to be most sensitive to race conditions. So make sure to test with more than one backend especially STDThread.

Avoid Locks

Mutexes are expensive. Avoid them as much as possible. Mutexes are usually implemented as a table of locks by the kernel. They take a lot of CPU cycles to acquire. Specially, if multiple threads want to acquire them in the same parallel section. Use atomic integers if necessary. Try your best to design your algorithm without modifying the same data concurrently.

Use Atomics Sparingly

Atomics are very useful and much more efficient that mutexes. However, overusing them may lead to performance issues. Try to design your algorithm in a way that you avoid locks and atomics. This also applies to using VTK classes that manipulate atomic integers such as MTime and reference count. Try to minimize operations that cause MTime or shared reference counts to change in parallel sections.

Grain Can Be Important

In some situation, setting the right value for grain may be important. TBB does a decent job with this but there are situations where it can't do the optimal thing. There are a number of documents on setting the grain size with TBB on the Web. If you are interested in tuning your code further, we recommend taking a look at some of them.

Minimize Data Movement

This is true for serial parts of your code too but it is specially important when there are bunch of threads all accessing main memory. This can really push the limits of the memory bus. Code that is not very intensive computationally compared to how much memory it consumes is unlikely to scale well. Good cache use helps of course but may not be always sufficient. Try to group work together in tighter loops.

Choose Computation over Memory

As mentioned earlier in this document, typically computation is much cheaper than data movement. As a result, it's a good idea to create compact data structures with minimal representational fat. Such data structures may require computation to extract important information: for example, a data structure that contains a vector 3-tuple need not represent the vector magnitude since this can be quickly computed. Depending on the number of times vector magnitude is needed, the cost of computing it is usually less than the cost of placing vector magnitude into memory. Of course, effects like this are a function of scale / data size and must be considered when designing applications.

Multi-Pass Implementations

Parallel algorithms often require significant bookkeeping to properly partition and transform input data to output data. Trivial algorithms, such as mapping an input vector array of 3-tuples to an output scalar array of vector magnitudes, are easy to partition and map: for each vector tuple, a single scalar is produced; and if there are N tuples, there are N scalars. However, more complex algorithms such as building cell links (creating lists of cells connected to a point) or smoothing stencils (identifying points connected to each other via a cell edge) require an initial pass to determine the size of output arrays (and then to allocate the output), followed by another pass to actually populate the output arrays. While at first counterintuitive, it turns out that allocating a small number of large memory blocks is much, much faster than many dynamic allocations of small amounts of memory. This is one reason that a common implementation pattern for parallel algorithms is to use multiple data processing passes consisting of simple computing operations. Such an approach is quite different than many serial algorithms that often perform multiple, complex algorithmic steps for each input data item to be processed.

A variation of this approach is to use thread local storage to perform computation on a local range of input, store the result in thread local, and then reduce/composite the local storage into the global output. While this is problematic for many reasons (especially since data movement is needed to composite the output data), it still can be used to effectively partition and transform input data to the output, especially if the thread local storage is relatively small in size.

Whatever approach is used, parallel algorithms are often implemented using multiple passes. When designing parallel algorithms, it is important to think in terms of passes, and implement algorithms accordingly.

Use Parallel Design Patterns

There are many parallel operations that are used repeatedly. Of course `for` loops and `fill()` are two obvious operations, but the `sort()` operation is more widely used than might be expected. Another is the prefix sum (or inclusive scan, or simply scan) typically used to build indices into data arrays. Become familiar with these and other parallel operations and the task of designing and implementing algorithms will be much easier.

10.2.7 Parallel Is Not Always Faster

Threading introduces overhead into computation. As a result, threaded computation is not always faster than an equivalent serial operation. For example, `for` loops across a small number of data items can easily slow down computation due to **thread creation overhead**. A simple addition on each entry of an array may become a bottleneck if done using a too fine grain, due to **false sharing** (threads continuously invalidating other thread's cache). Even complex operations such as prefix sums across large amounts of data may be slower than serial implementations because of **synchronization issues**. For this reason, use threading sparingly to address data or computation of large scale. In VTK it is not uncommon to see code that switches between serial and parallel implementations based on input data size. For that reason, vtkSMPTools has an empirically determined THRESHOLD value that can be used by a developer to switch between serial and parallel implementations.

Different backends may have significantly performance characteristics as well. TBB for example uses a thread pool combined with task stealing to address load balancing challenges. Empirically at the time of writing, in some situations TBB can significantly outperform the STDThread backend especially in situations where task loads are highly variable. Of course this may change as std::thread implementations mature and evolve.

10.3 vtkArrayDispatch and Related Tools

10.3.1 Background

VTK datasets store most of their important information in subclasses of `vtkDataArray`. Vertex locations (`vtkPoints::Data`), cell topology (`vtkCellArray::Ia`), and numeric point, cell, and generic attributes (`vtkFieldData::Data`) are the dataset features accessed most frequently by VTK algorithms, and these all rely on the `vtkDataArray` API.

10.3.2 Terminology

This page uses the following terms:

A **ValueType** is the element type of an array. For instance, `vtkFloatArray` has a ValueType of `float`.

An **ArrayType** is a subclass of `vtkDataArray`. It specifies not only a ValueType, but an array implementation as well. This becomes important as `vtkDataArray` subclasses will begin to stray from the typical "array-of-structs" ordering that has been exclusively used in the past.

A **dispatch** is a runtime-resolution of a `vtkDataArray`'s ArrayType, and is used to call a section of executable code that has been tailored for that ArrayType. Dispatching has compile-time and run-time components. At compile-time, the possible ArrayTypes to be used are determined and a worker code template is generated for each type. At run-time, the type of a specific array is determined and the proper worker instantiation is called.

Template explosion refers to a sharp increase in the size of a compiled binary that results from instantiating a template function or class on many different types.

vtkDataArray

The data array type hierarchy in VTK has a unique feature when compared to typical C++ containers: a non-templated base class. All arrays containing numeric data inherit `vtkDataArray`, a common interface that sports a very useful API. Without knowing the underlying ValueType stored in data array, an algorithm or user may still work with any `vtkDataArray` in meaningful ways: The array can be resized, reshaped, read, and rewritten easily using a generic API that substitutes double-precision floating point numbers for the array's actual ValueType. For instance, we can write a simple function that computes the magnitudes for a set of vectors in one array and store the results in another using nothing but the typeless `vtkDataArray` API:

```cpp
// 3 component magnitude calculation using the vtkDataArray API.
// Inefficient, but easy to write:
void calcMagnitude(vtkDataArray *vectors, vtkDataArray *magnitude)
{
  vtkIdType numVectors = vectors->GetNumberOfTuples();
  for (vtkIdType tupleIdx = 0; tupleIdx < numVectors; ++tupleIdx)
    {
    // What data types are magnitude and vectors using?
    // We don't care! These methods all use double.
    magnitude->SetComponent(tupleIdx, 0,
      std::sqrt(vectors->GetComponent(tupleIdx, 0) *
                vectors->GetComponent(tupleIdx, 0) +
                vectors->GetComponent(tupleIdx, 1) *
                vectors->GetComponent(tupleIdx, 1) +
                vectors->GetComponent(tupleIdx, 2) *
                vectors->GetComponent(tupleIdx, 2));
    }
}
```

The Costs of Flexibility

However, this flexibility comes at a cost. Passing data through a generic API has a number of issues:

Accuracy

Not all ValueTypes are fully expressible as a `double`. The truncation of integers with > 52 bits of precision can be a particularly nasty issue.

Performance

Virtual overhead: The only way to implement such a system is to route the `vtkDataArray` calls through a run-time resolution of ValueTypes. This is implemented through the virtual override mechanism of C++, which adds a small overhead to each API call.

Missed optimization: The virtual indirection described above also prevents the compiler from being able to make assumptions about the layout of the data in-memory. This information could be used to perform advanced optimizations, such as vectorization.

So what can one do if they want fast, optimized, type-safe access to the data stored in a `vtkDataArray`? What options are available?

The Old Solution: vtkTemplateMacro

The `vtkTemplateMacro` is described in this section. While it is no longer considered a best practice to use this construct in new code, it is still usable and likely to be encountered when reading the VTK source code. Newer code should use the `vtkArrayDispatch` mechanism, which is detailed later. The discussion of `vtkTemplateMacro` will help illustrate some of the practical issues with array dispatching.

With a few minor exceptions that we won't consider here, prior to VTK 7.1 it was safe to assume that all numeric `vtkDataArray` objects were also subclasses of `vtkDataArrayTemplate`. This template class provided the implementation of all documented numeric data arrays such as `vtkDoubleArray`, `vtkIdTypeArray`, etc, and stores the tuples in memory as a contiguous array-of-structs (AOS). For example, if we had an array that stored 3-component tuples as floating point numbers, we could define a tuple as:

```
struct Tuple { float x; float y; float z; };
```

An array-of-structs, or AOS, memory buffer containing this data could be described as:

```
Tuple ArrayOfStructsBuffer[NumTuples];
```

As a result, `ArrayOfStructsBuffer` will have the following memory layout:

```
{ x1, y1, z1, x2, y2, z2, x3, y3, z3, ...}
```

That is, the components of each tuple are stored in adjacent memory locations, one tuple after another. While this is not exactly how `vtkDataArrayTemplate` implemented its memory buffers, it accurately describes the resulting memory layout.

`vtkDataArray` also defines a `GetDataType` method, which returns an enumerated value describing a type. We can used to discover the ValueType stored in the array.

Combine the AOS memory convention and `GetDataType()` with a horrific little method on the data arrays named `GetVoidPointer()`, and a path to efficient, type-safe access was available. `GetVoidPointer()` does what it says on the tin: it returns the memory address for the array data's base location as a `void*`. While this breaks encapsulation and sets off warning bells for the more pedantic among us, the following technique was safe and efficient when used correctly:

```
// 3-component magnitude calculation using GetVoidPointer.
// Efficient and fast, but assumes AOS memory layout
template <typename ValueType>
void calcMagnitudeWorker(ValueType *vectors, ValueType *magnitude,
                         vtkIdType numVectors)
{
  for (vtkIdType tupleIdx = 0; tupleIdx < numVectors; ++tupleIdx)
    {
    // We now have access to the raw memory buffers, and assuming
    // AOS memory layout, we know how to access them.
    magnitude[tupleIdx] =
      std::sqrt(vectors[3 * tupleIdx + 0] *
                vectors[3 * tupleIdx + 0] +
                vectors[3 * tupleIdx + 1] *
                vectors[3 * tupleIdx + 1] +
                vectors[3 * tupleIdx + 2] *
                vectors[3 * tupleIdx + 2]);
    }
}
```

(continues on next page)

(continued from previous page)

```
void calcMagnitude(vtkDataArray *vectors, vtkDataArray *magnitude)
{
  assert("Arrays must have same datatype!" &&
         vtkDataTypesCompare(vectors->GetDataType(),
                             magnitude->GetDataType()));
  switch (vectors->GetDataType())
    {
    vtkTemplateMacro(calcMagnitudeWorker<VTK_TT*>(
      static_cast<VTK_TT*>(vectors->GetVoidPointer(0)),
      static_cast<VTK_TT*>(magnitude->GetVoidPointer(0)),
      vectors->GetNumberOfTuples());
    }
}
```

The `vtkTemplateMacro`, as you may have guessed, expands into a series of case statements that determine an array's ValueType from the `int GetDataType()` return value. The ValueType is then `typedef`'d to `VTK_TT`, and the macro's argument is called for each numeric type returned from `GetDataType`. In this case, the call to `calcMagnitudeWorker` is made by the macro, with `VTK_TT typedef`'d to the array's ValueType.

This is the typical usage pattern for `vtkTemplateMacro`. The `calcMagnitude` function calls a templated worker implementation that uses efficient, raw memory access to a typesafe memory buffer so that the worker's code can be as efficient as possible. But this assumes AOS memory ordering, and as we'll mention, this assumption may no longer be valid as VTK moves further into the field of in-situ analysis.

But first, you may have noticed that the above example using `vtkTemplateMacro` has introduced a step backwards in terms of functionality. In the `vtkDataArray` implementation, we didn't care if both arrays were the same ValueType, but now we have to ensure this, since we cast both arrays' `void` pointers to `VTK_TT*`. What if vectors is an array of integers, but we want to calculate floating point magnitudes?

vtkTemplateMacro with Multiple Arrays

The best solution prior to VTK 7.1 was to use two worker functions. The first is templated on vector's ValueType, and the second is templated on both array ValueTypes:

```
// 3-component magnitude calculation using GetVoidPointer and a
// double-dispatch to resolve ValueTypes of both arrays.
// Efficient and fast, but assumes AOS memory layout, lots of boilerplate
// code, and the sensitivity to template explosion issues increases.
template <typename VectorType, typename MagnitudeType>
void calcMagnitudeWorker2(VectorType *vectors, MagnitudeType *magnitude,
                          vtkIdType numVectors)
{
  for (vtkIdType tupleIdx = 0; tupleIdx < numVectors; ++tupleIdx)
    {
    // We now have access to the raw memory buffers, and assuming
    // AOS memory layout, we know how to access them.
    magnitude[tupleIdx] =
      std::sqrt(vectors[3 * tupleIdx + 0] *
                vectors[3 * tupleIdx + 0] +
                vectors[3 * tupleIdx + 1] *
                vectors[3 * tupleIdx + 1] +
```

(continues on next page)

(continued from previous page)

```
                vectors[3 * tupleIdx + 2] *
                vectors[3 * tupleIdx + 2]);
    }
}

// Vector ValueType is known (VectorType), now use vtkTemplateMacro on
// magnitude:
template <typename VectorType>
void calcMagnitudeWorker1(VectorType *vectors, vtkDataArray *magnitude,
                          vtkIdType numVectors)
{
  switch (magnitude->GetDataType())
    {
    vtkTemplateMacro(calcMagnitudeWorker2(vectors,
      static_cast<VTK_TT*>(magnitude->GetVoidPointer(0)), numVectors);
    }
}

void calcMagnitude(vtkDataArray *vectors, vtkDataArray *magnitude)
{
  // Dispatch vectors first:
  switch (vectors->GetDataType())
    {
    vtkTemplateMacro(calcMagnitudeWorker1<VTK_TT*>(
      static_cast<VTK_TT*>(vectors->GetVoidPointer(0)),
      magnitude, vectors->GetNumberOfTuples());
    }
}
```

This works well, but it's a bit ugly and has the same issue as before regarding memory layout. Double dispatches using this method will also see more problems regarding binary size. The number of template instantiations that the compiler needs to generate is determined by `I = T^D`, where `I` is the number of template instantiations, `T` is the number of types considered, and `D` is the number of dispatches. As of VTK 7.1, `vtkTemplateMacro` considers 14 data types, so this double-dispatch will produce 14 instantiations of `calcMagnitudeWorker1` and 196 instantiations of `calcMagnitudeWorker2`. If we tried to resolve 3 `vtkDataArrays` into raw C arrays, 2744 instantiations of the final worker function would be generated. As more arrays are considered, the need for some form of restricted dispatch becomes very important to keep this template explosion in check.

Data Array Changes in VTK 7.1

Starting with VTK 7.1, the Array-Of-Structs (AOS) memory layout is no longer the only `vtkDataArray` implementation provided by the library. The Struct-Of-Arrays (SOA) memory layout is now available through the `vtkSOADataArrayTemplate` class. The SOA layout assumes that the components of an array are stored separately, as in:

```
struct StructOfArraysBuffer
{
  float *x; // Pointer to array containing x components
  float *y; // Same for y
  float *z; // Same for z
};
```

The new SOA arrays were added to improve interoperability between VTK and simulation packages for live visualization of in-situ results. Many simulations use the SOA layout for their data, and natively supporting these arrays in VTK will allow analysis of live data without the need to explicitly copy it into a VTK data structure.

As a result of this change, a new mechanism is needed to efficiently access array data. `vtkTemplateMacro` and `GetVoidPointer` are no longer an acceptable solution – implementing `GetVoidPointer` for SOA arrays requires creating a deep copy of the data into a new AOS buffer, a waste of both processor time and memory.

So we need a replacement for `vtkTemplateMacro` that can abstract away things like storage details while providing performance that is on-par with raw memory buffer operations. And while we're at it, let's look at removing the tedium of multi-array dispatch and reducing the problem of 'template explosion'. The remainder of this page details such a system.

10.3.3 Best Practices for vtkDataArray Post-7.1

We'll describe a new set of tools that make managing template instantiations for efficient array access both easy and extensible. As an overview, the following new features will be discussed:

- `vtkGenericDataArray`: The new templated base interface for all numeric `vtkDataArray` subclasses.
- `vtkArrayDispatch`: Collection of code generation tools that allow concise and precise specification of restrictable dispatch for up to 3 arrays simultaneously.
- `vtkArrayDownCast`: Access to specialized downcast implementations from code templates.
- `vtkDataArrayAccessor`: Provides `Get` and `Set` methods for accessing/modifying array data as efficiently as possible. Allows a single worker implementation to work efficiently with `vtkGenericDataArray` subclasses, or fallback to use the `vtkDataArray` API if needed.
- `VTK_ASSUME`: New abstraction for the compiler `__assume` directive to provide optimization hints.

These will be discussed more fully, but as a preview, here's our familiar `calcMagnitude` example implemented using these new tools:

```
// Modern implementation of calcMagnitude using new concepts in VTK 7.1:
// A worker functor. The calculation is implemented in the function template
// for operator().
struct CalcMagnitudeWorker
{
  // The worker accepts VTK array objects now, not raw memory buffers.
  template <typename VectorArray, typename MagnitudeArray>
  void operator()(VectorArray *vectors, MagnitudeArray *magnitude)
  {
    // This allows the compiler to optimize for the AOS array stride.
    VTK_ASSUME(vectors->GetNumberOfComponents() == 3);
    VTK_ASSUME(magnitude->GetNumberOfComponents() == 1);

    // These allow this single worker function to be used with both
    // the vtkDataArray 'double' API and the more efficient
    // vtkGenericDataArray APIs, depending on the template parameters:
    vtkDataArrayAccessor<VectorArray> v(vectors);
    vtkDataArrayAccessor<MagnitudeArray> m(magnitude);

    vtkIdType numVectors = vectors->GetNumberOfTuples();
    for (vtkIdType tupleIdx = 0; tupleIdx < numVectors; ++tupleIdx)
      {
```

(continues on next page)

(continued from previous page)

```
      // Set and Get compile to inlined optimizable raw memory accesses for
      // vtkGenericDataArray subclasses.
      m.Set(tupleIdx, 0, std::sqrt(v.Get(tupleIdx, 0) * v.Get(tupleIdx, 0) +
                                   v.Get(tupleIdx, 1) * v.Get(tupleIdx, 1) +
                                   v.Get(tupleIdx, 2) * v.Get(tupleIdx, 2)));
      }
  }
};

void calcMagnitude(vtkDataArray *vectors, vtkDataArray *magnitude)
{
  // Create our worker functor:
  CalcMagnitudeWorker worker;

  // Define our dispatcher. We'll let vectors have any ValueType, but only
  // consider float/double arrays for magnitudes. These combinations will
  // use a 'fast-path' implementation generated by the dispatcher:
  typedef vtkArrayDispatch::Dispatch2ByValueType
    <
      vtkArrayDispatch::AllTypes, // ValueTypes allowed by first array
      vtkArrayDispatch::Reals // ValueTypes allowed by second array
    > Dispatcher;

  // Execute the dispatcher:
  if (!Dispatcher::Execute(vectors, magnitude, worker))
    {
    // If Execute() fails, it means the dispatch failed due to an
    // unsupported array type. In this case, it's likely that the magnitude
    // array is using an integral type. This is an uncommon case, so we won't
    // generate a fast path for these, but instead call an instantiation of
    // CalcMagnitudeWorker::operator()<vtkDataArray, vtkDataArray>.
    // Through the use of vtkDataArrayAccessor, this falls back to using the
    // vtkDataArray double API:
    worker(vectors, magnitude);
    }
}
```

10.3.4 vtkGenericDataArray

The `vtkGenericDataArray` class template drives the new `vtkDataArray` class hierarchy. The ValueType is introduced here, both as a template parameter and a class-scope `typedef`. This allows a typed API to be written that doesn't require conversion to/from a common type (as `vtkDataArray` does with double). It does not implement any storage details, however. Instead, it uses the CRTP idiom to forward key method calls to a derived class without using a virtual function call. By eliminating this indirection, `vtkGenericDataArray` defines an interface that can be used to implement highly efficient code, because the compiler is able to see past the method calls and optimize the underlying memory accesses instead.

There are two main subclasses of `vtkGenericDataArray`: `vtkAOSDataArrayTemplate` and `vtkSOADataArrayTemplate`. These implement array-of-structs and struct-of-arrays storage, respectively.

10.3.5 vtkTypeList

Type lists are a metaprogramming construct used to generate a list of C++ types. They are used in VTK to implement restricted array dispatching. As we'll see, `vtkArrayDispatch` offers ways to reduce the number of generated template instantiations by enforcing constraints on the arrays used to dispatch. For instance, if one wanted to only generate templated worker implementations for `vtkFloatArray` and `vtkIntArray`, a typelist is used to specify this:

```
// Create a typelist of 2 types, vtkFloatArray and vtkIntArray:
typedef vtkTypeList::Create<vtkFloatArray, vtkIntArray> MyArrays;

Worker someWorker = ...;
vtkDataArray *someArray = ...;

// Use vtkArrayDispatch to generate code paths for these arrays:
vtkArrayDispatch::DispatchByArray<MyArrays>(someArray, someWorker);
```

There's not much to know about type lists as a user, other than how to create them. As seen above, there is a set of macros named `vtkTypeList::Create<...>`, where X is the number of types in the created list, and the arguments are the types to place in the list. In the example above, the new type list is typically bound to a friendlier name using a local `typedef`, which is a common practice.

The `vtkTypeList.h` header defines some additional type list operations that may be useful, such as deleting and appending types, looking up indices, etc. `vtkArrayDispatch::FilterArraysByValueType` may come in handy, too. But for working with array dispatches, most users will only need to create new ones, or use one of the following predefined `vtkTypeLists`:

- `vtkArrayDispatch::Reals`: All floating point ValueTypes.
- `vtkArrayDispatch::Integrals`: All integral ValueTypes.
- `vtkArrayDispatch::AllTypes`: Union of Reals and Integrals.
- `vtkArrayDispatch::Arrays`: Default list of ArrayTypes to use in dispatches.

The last one is special – `vtkArrayDispatch::Arrays` is a typelist of ArrayTypes set application-wide when VTK is built. This `vtkTypeList` of `vtkDataArray` subclasses is used for unrestricted dispatches, and is the list that gets filtered when restricting a dispatch to specific ValueTypes.

Refining this list allows the user building VTK to have some control over the dispatch process. If SOA arrays are never going to be used, they can be removed from dispatch calls, reducing compile times and binary size. On the other hand, a user applying in-situ techniques may want them available, because they'll be used to import views of intermediate results.

By default, `vtkArrayDispatch::Arrays` contains all AOS arrays. The `CMake` option `VTK_DISPATCH_SOA_ARRAYS` will enable SOA array dispatch as well. More advanced possibilities exist and are described in `VTK/Common/Core/vtkCreateArrayDispatchArrayList.cmake`.

10.3.6 vtkArrayDownCast

In VTK, all subclasses of `vtkObject` (including the data arrays) support a downcast method called `SafeDownCast`. It is used similarly to the C++ `dynamic_cast` – given an object, try to cast it to a more derived type or return `NULL` if the object is not the requested type. Say we have a `vtkDataArray` and want to test if it is actually a `vtkFloatArray`. We can do this:

```
void DoSomeAction(vtkDataArray *dataArray)
{
  vtkFloatArray *floatArray = vtkFloatArray::SafeDownCast(dataArray);
```

(continues on next page)

(continued from previous page)

```
  if (floatArray)
    {
    // ... (do work with float array)
    }
}
```

This works, but it can pose a serious problem if `DoSomeAction` is called repeatedly. `SafeDownCast` works by performing a series of virtual calls and string comparisons to determine if an object falls into a particular class hierarchy. These string comparisons add up and can actually dominate computational resources if an algorithm implementation calls `SafeDownCast` in a tight loop.

In such situations, it's ideal to restructure the algorithm so that the downcast only happens once and the same result is used repeatedly, but sometimes this is not possible. To lessen the cost of downcasting arrays, a `FastDownCast` method exists for common subclasses of `vtkAbstractArray`. This replaces the string comparisons with a single virtual call and a few integer comparisons and is far cheaper than the more general SafeDownCast. However, not all array implementations support the `FastDownCast` method.

This creates a headache for templated code. Take the following example:

```
template <typename ArrayType>
void DoSomeAction(vtkAbstractArray *array)
{
  ArrayType *myArray = ArrayType::SafeDownCast(array);
  if (myArray)
    {
    // ... (do work with myArray)
    }
}
```

We cannot use `FastDownCast` here since not all possible ArrayTypes support it. But we really want that performance increase for the ones that do – `SafeDownCast`s are really slow! `vtkArrayDownCast` fixes this issue:

```
template <typename ArrayType>
void DoSomeAction(vtkAbstractArray *array)
{
  ArrayType *myArray = vtkArrayDownCast<ArrayType>(array);
  if (myArray)
    {
    // ... (do work with myArray)
    }
}
```

`vtkArrayDownCast` automatically selects `FastDownCast` when it is defined for the ArrayType, and otherwise falls back to `SafeDownCast`. This is the preferred array downcast method for performance, uniformity, and reliability.

10.3.7 vtkDataArrayAccessor

Array dispatching relies on having templated worker code carry out some operation. For instance, take this `vtkArrayDispatch` code that locates the maximum value in an array:

```
// Stores the tuple/component coordinates of the maximum value:
struct FindMax
{
  vtkIdType Tuple; // Result
  int Component; // Result

  FindMax() : Tuple(-1), Component(-1) {}

  template <typename ArrayT>
  void operator()(ArrayT *array)
  {
    // The type to use for temporaries, and a temporary to store
    // the current maximum value:
    typedef typename ArrayT::ValueType ValueType;
    ValueType max = std::numeric_limits<ValueType>::min();

    // Iterate through all tuples and components, noting the location
    // of the largest element found.
    vtkIdType numTuples = array->GetNumberOfTuples();
    int numComps = array->GetNumberOfComponents();
    for (vtkIdType tupleIdx = 0; tupleIdx < numTuples; ++tupleIdx)
      {
      for (int compIdx = 0; compIdx < numComps; ++compIdx)
        {
        if (max < array->GetTypedComponent(tupleIdx, compIdx))
          {
          max = array->GetTypedComponent(tupleIdx, compIdx);
          this->Tuple = tupleIdx;
          this->Component = compIdx;
          }
        }
      }
  }
};

void someFunction(vtkDataArray *array)
{
  FindMax maxWorker;
  vtkArrayDispatch::Dispatch::Execute(array, maxWorker);
  // Do work using maxWorker.Tuple and maxWorker.Component...
}
```

There's a problem, though. Recall that only the arrays in `vtkArrayDispatch::Arrays` are tested for dispatching. What happens if the array passed into someFunction wasn't on that list?

The dispatch will fail, and `maxWorker.Tuple` and `maxWorker.Component` will be left to their initial values of -1. That's no good. What if `someFunction` is a critical path where we want to use a fast dispatched worker if possible, but still have valid results to use if dispatching fails? Well, we can fall back on the `vtkDataArray` API and do things the slow way in that case. When a dispatcher is given an unsupported array, Execute() returns false, so let's just add a backup implementation:

```
// Stores the tuple/component coordinates of the maximum value:
struct FindMax
{ /* As before... */ };

void someFunction(vtkDataArray *array)
{
  FindMax maxWorker;
  if (!vtkArrayDispatch::Dispatch::Execute(array, maxWorker))
    {
    // Reimplement FindMax::operator(), but use the vtkDataArray API's
    // "virtual double GetComponent()" instead of the more efficient
    // "ValueType GetTypedComponent()" from vtkGenericDataArray.
    }
}
```

Ok, that works. But ugh…why write the same algorithm twice? That's extra debugging, extra testing, extra maintenance burden, and just plain not fun.

Enter `vtkDataArrayAccessor`. This utility template does a very simple, yet useful, job. It provides component and tuple based `Get` and `Set` methods that will call the corresponding method on the array using either the `vtkDataArray` or `vtkGenericDataArray` API, depending on the class's template parameter. It also defines an `APIType`, which can be used to allocate temporaries, etc. This type is double for `vtkDataArray`s and `vtkGenericDataArray::ValueType` for `vtkGenericDataArray`s.

Another nice benefit is that `vtkDataArrayAccessor` has a more compact API. The only defined methods are Get and Set, and they're overloaded to work on either tuples or components (though component access is encouraged as it is much, much more efficient). Note that all non-element access operations (such as `GetNumberOfTuples`) should still be called on the array pointer using `vtkDataArray` API.

Using `vtkDataArrayAccessor`, we can write a single worker template that works for both `vtkDataArray` and `vtkGenericDataArray`, without a loss of performance in the latter case. That worker looks like this:

```
// Better, uses vtkDataArrayAccessor:
struct FindMax
{
  vtkIdType Tuple; // Result
  int Component; // Result

  FindMax() : Tuple(-1), Component(-1) {}

  template <typename ArrayT>
  void operator()(ArrayT *array)
  {
    // Create the accessor:
    vtkDataArrayAccessor<ArrayT> access(array);

    // Prepare the temporary. We'll use the accessor's APIType instead of
    // ArrayT::ValueType, since that is appropriate for the vtkDataArray
    // fallback:
    typedef typename vtkDataArrayAccessor<ArrayT>::APIType ValueType;
    ValueType max = std::numeric_limits<ValueType>::min();

    // Iterate as before, but use access.Get instead of
    // array->GetTypedComponent. GetTypedComponent is still used
```

(continues on next page)

(continued from previous page)

```
    // when ArrayT is a vtkGenericDataArray, but
    // vtkDataArray::GetComponent is now used as a fallback when ArrayT
    // is vtkDataArray.
    vtkIdType numTuples = array->GetNumberOfTuples();
    int numComps = array->GetNumberOfComponents();
    for (vtkIdType tupleIdx = 0; tupleIdx < numTuples; ++tupleIdx)
      {
      for (int compIdx = 0; compIdx < numComps; ++compIdx)
        {
        if (max < access.Get(tupleIdx, compIdx))
          {
          max = access.Get(tupleIdx, compIdx);
          this->Tuple = tupleIdx;
          this->Component = compIdx;
          }
        }
      }
  }
};
```

Now when we call `operator()` with say, `ArrayT=vtkFloatArray`, we'll get an optimized, efficient code path. But we can also call this same implementation with `ArrayT=vtkDataArray` and still get a correct result (assuming that the `vtkDataArray`'s double API represents the data well enough).

Using the `vtkDataArray` fallback path is straightforward. At the call site:

```
void someFunction(vtkDataArray *array)
{
  FindMax maxWorker;
  if (!vtkArrayDispatch::Dispatch::Execute(array, maxWorker))
    {
    maxWorker(array); // Dispatch failed, call vtkDataArray fallback
    }
  // Do work using maxWorker.Tuple and maxWorker.Component -- now we know
  // for sure that they're initialized!
}
```

Using the above pattern for calling a worker and always going through `vtkDataArrayAccessor` to `Get/Set` array elements ensures that any worker implementation can be its own fallback path.

10.3.8 VTK_ASSUME

While performance testing the new array classes, we compared the performance of a dispatched worker using the `vtkDataArrayAccessor` class to the same algorithm using raw memory buffers. We managed to achieve the same performance out of the box for most cases, using both AOS and SOA array implementations. In fact, with `--ffast-math` optimizations on GCC 4.9, the optimizer is able to remove all function calls and apply SIMD vectorized instructions in the dispatched worker, showing that the new array API is thin enough that the compiler can see the algorithm in terms of memory access.

But there was one case where performance suffered. If iterating through an AOS data array with a known number of components using `GetTypedComponent`, the raw pointer implementation initially outperformed the dispatched array. To understand why, note that the AOS implementation of `GetTypedComponent` is along the lines of:

```
ValueType vtkAOSDataArrayTemplate::GetTypedComponent(vtkIdType tuple,
                                                       int comp) const
{
  // AOSData is a ValueType* pointing at the base of the array data.
  return this->AOSData[tuple * this->NumberOfComponents + comp];
}
```

Because `NumberOfComponents` is unknown at compile time, the optimizer cannot assume anything about the stride of the components in the array. This leads to missed optimizations for vectorized read/writes and increased complexity in the instructions used to iterate through the data.

For such cases where the number of components is, in fact, known at compile time (due to a calling function performing some validation, for instance), it is possible to tell the compiler about this fact using `VTK_ASSUME`.

`VTK_ASSUME` wraps a compiler-specific `__assume` statement, which is used to pass such optimization hints. Its argument is an expression of some condition that is guaranteed to always be true. This allows more aggressive optimizations when used correctly, but be forewarned that if the condition is not met at runtime, the results are unpredictable and likely catastrophic.

But if we're writing a filter that only operates on 3D point sets, we know the number of components in the point array will always be 3. In this case we can write:

```
VTK_ASSUME(pointsArray->GetNumberOfComponents() == 3);
```

in the worker function and this instructs the compiler that the array's internal `NumberOfComponents` variable will always be 3, and thus the stride of the array is known. Of course, the caller of this worker function should ensure that this is a 3-component array and fail gracefully if it is not.

There are many scenarios where `VTK_ASSUME` can offer a serious performance boost, the case of known tuple size is a common one that's really worth remembering.

10.3.9 vtkArrayDispatch

The dispatchers implemented in the vtkArrayDispatch namespace provide array dispatching with customizable restrictions on code generation and a simple syntax that hides the messy details of type resolution and multi-array dispatch. There are several "flavors" of dispatch available that operate on up to three arrays simultaneously.

Components Of A Dispatch

Using the `vtkArrayDispatch` system requires three elements: the array(s), the worker, and the dispatcher.

The Arrays

All dispatched arrays must be subclasses of `vtkDataArray`. It is important to identify as many restrictions as possible. Must every ArrayType be considered during dispatch, or is the array's ValueType (or even the ArrayType itself) restricted? If dispatching multiple arrays at once, are they expected to have the same ValueType? These scenarios are common, and these conditions can be used to reduce the number of instantiations of the worker template.

The Worker

The worker is some generic callable. In C++98, a templated functor is a good choice. In C++14, a generic lambda is a usable option as well. For our purposes, we'll only consider the functor approach, as C++14 is a long ways off for core VTK code.

At a minimum, the worker functor should define `operator()` to make it callable. This should be a function template with a template parameter for each array it should handle. For a three array dispatch, it should look something like this:

```
struct ThreeArrayWorker
{
  template <typename Array1T, typename Array2T, typename Array3T>
  void operator()(Array1T *array1, Array2T *array2, Array3T *array3)
  {
  /* Do stuff... */
  }
};
```

At runtime, the dispatcher will call `ThreeWayWorker::operator()` with a set of `Array1T`, `Array2T`, and `Array3T` that satisfy any dispatch restrictions.

Workers can be stateful, too, as seen in the `FindMax` worker earlier where the worker simply identified the component and tuple id of the largest value in the array. The functor stored them for the caller to use in further analysis:

```
// Example of a stateful dispatch functor:
struct FindMax
{
  // Functor state, holds results that are accessible to the caller:
  vtkIdType Tuple;
  int Component;

  // Set initial values:
  FindMax() : Tuple(-1), Component(-1) {}

  // Template method to set Tuple and Component ivars:
  template <typename ArrayT>
  void operator()(ArrayT *array)
  {
    /* Do stuff... */
  }
};
```

The Dispatcher

The dispatcher is the workhorse of the system. It is responsible for applying restrictions, resolving array types, and generating the requested template instantiations. It has responsibilities both at run-time and compile-time.

During compilation, the dispatcher will identify the valid combinations of arrays that can be used according to the restrictions. This is done by starting with a typelist of arrays, either supplied as a template parameter or by defaulting to `vtkArrayDispatch::Arrays`, and filtering them by ValueType if needed. For multi-array dispatches, additional restrictions may apply, such as forcing the second and third arrays to have the same ValueType as the first. It must then generate the required code for the dispatch – that is, the templated worker implementation must be instantiated for each valid combination of arrays.

At runtime, it tests each of the dispatched arrays to see if they match one of the generated code paths. Runtime type resolution is carried out using `vtkArrayDownCast` to get the best performance available for the arrays of interest. If it finds a match, it calls the worker's `operator()` method with the properly typed arrays. If no match is found, it returns `false` without executing the worker.

Restrictions: Why They Matter

We've made several mentions of using restrictions to reduce the number of template instantiations during a dispatch operation. You may be wondering if it really matters so much. Let's consider some numbers.

VTK is configured to use 13 ValueTypes for numeric data. These are the standard numeric types `float`, `int`, `unsigned char`, etc. By default, VTK will define `vtkArrayDispatch::Arrays` to use all 13 types with `vtkAOSDataArrayTemplate` for the standard set of dispatchable arrays. If enabled during compilation, the SOA data arrays are added to this list for a total of 26 arrays.

Using these 26 arrays in a single, unrestricted dispatch will result in 26 instantiations of the worker template. A double dispatch will generate 676 workers. A triple dispatch with no restrictions creates a whopping 17,576 functions to handle the possible combinations of arrays. That's a *lot* of instructions to pack into the final binary object.

Applying some simple restrictions can reduce this immensely. Say we know that the arrays will only contain `floats` or `doubles`. This would reduce the single dispatch to 4 instantiations, the double dispatch to 16, and the triple to 64. We've just reduced the generated code size significantly. We could even apply such a restriction to just create some 'fast-paths' and let the integral types fallback to using the `vtkDataArray` API by using `vtkDataArrayAccessors`. Dispatch restriction is a powerful tool for reducing the compiled size of a binary object.

Another common restriction is that all arrays in a multi-array dispatch have the same ValueType, even if that ValueType is not known at compile time. By specifying this restriction, a double dispatch on all 26 AOS/SOA arrays will only produce 52 worker instantiations, down from 676. The triple dispatch drops to 104 instantiations from 17,576.

Always apply restrictions when they are known, especially for multi-array dispatches. The savings are worth it.

Types of Dispatchers

Now that we've discussed the components of a dispatch operation, what the dispatchers do, and the importance of restricting dispatches, let's take a look at the types of dispatchers available.

---

vtkArrayDispatch::Dispatch

This family of dispatchers take no parameters and perform an unrestricted dispatch over all arrays in `vtkArrayDispatch::Arrays`.

Variations:

- `vtkArrayDispatch::Dispatch`: Single dispatch.
- `vtkArrayDispatch::Dispatch2`: Double dispatch.
- `vtkArrayDispatch::Dispatch3`: Triple dispatch.

Arrays considered: All arrays in `vtkArrayDispatch::Arrays`.

Restrictions: None.

Usecase: Used when no useful information exists that can be used to apply restrictions.

Example Usage:

```
vtkArrayDispatch::Dispatch::Execute(array, worker);
```

vtkArrayDispatch::DispatchByArray

This family of dispatchers takes a `vtkTypeList` of explicit array types to use during dispatching. They should only be used when an array's exact type is restricted. If dispatching multiple arrays and only one has such type restrictions, use `vtkArrayDispatch::Arrays` (or a filtered version) for the unrestricted arrays.

Variations:

- `vtkArrayDispatch::DispatchByArray`: Single dispatch.
- `vtkArrayDispatch::Dispatch2ByArray`: Double dispatch.
- `vtkArrayDispatch::Dispatch3ByArray`: Triple dispatch.

Arrays considered: All arrays explicitly listed in the parameter lists.

Restrictions: Array must be explicitly listed in the dispatcher's type.

Usecase: Used when one or more arrays have known implementations.

Example Usage:

An example here would be a filter that processes an input array of some integral type and produces either a `vtkDoubleArray` or a `vtkFloatArray`, depending on some condition. Since the input array's implementation is unknown (it comes from outside the filter), we'll rely on a ValueType-filtered version of `vtkArrayDispatch::Arrays` for its type. However, we know the output array is either `vtkDoubleArray` or `vtkFloatArray`, so we'll want to be sure to apply that restriction:

```
// input has an unknown implementation, but an integral ValueType.
vtkDataArray *input = ...;

// Output is always either vtkFloatArray or vtkDoubleArray:
vtkDataArray *output = someCondition ? vtkFloatArray::New()
                                     : vtkDoubleArray::New();

// Define the valid ArrayTypes for input by filtering
// vtkArrayDispatch::Arrays to remove non-integral types:
typedef typename vtkArrayDispatch::FilterArraysByValueType
  <
  vtkArrayDispatch::Arrays,
  vtkArrayDispatch::Integrals
  >::Result InputTypes;

// For output, create a new vtkTypeList with the only two possibilities:
typedef vtkTypeList::Create<vtkFloatArray, vtkDoubleArray> OutputTypes;

// Typedef the dispatch to a more manageable name:
typedef vtkArrayDispatch::Dispatch2ByArray
  <
  InputTypes,
  OutputTypes
  > MyDispatch;
```

(continues on next page)

(continued from previous page)

```
// Execute the dispatch:
MyDispatch::Execute(input, output, someWorker);
```

vtkArrayDispatch::DispatchByValueType

This family of dispatchers takes a vtkTypeList of ValueTypes for each array and restricts dispatch to only arrays in vtkArrayDispatch::Arrays that have one of the specified value types.

Variations:

- `vtkArrayDispatch::DispatchByValueType`: Single dispatch.
- `vtkArrayDispatch::Dispatch2ByValueType`: Double dispatch.
- `vtkArrayDispatch::Dispatch3ByValueType`: Triple dispatch.

Arrays considered: All arrays in `vtkArrayDispatch::Arrays` that meet the ValueType requirements.

Restrictions: Arrays that do not satisfy the ValueType requirements are eliminated.

Usecase: Used when one or more of the dispatched arrays has an unknown implementation, but a known (or restricted) ValueType.

Example Usage:

Here we'll consider a filter that processes three arrays. The first is a complete unknown. The second is known to hold `unsigned char`, but we don't know the implementation. The third holds either `doubles` or `floats`, but its implementation is also unknown.

```
// Complete unknown:
vtkDataArray *array1 = ...;
// Some array holding unsigned chars:
vtkDataArray *array2 = ...;
// Some array holding either floats or doubles:
vtkDataArray *array3 = ...;

// Typedef the dispatch to a more manageable name:
typedef vtkArrayDispatch::Dispatch3ByValueType
  <
  vtkArrayDispatch::AllTypes,
  vtkTypeList::Create<unsigned char>,
  vtkArrayDispatch::Reals
  > MyDispatch;

// Execute the dispatch:
MyDispatch::Execute(array1, array2, array3, someWorker);
```

vtkArrayDispatch::DispatchByArrayWithSameValueType

This family of dispatchers takes a `vtkTypeList` of ArrayTypes for each array and restricts dispatch to only consider arrays from those typelists, with the added requirement that all dispatched arrays share a ValueType.

Variations:

- `vtkArrayDispatch::Dispatch2ByArrayWithSameValueType`: Double dispatch.
- `vtkArrayDispatch::Dispatch3ByArrayWithSameValueType`: Triple dispatch.

Arrays considered: All arrays in the explicit typelists that meet the ValueType requirements.

Restrictions: Combinations of arrays with differing ValueTypes are eliminated.

Usecase: When one or more arrays are known to belong to a restricted set of ArrayTypes, and all arrays are known to share the same ValueType, regardless of implementation.

Example Usage:

Let's consider a double array dispatch, with `array1` known to be one of four common array types (AOS `float`, `double`, `int`, and `vtkIdType` arrays), and the other is a complete unknown, although we know that it holds the same ValueType as `array1`.

```
// AOS float, double, int, or vtkIdType array:
vtkDataArray *array1 = ...;
// Unknown implementation, but the ValueType matches array1:
vtkDataArray *array2 = ...;

// array1's possible types:
typedef vtkTypeList;:Create<vtkFloatArray, vtkDoubleArray,
                            vtkIntArray, vtkIdTypeArray> Array1Types;

// array2's possible types:
typedef typename vtkArrayDispatch::FilterArraysByValueType
  <
  vtkArrayDispatch::Arrays,
  vtkTypeList::Create<float, double, int, vtkIdType>
  > Array2Types;

// Typedef the dispatch to a more manageable name:
typedef vtkArrayDispatch::Dispatch2ByArrayWithSameValueType
  <
  Array1Types,
  Array2Types
  > MyDispatch;

// Execute the dispatch:
MyDispatch::Execute(array1, array2, someWorker);
```

vtkArrayDispatch::DispatchBySameValueType

This family of dispatchers takes a single `vtkTypeList` of ValueType and restricts dispatch to only consider arrays from `vtkArrayDispatch::Arrays` with those ValueTypes, with the added requirement that all dispatched arrays share a ValueType.

Variations:

- `vtkArrayDispatch::Dispatch2BySameValueType`: Double dispatch.
- `vtkArrayDispatch::Dispatch3BySameValueType`: Triple dispatch.
- `vtkArrayDispatch::Dispatch2SameValueType`: Double dispatch using `vtkArrayDispatch::AllTypes`.
- `vtkArrayDispatch::Dispatch3SameValueType`: Triple dispatch using `vtkArrayDispatch::AllTypes`.

Arrays considered: All arrays in `vtkArrayDispatch::Arrays` that meet the ValueType requirements.

Restrictions: Combinations of arrays with differing ValueTypes are eliminated.

Usecase: When one or more arrays are known to belong to a restricted set of ValueTypes, and all arrays are known to share the same ValueType, regardless of implementation.

Example Usage:

Let's consider a double array dispatch, with `array1` known to be one of four common ValueTypes (`float`, `double`, `int`, and `vtkIdType` arrays), and `array2` known to have the same ValueType as `array1`.

```cpp
// Some float, double, int, or vtkIdType array:
vtkDataArray *array1 = ...;
// Unknown, but the ValueType matches array1:
vtkDataArray *array2 = ...;

// The allowed ValueTypes:
typedef vtkTypeList::Create<float, double, int, vtkIdType> ValidValueTypes;

// Typedef the dispatch to a more manageable name:
typedef vtkArrayDispatch::Dispatch2BySameValueType
  <
  ValidValueTypes
  > MyDispatch;

// Execute the dispatch:
MyDispatch::Execute(array1, array2, someWorker);
```

10.3.10 Advanced Usage

Accessing Memory Buffers

Despite the thin `vtkGenericDataArray` API's nice feature that compilers can optimize memory accesses, sometimes there are still legitimate reasons to access the underlying memory buffer. This can still be done safely by providing overloads to your worker's `operator()` method. For instance, `vtkDataArray::DeepCopy` uses a generic implementation when mixed array implementations are used, but has optimized overloads for copying between arrays with the same ValueType and implementation. The worker for this dispatch is shown below as an example:

```
// Copy tuples from src to dest:
struct DeepCopyWorker
{
  // AoS --> AoS same-type specialization:
  template <typename ValueType>
  void operator()(vtkAOSDataArrayTemplate<ValueType> *src,
                  vtkAOSDataArrayTemplate<ValueType> *dst)
  {
    std::copy(src->Begin(), src->End(), dst->Begin());
  }

  // SoA --> SoA same-type specialization:
  template <typename ValueType>
  void operator()(vtkSOADataArrayTemplate<ValueType> *src,
                  vtkSOADataArrayTemplate<ValueType> *dst)
  {
    vtkIdType numTuples = src->GetNumberOfTuples();
    for (int comp; comp < src->GetNumberOfComponents(); ++comp)
      {
      ValueType *srcBegin = src->GetComponentArrayPointer(comp);
      ValueType *srcEnd = srcBegin + numTuples;
      ValueType *dstBegin = dst->GetComponentArrayPointer(comp);

      std::copy(srcBegin, srcEnd, dstBegin);
      }
  }

  // Generic implementation:
  template <typename Array1T, typename Array2T>
  void operator()(Array1T *src, Array2T *dst)
  {
    vtkDataArrayAccessor<Array1T> s(src);
    vtkDataArrayAccessor<Array2T> d(dst);

    typedef typename vtkDataArrayAccessor<Array2T>::APIType DestType;

    vtkIdType tuples = src->GetNumberOfTuples();
    int comps = src->GetNumberOfComponents();

    for (vtkIdType t = 0; t < tuples; ++t)
      {
      for (int c = 0; c < comps; ++c)
        {
        d.Set(t, c, static_cast<DestType>(s.Get(t, c)));
        }
      }
  }
};
```

10.3.11 Putting It All Together

Now that we've explored the new tools introduced with VTK 7.1 that allow efficient, implementation agnostic array access, let's take another look at the `calcMagnitude` example from before and identify the key features of the implementation:

```
// Modern implementation of calcMagnitude using new concepts in VTK 7.1:
struct CalcMagnitudeWorker
{
  template <typename VectorArray, typename MagnitudeArray>
  void operator()(VectorArray *vectors, MagnitudeArray *magnitude)
  {
    VTK_ASSUME(vectors->GetNumberOfComponents() == 3);
    VTK_ASSUME(magnitude->GetNumberOfComponents() == 1);

    vtkDataArrayAccessor<VectorArray> v(vectors);
    vtkDataArrayAccessor<MagnitudeArray> m(magnitude);

    vtkIdType numVectors = vectors->GetNumberOfTuples();
    for (vtkIdType tupleIdx = 0; tupleIdx < numVectors; ++tupleIdx)
      {
      m.Set(tupleIdx, 0, std::sqrt(v.Get(tupleIdx, 0) * v.Get(tupleIdx, 0) +
                                   v.Get(tupleIdx, 1) * v.Get(tupleIdx, 1) +
                                   v.Get(tupleIdx, 2) * v.Get(tupleIdx, 2)));
      }
  }
};

void calcMagnitude(vtkDataArray *vectors, vtkDataArray *magnitude)
{
  CalcMagnitudeWorker worker;
  typedef vtkArrayDispatch::Dispatch2ByValueType
    <
      vtkArrayDispatch::AllTypes,
      vtkArrayDispatch::Reals
    > Dispatcher;

  if (!Dispatcher::Execute(vectors, magnitude, worker))
    {
    worker(vectors, magnitude); // vtkDataArray fallback
    }
}
```

This implementation:

- Uses dispatch restrictions to reduce the number of instantiated templated worker functions.
- Assuming 26 types are in `vtkArrayDispatch::Arrays` (13 AOS + 13 SOA).
- The first array is unrestricted. All 26 array types are considered.
- The second array is restricted to `float` or `double` ValueTypes, which translates to 4 array types (one each, SOA and AOS).
- 26 * 4 = 104 possible combinations exist. We've eliminated 26 * 22 = 572 combinations that an unrestricted double-dispatch would have generated (it would create 676 instantiations).

- The calculation is still carried out at `double` precision when the ValueType restrictions are not met.
- Just because we don't want those other 572 cases to have special code generated doesn't necessarily mean that we wouldn't want them to run.
- Thanks to `vtkDataArrayAccessor`, we have a fallback implementation that reuses our templated worker code.
- In this case, the dispatch is really just a fast-path implementation for floating point output types.
- The performance should be identical to iterating through raw memory buffers.
- The `vtkGenericDataArray` API is transparent to the compiler. The specialized instantiations of `operator()` can be heavily optimized since the memory access patterns are known and well-defined.
- Using `VTK_ASSUME` tells the compiler that the arrays have known strides, allowing further compile-time optimizations.

Hopefully this has convinced you that the `vtkArrayDispatch` and related tools are worth using to create flexible, efficient, typesafe implementations for your work with VTK. Please direct any questions you may have on the subject to the VTK Discourse forum.

10.4 Data Assembly

VTK 10.0 introduces a new mechanism for representing data hierarchies using vtkPartitionedDataSetCollection and vtkDataAssembly. This document describes the design details.

10.4.1 Data Model

The design is based on three classes:

- `vtkPartitionedDataSet` is a collection of datasets (not to be confused with `vtkDataSet`).
- `vtkPartitionedDataSetCollection` is a collection of `vtkPartitionedDataSets`.
- `vtkDataAssembly` defines the hierarchical relationships between items in a `vtkPartitionedDataSetCollection`.

Partitioned Dataset

`vtkPartitionedDataSet` is simply a collection of datasets that are to be treated as a logical whole. In data-parallel applications, each dataset may represent a partition of the complete dataset on the current worker process, rank, or thread. Each dataset in a `vtkPartitionedDataSet` is called a **partition**, implying it is only a part of a whole.

All non-null partitions have similar field and attribute arrays. For example, if a `vtkPartitionedDataSet` comprises of `vtkDataSet` subclasses, all will have exactly the same number of point data/cell data arrays, with same names, same number of components, and same data types.

Partitioned Dataset Collection

`vtkPartitionedDataSetCollection` is a collection of `vtkPartitionedDataSet`. Thus, it is simply a mechanism to group multiple `vtkPartitionedDataSet` instances together. Since each `vtkPartitionedDataSet` represents a whole dataset (not be confused with `vtkDataSet`), we can refer to each item in a `vtkPartitionedDataSetCollection` as a **partitioned-dataset**.

Unlike items in the `vtkPartitionedDataSet`, there are no restrictions of consistency between each items, partitioned-datasets, in the `vtkPartitionedDataSetCollection`. Thus, in the multiblock-dataset parlance, each item in this collection can be thought of as a block.

Data Assembly

`vtkDataAssembly` is a means to define an hierarchical organization of items in a `vtkPartitionedDataSetCollection`. This is literally a tree made up of named nodes. Each node in the tree can have associated dataset-indices. For a `vtkDataAssembly` is associated with a `vtkPartitionedDataSetCollection`, each of the dataset-indices is simply the index of a partitioned-dataset in the `vtkPartitionedDataSetCollection`. A dataset-index can be associated with multiple nodes in the assembly, however, a dataset-index cannot be associated with the same node more than once.

An assembly provides an ability to define a more complex view of the raw data blocks in a more application-specific form. This is not much different than what could be achieved using simply a `vtkMultiBlockDataSet`. However, there are several advantages to this separation of storage (`vtkPartitionedDataSetCollection`) and organization (`vtkDataAssembly`). These will become clear as we cover different use-cases.

While nodes in the data-assembly have unique ids, public facing algorithm APIs should not use them. For example an extract-block filter that allows users to choose which blocks (rather partitioned-datasets) to extract from vtkPartitionedDataSetCollection can expose an API that lets users provide path-expression to identify nodes in the associated data-assembly using their names.

Besides accessing nodes by querying using their names, `vtkDataAssembly` also supports a mechanism to iterate over all nodes in depth-first or breadth-first order using a *visitor*. vtkDataAssemblyVisitor defines a API that can be implemented to do custom action as each node in the tree is visited.

10.4.2 Design Implications

1. Since `vtkPartitionedDataSet` is simply parts of a whole, there is no specific significance to the number of partitions. In distributed pipelines, for example, a `vtkPartitionedDataSet` on each rank can have arbitrarily many partitions. Furthermore, filters can add/remove partitions as needed. Since the `vtkDataAssembly` never refers to individual partitions, this has no implication to filters that use the hierarchical relationships.
2. When constructing `vtkPartitionedDataSetCollection` in distributed data-parallel cases, each rank should have exactly the same number of partitioned-datasets. In this case, each `vtkPartitionedDataSet` at a specific index across all ranks together is treated as a whole dataset. Similarly, the `vtkDataAssembly` on each should be identical.
3. When developing filters, it is worth considering whether the filter really is a `vtkPartitionedDataSetCollection` filter or simply a `vtkPartitionedDataSet`-aware filter that needs to operate on each `vtkPartitionedDataSet` individually. For example, typical multiblock-aware filters like ghost-cell-generation, data-redistribution, etc., are simply `vtkPartitionedDataSet` filters. For `vtkPartitionedDataSet`-only filters, when the input is a `vtkPartitionedDataSetCollection`, the executive takes care of looping over each of the partitioned-dataset in the collection, thus simplifying the filter development.
4. Filters that don't change the number of partitioned-datasets in a vtkPartitionedDataSetCollection don't generally affect the relationships between the partitioned-datasets and hence can largely pass through the vtkDataAssembly.

Only filter like extract-block that remove partitioned-datasets need to update the vtkDataAssembly. There too, vtkDataAssembly provides several convenience methods to update the tree with ease.

5. It is possible to develop a mapper that uses the `vtkDataAssembly`. Using APIs that let users use path-queries to specify rendering properties for various nodes, the mapper can support use-cases where the input structure keeps changing but the relationships remain largely intact. Since the same dataset-index can be associated with multiple nodes in a `vtkDataAssembly`, the mapper can effectively support scene-graph like capabilities where user can specify transforms, and other rendering parameters, while reusing the heavy datasets. The mapper can easily tell if a dataset has already been uploaded to the rendering pipeline since it will have the same id and indeed be the same instance even if is being visited through different branches in the tree.

10.5 VTK Legacy Reader/Writer Information Format

10.5.1 Overview

The legacy vtk data file readers / writers store certain `vtkInformation` entries that are set on `vtkAbstractArray`'s `GetInformation()` object. Support is currently limited to numeric and string information keys, both single- and vector-valued. Only the information objects attached to arrays are encoded.

10.5.2 Array Metadata Blocks

A block of metadata may immediately follow the specification of an array. Whitespace is permitted between the array data and the opening `METADATA` tag. The metadata block is terminated by an empty line.

```
# vtk DataFile Version 4.1
vtk output
ASCII
DATASET UNSTRUCTURED_GRID
POINTS 6 float
0 0 0 1 0 0 0.5 1 0
0.5 0.5 1 0.5 -1 0 0.5 -0.5 1

METADATA
COMPONENT_NAMES
X%20coordinates
Y%20coordinates
Z%20coordinates
INFORMATION 8
NAME Double LOCATION TestKey
DATA 1
NAME DoubleVector LOCATION TestKey
DATA 3 1 90 260
NAME IdType LOCATION TestKey
DATA 5
NAME String LOCATION TestKey
DATA Test%20String!%0ALine2
NAME Integer LOCATION TestKey
DATA 408
NAME IntegerVector LOCATION TestKey
DATA 3 1 5 45
NAME StringVector LOCATION TestKey
```

(continues on next page)

(continued from previous page)

```
DATA 3
First
Second%20(with%20whitespace!)
Third%20(with%0Anewline!)
NAME UnsignedLong LOCATION TestKey
DATA 9

CELLS 3 15
4 0 1 2 3
4 0 4 1 5
4 5 3 1 0

CELL_TYPES 3
10
10
10

CELL_DATA 3
FIELD FieldData 1
vtkGhostType 1 3 unsigned_char
0 1 1
METADATA
COMPONENT_NAMES
Ghost%20level%20information
INFORMATION 1
NAME UNITS_LABEL LOCATION vtkDataArray
DATA radians
```

As shown, a metadata block can have two sections, `COMPONENT_NAMES` and `INFORMATION`. The `INFORMATION` tag is followed by the number of information keys that follow.

COMPONENT_NAMES

If the `METADATA` block contains the line `COMPONENT_NAMES`, the following lines are expected to be encoded strings containing the names of each component. There must be one line per component.

INFORMATION

If the `METADATA` block contains the line `INFORMATION`, the number of information keys is read from the INFORMATION line and `vtkInformation` data that follows is parsed. The general form of a single valued information entry is:

```
NAME [key name] LOCATION [key location (e.g. class name)]
DATA [value]
```

A vector information key is generally represented as:

```
NAME [key name] LOCATION [key location (e.g. class name)]
DATA [vector length] [value0] [value1] [value2] ...
```

The exception is a string vector, which contains encoded entries separated by newlines.

Specific examples of supported key types:

vtkInformationDoubleKey

```
NAME Double LOCATION TestKey
DATA 1
```

vtkInformationDoubleVectorKey

```
NAME DoubleVector LOCATION TestKey
DATA 3 1 90 260
```

vtkInformationIdTypeKey

```
NAME IdType LOCATION TestKey
DATA 5
```

vtkInformationStringKey

```
NAME String LOCATION TestKey
DATA Test%20String!%0ALine2
```

vtkInformationIntegerKey

```
NAME Integer LOCATION TestKey
DATA 408
```

vtkInformationIntegerVectorKey

```
NAME IntegerVector LOCATION TestKey
DATA 3 1 5 45
```

vtkInformationStringVectorKey

```
NAME StringVector LOCATION TestKey
DATA 3
First
Second%20(with%20whitespace!)
Third%20(with%0Anewline!)
```

vtkInformationUnsignedLongKey

```
NAME UnsignedLong LOCATION TestKey
DATA 9
```

10.6 VTK XML Reader/Writer Information Format

10.6.1 Overview

The vtk xml data file readers / writers store certain `vtkInformation` entries that are set on `vtkAbstractArray`'s `GetInformation()` object. Support is currently limited to numeric and string information keys, both single- and vector-valued. Only the information objects attached to arrays are written/read.

10.6.2 Array Information

Array information is embedded in the `<DataArray>` XML element as a series of `<InformationKey>` elements. The required attributes `name` and `location` specify the name and location strings associated with the key – for instance, the `vtkDataArray::UNITS_LABEL()` key has `name="UNITS_LABEL"` and `location="vtkDataArray"`. The `length` attribute is required for vector keys.

```
<DataArray [...]>
  <InformationKey name="KeyName" location="KeyLocation" [ length="N" ]>
    [...]
  </InformationKey>
  <InformationKey [...]>
    [...]
  </InformationKey>
  [...]
</DataArray>
```

Specific examples of supported key types:

vtkInformationDoubleKey

```
<InformationKey name="Double" location="XMLTestKey">
  1
</InformationKey>
```

vtkInformationDoubleVectorKey

```
<InformationKey name="DoubleVector" location="XMLTestKey" length="3">
  <Value index="0">
    1
  </Value>
  <Value index="1">
    90
  </Value>
  <Value index="2">
    260
  </Value>
</InformationKey>
```

vtkInformationIdTypeKey

```
<InformationKey name="IdType" location="XMLTestKey">
  5
</InformationKey>
```

vtkInformationStringKey

```
<InformationKey name="String" location="XMLTestKey">
  Test String!
Line2
</InformationKey>
```

vtkInformationIntegerKey

```
<InformationKey name="Integer" location="XMLTestKey">
  408
</InformationKey>
```

vtkInformationIntegerVectorKey

```
<InformationKey name="IntegerVector" location="XMLTestKey" length="3">
  <Value index="0">
    1
  </Value>
  <Value index="1">
    5
  </Value>
  <Value index="2">
    45
  </Value>
</InformationKey>
```

vtkInformationStringVectorKey

```
<InformationKey name="StringVector" location="XMLTestKey" length="3">
  <Value index="0">
    First
  </Value>
  <Value index="1">
    Second (with whitespace!)
  </Value>
  <Value index="2">
    Third (with
newline!)
  </Value>
</InformationKey>
```

vtkInformationUnsignedLongKey

```
<InformationKey name="UnsignedLong" location="XMLTestKey">
  9
</InformationKey>
```

10.7 Field Data as Time Meta-Data in VTK XML File Formats

As of VTK 8.2, VTK XML readers and writers support embedding time meta-data as a field array. This is demonstrated best with an example:

```
<VTKFile type="PolyData" version="1.0" byte_order="LittleEndian" header_type="UInt64">
  <PolyData>
    <FieldData>
      <DataArray type="Float64" Name="TimeValue" NumberOfTuples="1">1.24
      </DataArray>
    </FieldData>
    ...
</VTKFile>
```

Here TimeValue is a regular double precision array that has a single value of 1.24. The XML readers will treat this array in a special way. When they encounter this array during the meta-data stage (`RequestInformation()`), they will read the value from this array and generate a `vtkStreamingDemandDrivenPipeline::TIME_STEPS()` key in the output information containing this value.

In addition, the XML writers will generate a field array of name TimeValue in the output, if they encounter time value in their input (`vtkDataObject::DATA_TIME_STEP()`). This is done even if the data does not have a TimeValue array. Furthermore, even such an array exists, it will be replaced with one that contains the value from `vtkDataObject::DATA_TIME_STEP()` to make sure that the value is consistent with the pipeline value.

This change may appear pointless on its own as a single time value is not very useful. Its main use is when reading file series as it is done by ParaView's file (time) series readers.

10.8 MomentInvariants Architecture

10.8.1 Rotation-invariant Pattern Detection

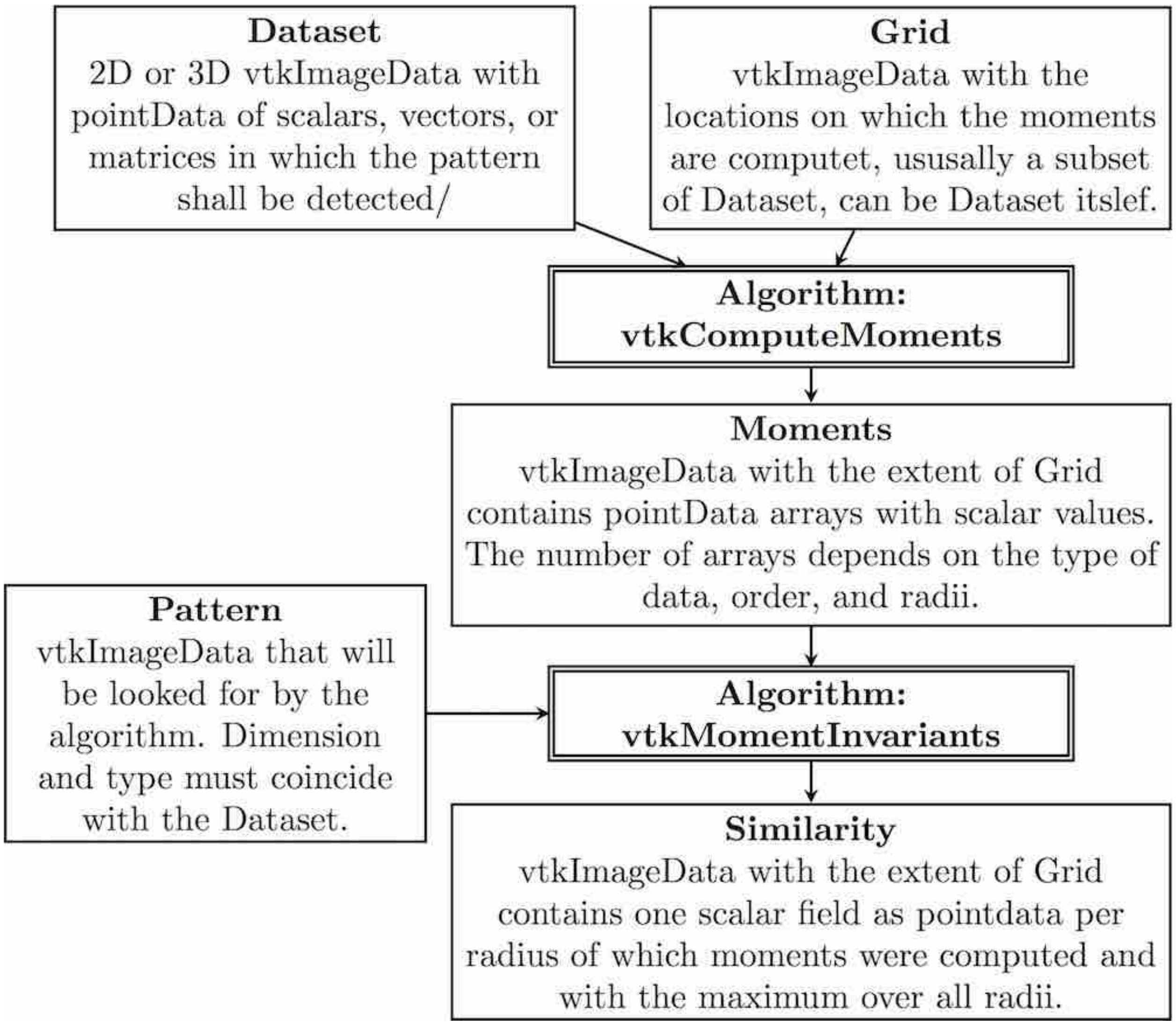
For pattern detection, the orientation of the pattern is usually not known a priory. The process should not be decelerated more than necessary while the pattern detection algorithm looks for all possible rotated copies of the template. Therefore, rotation invariance is a critical requirement. Moment invariants can achieve rotation invariance without the need for point to point correlations, which are difficult to generate in smooth fields. For an introduction, we recommend

Flusser, J., Suk, T., & Zitová, B. (2016). 2D and 3D Image Analysis by Moments. John Wiley & Sons.

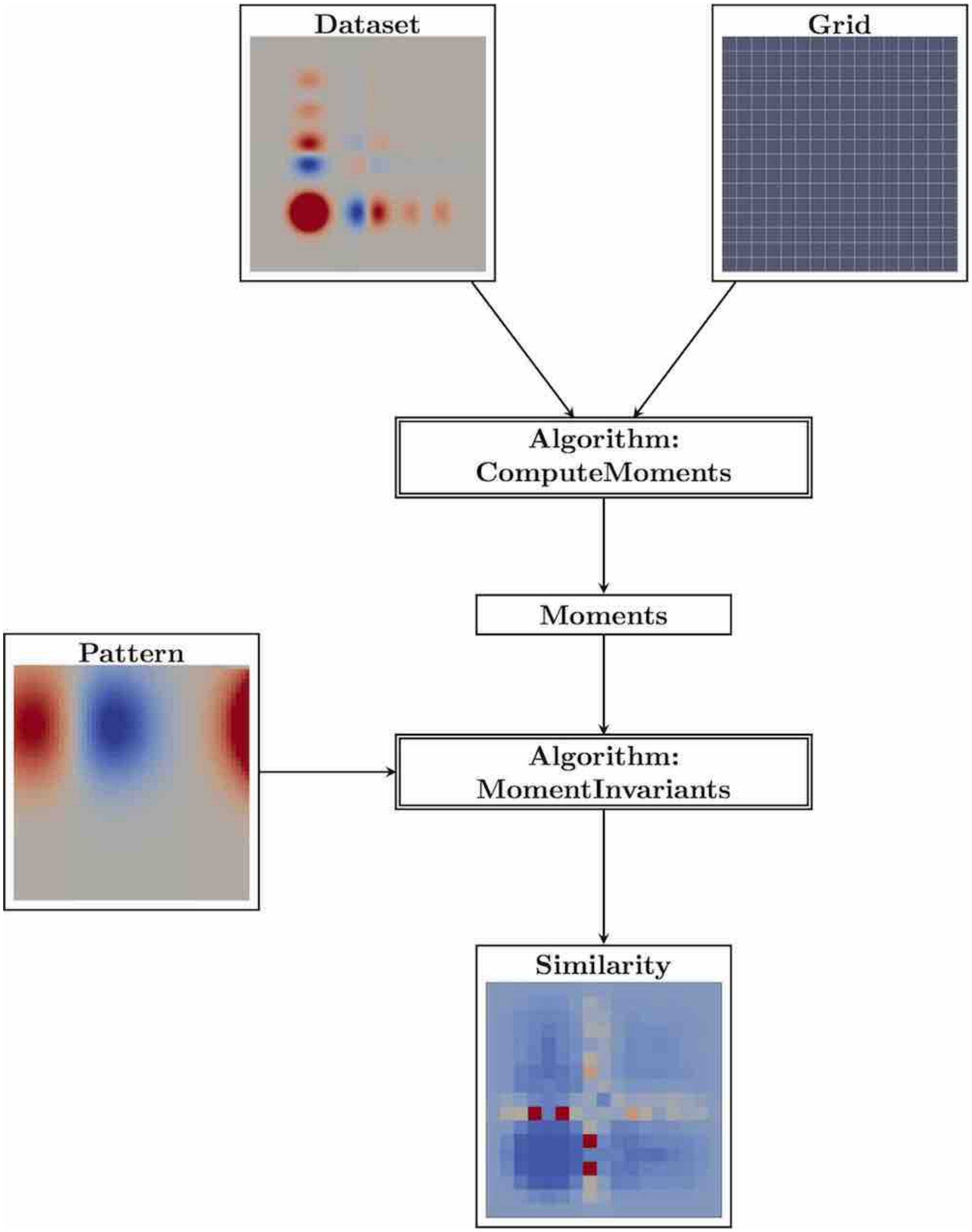
We have implemented the prototypes of two vtk filters that together are able to perform pattern detection. The algorithm, which we used, is described in

Bujack, R., & Hagen, H. (2017). Moment Invariants for Multi-Dimensional Data. In Modeling, Analysis, and Visualization of Anisotropy (pp. 43-64). Springer, Cham.

The first filter computes the moments and the second one performs the normalization based on the given pattern and computes the similarity. They are able to handle two- and three-dimensional scalar, vector, and matrix fields in the format of a vtkImageData. The architecture with inputs and outputs and their types can be found in the following figure.


Dataset
2D or 3D vtkImageData with pointData of scalars, vectors, or matrices in which the pattern shall be detected/
Grid
vtkImageData with the locations on which the moments are computet, ususally a subset of Dataset, can be Dataset itslef.
Algorithm: vtkComputeMoments
Moments
vtkImageData with the extent of Grid contains pointData arrays with scalar values. The number of arrays depends on the type of data, order, and radii.
Pattern
vtkImageData that will be looked for by the algorithm. Dimension and type must coincide with the Dataset.
Algorithm: vtkMomentInvariants
Similarity
vtkImageData with the extent of Grid contains one scalar field as pointdata per radius of which moments were computed and with the maximum over all radii.

The architecture illustrated with example images is shown the following figure.


Dataset
Grid
Algorithm:
ComputeMoments
Moments
Pattern
Algorithm:
MomentInvariants
Similarity

10.8.2 Extensions

The **MomentInvariants** module contains actually a bunch of extra algorithms and helper classes.

The class **vtkMomentsHelper** provides functions for the moments computation that will be needed by vtkCompute-Moments and vtkMomentInvariants.

The class **vtkMomentsTensor** provides the functionality to treat tensors of arbitrary dimension and rank. It supports addition, outer product, and contractions.

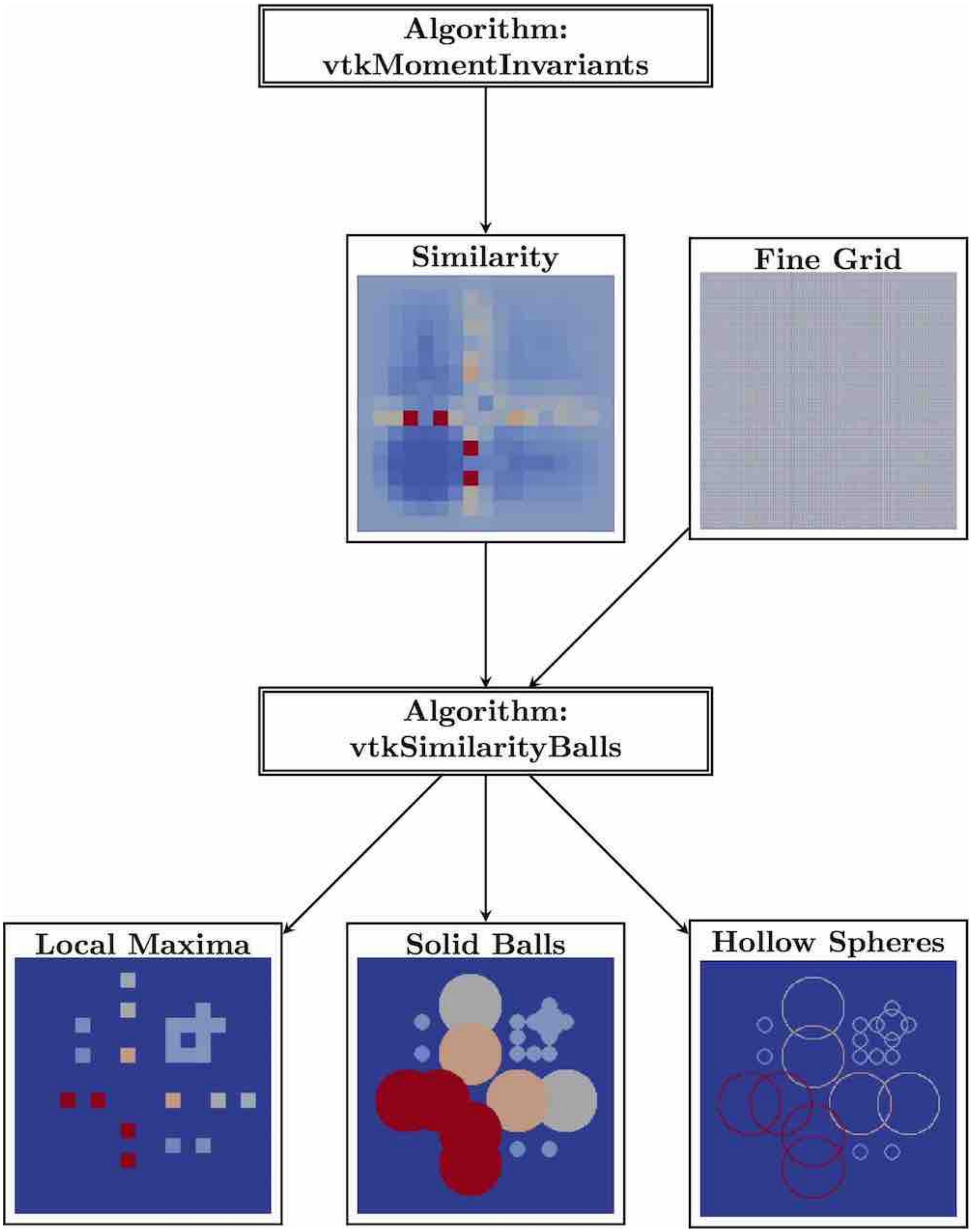
The algorithm **vtkSimilarityBalls** is a filter that takes the similarity field as produced by vtkMomentInvariants and a grid of type vtkImageData. It computes the local maxima in space plus scale and produces the output localMaxSimilarity that contains the similarity value together with the corresponding radius at the maxima. All other points are zero. For further visualization, it also produces two output fields that encode the radius through drawing a solid ball or a hollow sphere around those places. The second input, i.e. the grid, steers the resolution of the balls. It is helpful if its extent is a multiple of the first input's. Then, the circles are centered nicely. The spheres/circles are good for 2D visualizations, because they can be laid over a visualization of the field. The balls are good for 3D volume rendering or steering of the seeding of visualization elements. The 2D visualization is described in

Bujack, R., Hotz, I., Scheuermann, G., & Hitzer, E. (2015). Moment invariants for 2D flow fields via normalization in detail. IEEE transactions on visualization and computer graphics, 21(8), 916-929

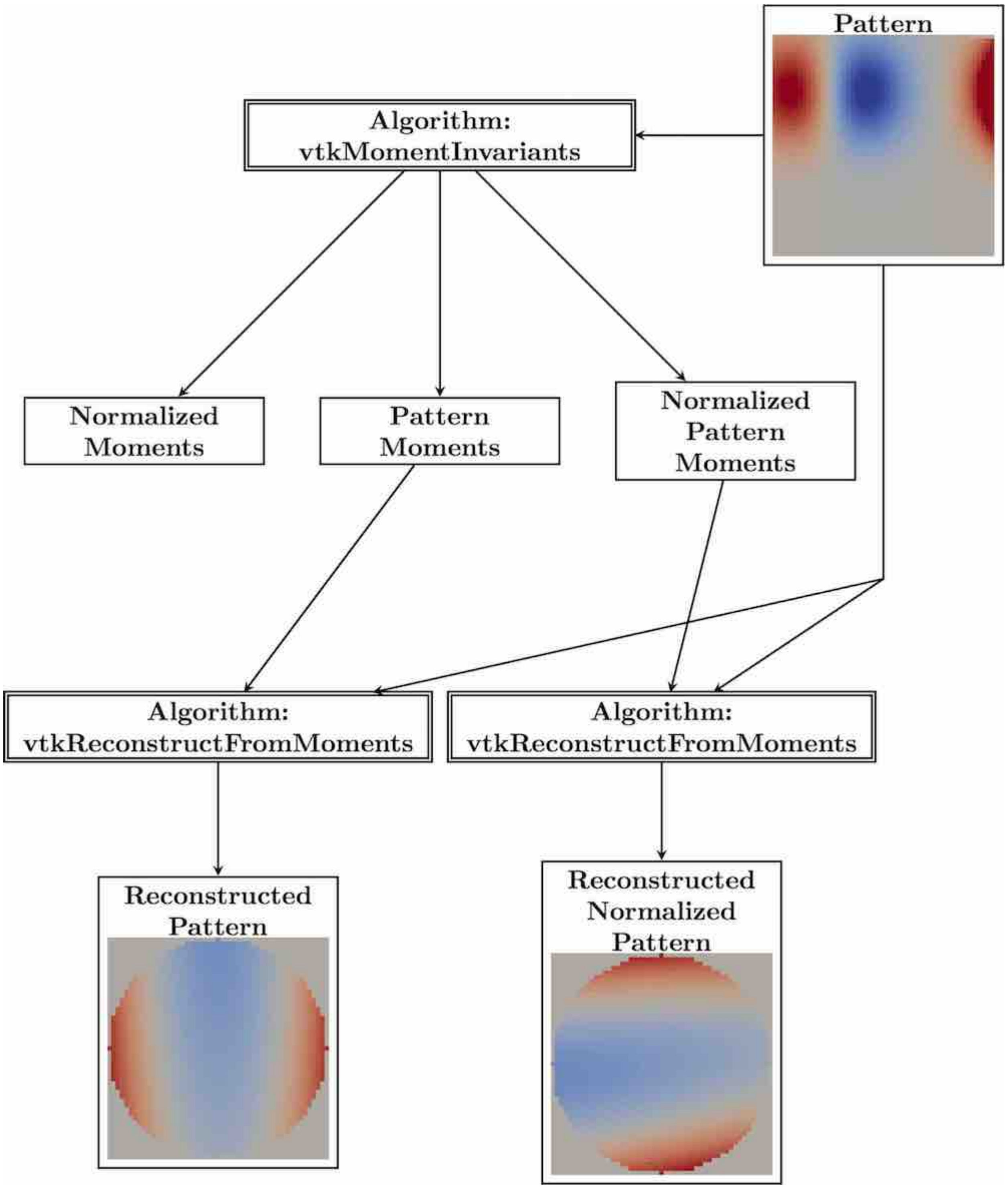
and the 3D counterpart in

Bujack, R., Kasten, J., Hotz, I., Scheuermann, G., & Hitzer, E. (2015, April). Moment invariants for 3D flow fields via normalization. In Visualization Symposium (PacificVis), 2015 IEEE Pacific (pp. 9-16). IEEE.

A schematic overview of the use of vtkSimilarityBalls with example images is given in the following Figure.

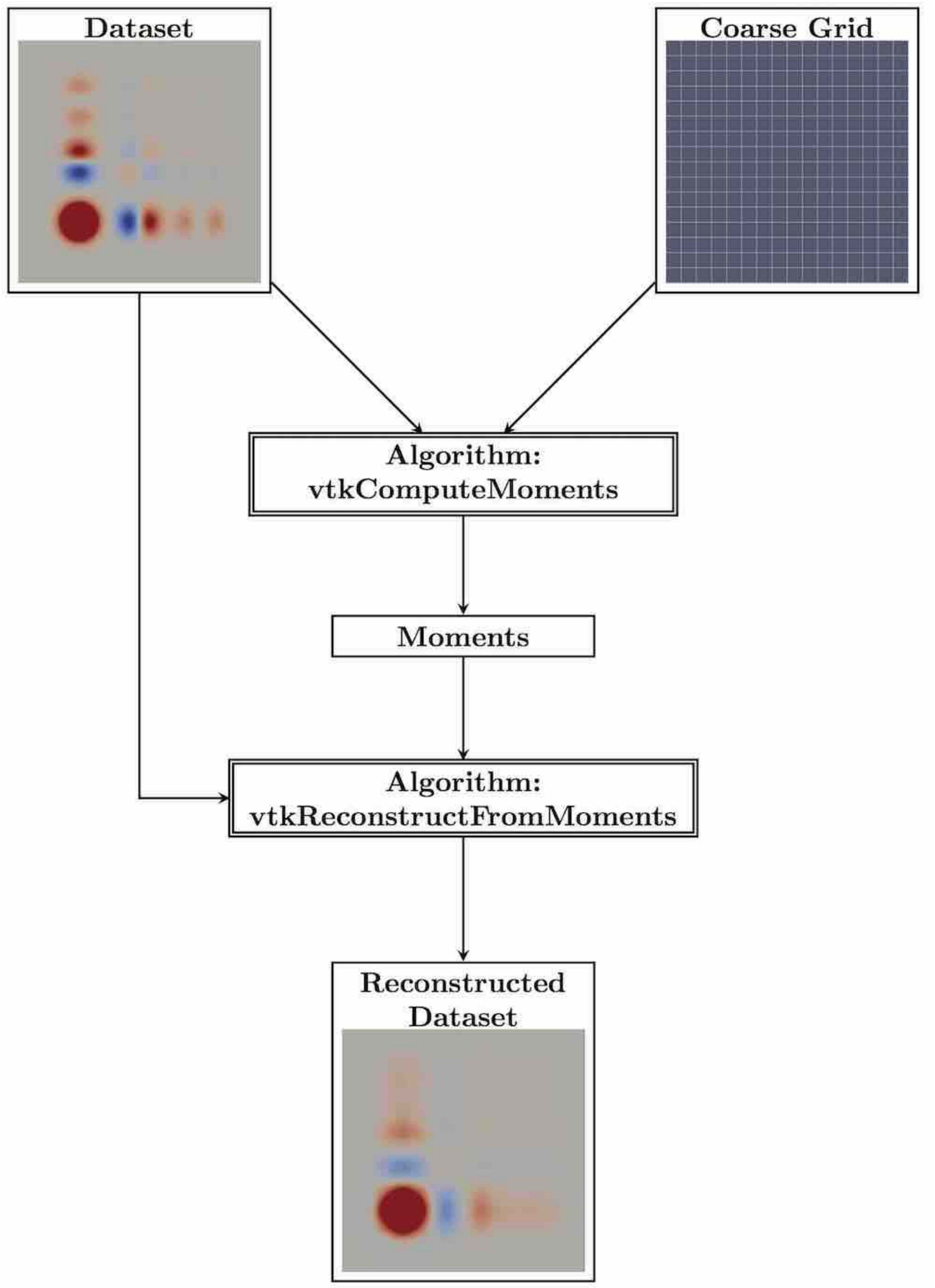


The algorithm **vtkReconstructFromMoments** is a filter that takes the momentData as produced by vtkComputeMoments or vtkMomentInvariants and a grid. It reconstructs the function from the moments, just like from the coefficients of a Taylor series. For the reconstruction, we need to orthonormalize the moments first. Then, we multiply the coefficients with their corresponding basis function and add them up. There are in principal three applications. First, if we put in the moments of the pattern and the grid of the pattern, we see which parts of the template the algorithm can actually grasp with the given order during the pattern detection. Tte following Figure shows images created using moments up to second order.


Pattern
Algorithm: vtkMomentInvariants
Normalized Moments
Pattern Moments
Normalized Pattern Moments
Algorithm: vtkReconstructFromMoments
Algorithm: vtkReconstructFromMoments
Reconstructed Pattern
Reconstructed Normalized Pattern

Second, if we put in the normalized moments of the pattern and the grid of the pattern, we can see how the standard position looks like. There might be several standard positions due to the ambiguity of the eigenvectors that differ by rotations of 180 degree and possibly a reflection. The algorithm will use the first one. In the previous Figure, the reflection along the x-axis would also be a standard position.

Third, if we put in the moments of the field and the original field data, we can see how well the subset of points, on which the moments were computed, actually represents the field. The following Figure depicts an example using a 16 x 16 coarse grid and moments up to second order.


Dataset
Coarse Grid
Algorithm:
vtkComputeMoments
Moments
Algorithm:
vtkReconstructFromMoments
Reconstructed
Dataset

CHAPTER ELEVEN

DEVELOPER'S GUIDE

This guide is a comprehensive resource for contributing to VTK – for both new and experienced contributors. We welcome your contributions to VTK !

11.1 Develop

This page documents how to develop VTK using GitLab and Git. See the README for more information.

Git is an extremely powerful version control tool that supports many different "workflows" for individual development and collaboration. Here we document procedures used by the VTK development community. In the interest of simplicity and brevity we do *not* provide an explanation of why we use this approach.

For a quickstart guide see here

11.1.1 Workflow

VTK development uses a branchy workflow based on topic branches. Our collaboration workflow consists of three main steps:

1. Local Development:
   - *Update*
   - *Create a Topic*
2. Code Review (requires GitLab Access):
   - *Share a Topic*
   - *Create a Merge Request*
   - *Review a Merge Request*
   - *Revise a Topic*
3. Integrate Changes:
   - *Merge a Topic* (requires permission in GitLab)
   - *Delete a Topic*

11.1.2 Update

1. Update your local `master` branch:

   ```
   $ git checkout master
   $ git pull
   ```

2. Optionally push `master` to your fork in GitLab:

   ```
   $ git push gitlab master
   ```

   to keep it in sync. The `git gitlab-push` script used to *Share a Topic* below will also do this.

11.1.3 Create a Topic

All new work must be committed on topic branches. Name topics like you might name functions: concise but precise. A reader should have a general idea of the feature or fix to be developed given just the branch name.

1. To start a new topic branch:

   ```
   $ git fetch origin
   ```

2. For new development, start the topic from `origin/master`:

   ```
   $ git checkout -b my-topic origin/master
   ```

   For release branch fixes, start the topic from `origin/release`, and by convention use a topic name starting in `release-`:

   ```
   $ git checkout -b release-my-topic origin/release
   ```

   If backporting a change, you may rebase the branch back onto `origin/release`:

   ```
   $ git checkout -b release-my-topic my-topic
   $ git rebase --onto origin/release origin/master
   ```

   Alternatively, for more targeted or aggregate backports, use the `-x` flag when performing `git cherry-pick` so that a reference to the original commit is added to the commit message:

   ```
   $ git checkout -b release-my-topic origin/release
   $ git cherry-pick -x $hash_a $hash_b $hash_c
   $ git cherry-pick -x $hash_d $hash_e $hash_f
   ```

3. Edit files and create commits (repeat as needed):

   ```
   $ edit file1 file2 file3
   $ git add file1 file2 file3
   $ git commit
   ```

   Caveats:

   - To add data follow *these instructions*.
   - If your change modifies third party code, see *Updating Third Party Projects*.
   - To deprecate APIs, see *Deprecation Process*.

11.1.4 Guidelines for Commit logs

Remember to *motivate & summarize*. When writing commit logs, make sure that there is enough information there for any developer to read and glean relevant information such as:

1. Is this change important and why?
2. If addressing an issue, which issue(s)?
3. If a new feature, why is it useful and/or necessary?
4. Are there background references or documentation?

A short description of what the issue being addressed and how will go a long way towards making the log more readable and the software more maintainable.

Style guidelines for commit logs are as follows:

1. Separate subject from body with a blank line
2. Limit the subject line to 60 characters
3. Capitalize the subject line
4. Use the imperative mood in the subject line e.g. "Refactor foo" or "Fix Issue #12322", instead of "Refactoring foo", or "Fixing issue #12322".
5. Wrap the body at 80 characters
6. Use the body to explain `what` and `why` and if applicable a brief `how`.

11.1.5 Share a Topic

When a topic is ready for review and possible inclusion, share it by pushing to a fork of your repository in GitLab. Be sure you have registered and signed in for GitLab Access and created your fork by visiting the main VTK GitLab repository page and using the "Fork" button in the upper right.

1. Checkout the topic if it is not your current branch:

   ```
   $ git checkout my-topic
   ```

2. Check what commits will be pushed to your fork in GitLab:

   ```
   $ git prepush
   ```

3. Push commits in your topic branch to your fork in GitLab:

   ```
   $ git gitlab-push
   ```

   Notes:

   - If you are revising a previously pushed topic and have rewritten the topic history, add `-f` or `--force` to overwrite the destination.
   - If the topic adds data see *this note*.
   - The `gitlab-push` script also pushes the `master` branch to your fork in GitLab to keep it in sync with the upstream `master`.

   The output will include a link to the topic branch in your fork in GitLab and a link to a page for creating a Merge Request.

11.1.6 Create a Merge Request

(If you already created a merge request for a given topic and have reached this step after revising it, skip to the *next step*.)

Visit your fork in GitLab, browse to the "**Merge Requests**" link on the left, and use the "**New Merge Request**" button in the upper right to reach the URL printed at the end of the *previous step*. It should be of the form:

```
https://gitlab.kitware.com/<username>/vtk/-/merge_requests/new
```

Follow these steps:

1. In the "**Source branch**" box select the `<username>/vtk` repository and the `my-topic` branch.
2. In the "**Target branch**" box select the `vtk/vtk` repository and the `master` branch. It should be the default.

   If your change is a fix for the `release` branch, you should still select the `master` branch as the target because the change needs to end up there too.

   For other `release` branches (e.g., `release-6.3`), merge requests should go directly to the branch (they are not tied with `master` in our workflow).
3. Use the "**Compare branches**" button to proceed to the next page and fill out the merge request creation form.
4. In the "**Title**" field provide a one-line summary of the entire topic. This will become the title of the Merge Request.

   Example Merge Request Title:

   ```
   Wrapping: Add Java 1.x support
   ```
5. In the "**Description**" field provide a high-level description of the change the topic makes and any relevant information about how to try it.

   - Use `@username` syntax to draw attention of specific developers. This syntax may be used anywhere outside literal text and code blocks. Or, wait until the *next step* and add comments to draw attention of developers.
   - If your change is a fix for the `release` branch, indicate this so that a maintainer knows it should be merged to `release`.
   - Optionally use a fenced code block with type `message` to specify text to be included in the generated merge commit message when the topic is *merged*.

   Example Merge Request Description:

   ~~~
   This branch requires Java 1.x which is not generally available yet.
   Get Java 1.x from ... in order to try these changes.

   ```message
   Add support for Java 1.x to the wrapping infrastructure.
   ```

   Cc: @user1 @user2
   ~~~
6. The "**Assign to**", "**Milestone**", and "**Labels**" fields may be left blank.
7. Use the "**Submit merge request**" button to create the merge request and visit its page.

11.1.7 Guidelines for Merge Requests

Remember to *motivate & summarize*. When creating a merge request, consider the reviewers and future perusers of the software. Provide enough information to motivate the merge request such as:

1. Is this merge request important and why?
2. If addressing an issue, which issue(s)?
3. If a new feature, why is it useful and/or necessary?
4. Are there background references or documentation?

Also provide a summary statement expressing what you did and if there is a choice in implementation or design pattern, the rationale for choosing a certain path. Notable software or data features should be mentioned as well.

A well written merge request will motivate your reviewers, and bring them up to speed faster. Future software developers will be able to understand the reasons why something was done, and possibly avoid chasing down dead ends, Although it may take you a little more time to write a good merge request, you'll likely see payback in faster reviews and better understood and maintainable software.

11.1.8 Review a Merge Request

Add comments mentioning specific developers using `@username` syntax to draw their attention and have the topic reviewed. After typing `@` and some text, GitLab will offer completions for developers whose real names or user names match.

Here is a list of developers usernames and their specific area of expertise. A merge request without a developer tagged has very low chance to be merged in a reasonable timeframe.

- @mwestphal: Qt, filters, data Model, widgets, parallel, anything else.
- @charles.gueunet: filters, data model, SMP, events, pipeline, computational geometry, distributed algorithms.
- @kmorel: General VTK Expertise, VTK-m accelerators.
- @demarle: Ray tracing.
- @will.schroeder: algorithms, computational geometry, filters, SPH, SMP, widgets, point cloud, spatial locators.
- @sujin.philip: VTK-m Accelerators, SMP, DIY.
- @yohann.bearzi: filters, data model, HTG, computational geometry, algorithms.
- @sebastien.jourdain: web, WebAssembly, Python, Java.
- @allisonvacanti: VTK-m, vtkDataArray, vtkArrayDispatch, vtk::Range, data model, text rendering.
- @sankhesh: volume rendering, Qt, OpenGL, widgets, vtkImageData, DICOM, VR.
- @ben.boeckel: CMake, module system, third-parties.
- @cory.quammen: readers, filters, data modeling, general usage, documentation.
- @seanm: macOS, Cocoa, cppcheck, clang.
- @spiros.tsalikis: filters, SMP, computational geometry.
- @thomas.galland: readers, filters, selection, VR.

If you would like to be included in this list, juste create a merge request.

Human Reviews

Reviewers may add comments providing feedback or to acknowledge their approval. When a human reviewers suggest a change, please take it into account or discuss your choices with the reviewers until an agreement is reached. At this point, please `resolve` the discussion by clicking on the dedicated button.

When all discussion have been addressed, the reviewers will either do another pass of comment or acknowledge their approval in some form.

Please be swift to address or discuss comments, it will increase the speed at which your changes will be merged.

Comments Formatting

Comments use GitLab Flavored Markdown for formatting. See GitLab documentation on Special GitLab References to add links to things like merge requests and commits in other repositories.

Lines of specific forms will be extracted during *merging* and included as trailing lines of the generated merge commit message.

A commit message consists of up to three parts which must be specified in the following order: the *leading line*, then *middle lines*, then *trailing lines*. Each part is optional, but they must be specified in this order.

Leading Line

The *leading* line of a comment may optionally be exactly one of the following votes followed by nothing but whitespace before the end of the line:

- `-1` or `:-1:` indicates "the change is not ready for integration".
- `+1` or `:+1:` indicates "I like the change". This adds an `Acked-by:` trailer to the merge commit message.
- `+2` indicates "the change is ready for integration". This adds a `Reviewed-by:` trailer to the merge commit message.
- `+3` indicates "I have tested the change and verified it works". This adds a `Tested-by:` trailer to the merge commit message.

Middle Lines

The middle lines of a comment may be free-form GitLab Flavored Markdown.

Trailing Lines

Zero or more *trailing* lines in the last section of a comment may each contain exactly one of the following votes followed by nothing but whitespace before the end of the line:

- `Rejected-by: me` means "The change is not ready for integration."
- `Acked-by: me` means "I like the change but defer to others."
- `Reviewed-by: me` means "The change is ready for integration."
- `Tested-by: me` means "I have tested the change and verified it works."

Each `me` reference may instead be an `@username` reference or a full `Real Name <user@domain>` reference to credit someone else for performing the review. References to `me` and `@username` will automatically be transformed into a real name and email address according to the user's GitLab account profile.

Fetching Changes

One may fetch the changes associated with a merge request by using the `git fetch` command line shown at the top of the Merge Request page. It is of the form:

```
$ git fetch https://gitlab.kitware.com/$username/vtk.git $branch
```

This updates the local `FETCH_HEAD` to refer to the branch.

There are a few options for checking out the changes in a work tree:

- One may checkout the branch:

  ```
  $ git checkout FETCH_HEAD -b $branch
  ```

  or checkout the commit without creating a local branch:

  ```
  $ git checkout FETCH_HEAD
  ```

- Or, one may cherry-pick the commits to minimize rebuild time:

  ```
  $ git cherry-pick ..FETCH_HEAD
  ```

Robot Reviews

The "Kitware Robot" automatically performs basic checks on the commits and adds a comment acknowledging or rejecting the topic. This will be repeated automatically whenever the topic is pushed to your fork again. A re-check may be explicitly requested by adding a comment with a single trailing *line*:

```
Do: check
```

A topic cannot be *merged* until the automatic review succeeds.

Continuous Integration

VTK uses GitLab CI to test its functionality. CI results are published to CDash and a link is added to the `External` stage of the CI pipeline by `@kwrobot`. Developers and reviewers should start jobs which make sense for the change using the following methods:

- The first thing to check is that CI is enabled in your fork of VTK. If you see a `CI/CD` item on the left sidebar in your fork's project, you're all set. If not, go to `Settings > General` and enable `CI/CD` for "Everyone With Access" under the "Visibility, project features, permissions" section.
- Merge request authors should visit their merge request's pipeline and click the "Play" button on one or more jobs manually. If the merge request has the "Allow commits from members who can merge to the target branch" check box enabled, VTK developers and maintainers may use the "Play" button as well. This flag is visible when editing the merge request. When in doubt, it's a good idea to run a few jobs as smoke tests to catch early build/test failures before a full CI run that would tie up useful resources. Note that, as detailed below, a full CI run is necessary before the request can be merged.
- VTK Project developers may trigger CI on a merge request by adding a comment with a command among the [trailing lines][#trailing-lines]:

  Do: test

`@kwrobot` will add an award emoji to the comment to indicate that it was processed and trigger all jobs that are awaiting manual interaction in the merge request's pipelines.

The `Do: test` command accepts the following arguments:

- `--named <regex>` or `-n <regex>`: Trigger jobs matching `<regex>` anywhere in their name. Job names may be seen on the merge request's Pipelines tab.
- `--stage <stage>` or `-s <stage>`: Only affect jobs in a given stage. Stage names may be seen on the merge request's Pipelines tab. Note that the stage names are determined by what is in the `.gitlab-ci.yml` file and may be capitalized in the web page, so lowercasing the webpage's display name for stages may be required.
- `--action <action>` or `-a <action>`: The action to perform on the jobs. Possible actions:
  - `manual` (the default): Start jobs awaiting manual interaction.
  - `unsuccessful`: Start or restart jobs which have not completed successfully.
  - `failed`: Restart jobs which have completed, but without success.
  - `completed`: Restart all completed jobs.

If the merge request topic branch is updated by a push, a new manual trigger using one of the above methods is needed to start CI again.

Before the merge, all the jobs, including tidy, must be run and reviewed, see below.

If you have any question about the CI process, do not hesitate to ask a CI maintainer:

- @ben.boeckel
- @mwestphal

Reading CI Results

Reading CI results is a very important part of the merge request process and is the responsibility of the author of the merge request, although reviewers can usually help. There are two locations to read the results, GitLab CI and CDash. Both should be checked and considered clean before merging.

To read GitLab CI result, click on the Pipelines tab then on the last pipeline. It is expected to be fully green. If there is a yellow warning job, please consult CDash. If there is a red failed job, click on it to see the reason for the failure. It should clearly appears at the bottom of the log. Possible failures are:

- Timeouts: please rerun the job and report to CI maintainers
- Memory related errors: please rerun the job and report to CI maintainers
- Testing errors: please consult CDash for more information, usually an issue in your code
- Non disclosed error: please consult CDash, usually a build error in your code

To read CDash results, on the job page, click on the "cdash-commit" external job which will open the commit-specific CDash page. Once it is open, make sure to show "All Build" on the bottom left of the page. CDash results displays error, warnings, and test failures for all the jobs. It is expected to be green *except* for the "NoRun" and "Test Timings" categories, which can be ignored.

- Configure warnings: there **must** not be any; to fix before the merge
- Configure errors: there **must** not be any; to fix before the merge
- Build warnings: there **must** not be any; to fix before the merge. If unrelated to your code, report to CI maintainers.
- Build errors: there **must** not be any; to fix before the merge. If unrelated to your code, rerun the job and report to CI maintainers.

- NotRun test : ignore; these tests have self-diagnosed that they are not relevant on the testing machine.
- Testing failure: there **should** not be any, ideally, to fix before the merge. If unrelated to your code, check the test history to see if it is a flaky test and report to CI maintainers.
- Testing success: if your MR creates or modifies tests, please check that your test are listed there.
- Test timings errors: can be ignored, but if it is all red, you may want to report it to CI maintainers.

To check the history of a failing test, on the test page, click on the "Summary" link to see a summary of the test for the day, then click on the date controls on the top of the page to go back in time. If the test fails on other MRs or on master, this is probably a flaky test, currently in the process of being fixed or excluded. A flaky test can be ignored.

As a reminder, here is our current policy regarding CI results. All the jobs must be run before merging, *including tidy*. Configure warnings and errors are not acceptable to merge and must be fixed. Build warning and errors are not acceptable to merge and must be fixed. Testing failure should be fixed before merging but can be accepted if a flaky test has been clearly identified.

11.1.9 Revise a Topic

If a topic is approved during GitLab review, skip to the *next step*. Otherwise, revise the topic and push it back to GitLab for another review as follows:

1. Checkout the topic if it is not your current branch:

   ```
   $ git checkout my-topic
   ```

2. To revise the 3rd commit back on the topic:

   ```
   $ git rebase -i HEAD~3
   ```

   (Substitute the correct number of commits back, as low as 1.) Follow Git's interactive instructions.

3. Return to the *above step* to share the revised topic.

11.1.10 Merge a Topic

Once review has concluded that the MR topic is ready for integration (at least one +2), authorized developers may add a comment with a single trailing *line*:

```
Do: merge
```

in order for your change to be merged into the upstream repository.

If your merge request has been already approved by developers but not merged yet, do not hesitate to tag an authorized developer and ask for a merge.

By convention, do not request a merge if any `-1` or `Rejected-by:` review comments have not been resolved and superseded by at least `+1` or `Acked-by:` review comments from the same user.

The `Do: merge` command accepts the following arguments:

- `-t <topic>`: substitute `<topic>` for the name of the MR topic branch in the constructed merge commit message.

Additionally, `Do: merge` extracts configuration from trailing lines in the MR description (the following have no effect if used in a MR comment instead):

- `Backport: release[:<commit-ish>]`: merge the topic branch into the `release` branch to backport the change. This is allowed only if the topic branch is based on a commit in `release` already. If only part of the topic branch should be backported, specify it as `:<commit-ish>`. The `<commit-ish>` may use git rev-parse syntax to reference commits relative to the topic `HEAD`. See additional backport instructions for details. For example:
- `Backport: release` Merge the topic branch head into both `release` and `master`.
- `Backport: release:HEAD~1^2` Merge the topic branch head's parent's second parent commit into the `release` branch. Merge the topic branch head to `master`.
- `Topic-rename: <topic>`: substitute `<topic>` for the name of the MR topic branch in the constructed merge commit message. It is also used in merge commits constructed by `Do: stage`. The `-t` option to a `Do: merge` command overrides any topic rename set in the MR description.

Merge Success

If the merge succeeds the topic will appear in the upstream repository `master` branch and the Merge Request will be closed automatically.

Merge Failure

If the merge fails (likely due to a conflict), a comment will be added describing the failure. In the case of a conflict, fetch the latest upstream history and rebase on it:

```
$ git fetch origin
$ git rebase origin/master
```

(If you are fixing a bug in the latest release then substitute `origin/release` for `origin/master`.)

Return to the *above step* to share the revised topic.

11.1.11 Delete a Topic

After a topic has been merged upstream the Merge Request will be closed. Now you may delete your copies of the branch.

1. In the GitLab Merge Request page a "**Remove Source Branch**" button will appear. Use it to delete the `my-topic` branch from your fork in GitLab.
2. In your work tree checkout and update the `master` branch:

   ```
   $ git checkout master
   $ git pull
   ```

3. Delete the local topic branch:

   ```
   $ git branch -d my-topic
   ```

   The `branch -d` command works only when the topic branch has been correctly merged. Use `-D` instead of `-d` to force the deletion of an unmerged topic branch (warning - you could lose commits).

11.2 Regression Testing

11.2.1 Testing and dashboard submitter setup

Regression testing in VTK takes the form of a set of programs, that are included in the VTK source code and enabled in builds configured through CMake to have the `VTK_BUILD_TESTING` flag turned on. Test pass/fail results are returned to CTest via a test program's exit code. VTK contains helper classes that do specific checks, such as comparing a produced image against a known valid one, that are used in many of the regression tests. Test results may be submitted to Kitware's CDash instance, were they will be gathered and displayed at http://open.cdash.org/index.php?project=VTK

All proposed changes to VTK are automatically tested on Windows, Mac and Linux machines. All changes that are merged into the master branch are subsequently tested again by more rigorously configured Windows, Mac and Linux continuous dashboard submitters. After 9PM Eastern Time, the master branch is again tested by a wider set of machines and platforms. These results appear in the next day's page.

At each step in the code integration path the developers who contribute and merge code are responsible for checking the test results to look for problems that the new code might have introduced. Plus signs in CDash indicate newly detected problems. Developers can correlate problems with contributions by logging in to CDash. Submissions that contain a logged in developer's change are highlighted with yellow dots.

It is highly recommended that developers test changes locally before submitting them. To run tests locally:

1. Configure with `VTK_BUILD_TESTING` set ON

   The exact set of tests created depends on many configuration options. Tests in non-default modules are only tested when those modules are purposefully enabled, the smoke tests described in the Coding Style section above are enabled only when the python or Tcl interpreter is installed, tests written in wrapped languages are only enabled when wrapping is turned on, etc.

2. Build.

   VTK tests are only available from the build tree.

3. Run ctest at the command line in the build directory or make the TESTING target in Visual Studio.

   As ctest runs the tests it prints a summary. You should expect 90% of the tests or better to pass if your VTK is configured correctly. Detailed results (which are also printed if you supply a –V argument to ctest) are put into the Testing/Temporary directory. The detailed results include the command line that ctest uses to spawn each test. Other particularly useful arguments are:

   ```
   --R TestNameSubstringToInclude to choose tests by name

   --E TestNameSubstringToExclude to reject tests by name

   --I start,stop,step to run a portion of the tests

   --j N to run N tests simultaneously.
   ```

Dashboard submitting machines work at a slightly higher level of abstraction that adds the additional stages of downloading, configuring and building VTK before running the tests, and submitting all results to CDash afterward. With a build tree in place you can run "ctest –D Experimental" to run at this level and submit the results to the experimental section of the VTK dashboard or "ctest –M Experimental -T Build –T Submit" etc to pick and choose from among the stages. When setting up a test submitter machine one should start with the experimental configuration and then, once the kinks are worked out, promote the submitter to the Nightly section.

The volunteer machines use cron or Windows task scheduler to run CMake scripts that configure a VTK build with specific options, and then run ctest –D as above. Within CDash, you can see each test machine's specific configuration by clicking on the Advanced View and then clicking on the note icon in the Build Name column. This is a useful starting point when setting up a new submitter. It is important that each submitter's dashboard script include the name of the person who configures or maintains the machine so that, when the machine has problems, the dashboard maintainer can address it.

For details about the Continuous Integration infrastructure hosted at Kitware see *here*.

11.2.2 Run-time environment of tests using `ctest`

When running a test using `ctest`, an extra empty environment variable is set: `VTK_TESTING`. One can catch this environment variable and know that the code is executed under ctest. In particular, `VTK_TESTING` is used to disable anti-aliasing in the constructor of `vtkOpenGLRenderWindow` for the sake of making comparing image baseline more robust against graphics drivers discrepancies.

11.3 Adding Tests

This page documents how to add test data while developing VTK with Git. See the README for more information.

11.3.1 Setup

The workflow below depends on local hooks to function properly. Follow the main developer setup instructions before proceeding. In particular, run SetupForDevelopment.sh:

```
$ ./Utilities/SetupForDevelopment.sh
```

11.3.2 Workflow

Our workflow for adding data integrates with our standard Git *development process*. Start by *creating a topic*. Return here when you reach the "edit files" step.

These instructions follow a typical use case of adding a new test with a baseline image.

Writing new tests

All new features that go into VTK must be accompanied by tests. This ensures that the feature works on many platforms and that it will continue to work as VTK evolves.

Tests for the classes in each module of VTK are placed underneath the module's Testing/ subdirectory. Modules that the tests depend upon beyond those that the module itself depends upon are declared with the TEST_DEPENDS argument in the `vtk.module` file. Test executables are added to VTK's build system by naming them in the `CMakeLists.txt` files in each Testing/ directory. In those `CMakeLists`, standard `add_executable()` + `add_test()` command pairs could be used, but the following macros defined in `vtkModuleTesting.cmake` are preferable as they consolidate multiple tests together, participate in VTK's modular build scripts, and ensure consistency:

- *`vtk_add_test_cxx()`*
- *`vtk_add_test_mpi()`*
- *`vtk_add_test_python()`*

Tests indicate success to CTest by returning `EXIT_SUCCESS` (0) and failure by returning `EXIT_FAILURE` (1). How the test determines what result to return is up to the developer. VTK contains a number of utilities for this task. For example, vtkRegressionTester is a helper class that does a fuzzy comparison of images drawn by VTK against known good baseline images and returns a metric that can be simply compared against a numeric threshold.

Many tests require data files to run. The image comparison tests for example need baseline images to compare against, and many tests open up one or more files to visualize.

The source code and data file versions are kept in sync because the Testing/Data directory contains, instead of the real files, similarly named files which contain only the SHA512 hash of the matching data files. During the build process, when CMake sees that a required data file is not available, it downloads it into the directory defined by the ExternalData_OBJECT_STORES cmake configuration entry. The test executables read all data from there. The default setting for ExternalData_OBJECT_STORES is the ExternalData directory underneath the VTK build tree.

To make a change to VTK that modifies or adds a new test data file, place the new version in the Testing/Data or directory (for input data files) or Module/Name/Testing/Data (for regression test images), and build (or run cmake). CMake will do the work of moving the original out of the way and replacing it with an SHA512 link file. When you push the new link file to Gitlab, `git pre-commit` hooks push the original file up to Kitware's data service, where everyone can retrieve it.

Add Test

1. Write a new test, e.g.

```
$ edit Some/Module/Testing/Cxx/MyTest.cxx
```

2. Edit the corresponding `CMakeLists.txt` file:

```
$ edit Some/Module/Testing/Cxx/CMakeLists.txt
```

   and add the test in a `vtk_add_test_...` call (which references baselines automatically).

3. For tests not using such a call, reference the data file in an `ExternalData_add_test` call. Specify the file inside `DATA{...}` using a path relative to the test directory:

```
$ edit Some/Module/Testing/Cxx/CMakeLists.txt
ExternalData_add_test("${_vtk_build_TEST_DATA_TARGET}"
  NAME ${_vtk_build_test}Cxx-MyTest
  COMMAND <VTK_MODULE_NAME>CxxTests MyTest
          ... -V DATA{../Data/Baseline/MyTest.png,:} ...
  )
```

4. Some tests may require additional files not referenced on the command line. For these files, add references to a `vtk_module_test_data` call (usually in the `Testing` parent directory). For example, adding `Testing/Data/lines.vtp` would mean adding `Data/lines.vtp` entry to the call (the `Testing` directory is part of the path that is looked in automatically.

```
vtk_module_test_data(
  Data/lines.vtp)
```

Notes:

- If the data file references other data files, e.g. `.mhd -> .raw`, read the ExternalData module documentation on "associated" files.
- Multiple baseline images and other series are handled automatically when the reference ends in the `,:` option. Read ExternalData module documentation for details.

Build and Run the Test

If you already have a data file, skip to the *next step* to add it. Otherwise, use the following steps to produce a test baseline image file. We assume a build tree has been previously generated by CMake.

1. Switch to the build tree:

   ```
   $ cd ../VTK-build
   ```

2. Run CMake:

   ```
   $ cmake .
   ```

   Since we have not yet created the baseline image data file, CMake will warn that it does not exist but proceed to generate the test anyway.

3. Build

   ```
   $ make
   ```

4. Run the test

   ```
   $ ctest -R MyTest
   ```

   It will fail but place the baseline image in `Testing/Temporary`.

5. Switch back to the source tree:

   ```
   $ cd ../VTK
   ```

Add Data

Copy the data file into your local source tree.

```
$ mkdir -p Some/Module/Testing/Data/Baseline
$ cp ../VTK-build/Testing/Temporary/MyTest.png Some/Module/Testing/Data/Baseline
```

Run CMake

1. Switch to the build tree:

   ```
   $ cd ../VTK-build
   ```

2. Run CMake:

   ```
   $ cmake .
   ```

   CMake will *move the original file*. Keep your own copy if necessary. See *below* to recover the original file.

   During configuration CMake will display a message such as:

   ```
   Linked Some/Module/Testing/Data/Baseline/MyTest.png.sha512 to ExternalData SHA512/..
   ↪.
   ```

   This means that CMake converted the file into a data object referenced by a “content link” named like the original file but with a `.sha512` extension. CMake also *renamed the original file*.

3. Build

   ```
   $ make
   ```

   During the build, the ExternalData module will make the data file available where the test expects to find it.

4. Run the test

   ```
   $ ctest -R MyTest
   ```

   It should pass using the new data file.

5. Switch back to the source tree:

   ```
   $ cd ../VTK
   ```

Commit

Continue to *create the topic* and edit other files as necessary. Add the content link and commit it along with the other changes:

```
$ git add Some/Module/Testing/Data/Baseline/MyTest.png.sha512
$ git add Some/Module/Testing/Data/CMakeLists.txt
$ git commit
```

The local `pre-commit` hook will display a message such as:

```
Some/Module/Testing/Data/Baseline/MyTest.png.sha512: Added content to Git at refs/data/
↪SHA512/...
Some/Module/Testing/Data/Baseline/MyTest.png.sha512: Added content to local store at .
↪ExternalData/SHA512/...
Content link Some/Module/Testing/Data/Baseline/MyTest.png.sha512 -> .ExternalData/SHA512/
↪...
```

This means that the pre-commit hook recognized that the content link references a new data object and *prepared it for upload*.

Push

Follow the instructions to *share the topic*. When you push it to GitLab for review using

```
$ git gitlab-push
```

part of the output will be of the form:

```
*       ...:refs/data/...      [new branch]
*       HEAD:refs/heads/my-topic  [new branch]
Pushed refs/data/... and removed local ref.
```

This means that the `git-gitlab-push` script pushed the topic and *uploaded the data* it references.

Options for `gitlab-push` include:

- `--dry-run`: Report push that would occur without actually doing it
- `--no-topic`: Push the data referenced by the topic but not the topic itself

Note: One must `git gitlab-push` from the same work tree as was used to create the commit. Do not `git push` to another computer first and try to push to GitLab from there because the data will not follow.

11.3.3 Building

Download

For the test data to be downloaded and made available to the tests in your build tree the `VTKData` target must be built. One may build the target directly, e.g. `make VTKData`, to obtain the data without a complete build. The output will be something like

```
-- Fetching ".../ExternalData/SHA512/..."
-- [download 100% complete]
-- Downloaded object: "VTK-build/ExternalData/Objects/SHA512/..."
```

The downloaded files appear in `VTK-build/ExternalData` by default.

Local Store

It is possible to configure one or more local ExternalData object stores shared among multiple builds. Configure for each build the advanced cache entry `ExternalData_OBJECT_STORES` to a directory on your local disk outside all build trees, e.g. `/home/user/.ExternalData`:

```
$ cmake -DExternalData_OBJECT_STORES=/home/user/.ExternalData ../VTK
```

The ExternalData module will store downloaded objects in the local store instead of the build tree. Once an object has been downloaded by one build it will persist in the local store for re-use by other builds without downloading again.

11.3.4 Discussion

A VTK test data file is not stored in the main source tree under version control. Instead the source tree contains a "content link" that refers to a data object by a hash of its content. At build time the ExternalData module fetches data needed by enabled tests. This allows arbitrarily large data to be added and removed without bloating the version control history.

The above *workflow* allows developers to add a new data file almost as if committing it to the source tree. The following subsections discuss details of the workflow implementation.

ExternalData

While *CMake runs* the ExternalData module evaluates *DATA{} references*. VTK sets in vtkExternalData.cmake the `ExternalData_LINK_CONTENT` option to `SHA512` to enable automatic conversion of raw data files into content links. When the module detects a real data file in the source tree it performs the following transformation as specified in the module documentation:

- Compute the SHA512 hash of the file
- Store the `${hash}` in a file with the original name plus `.sha512`
- Rename the original file to `.ExternalData_SHA512_${hash}`

The real data now sit in a file that we tell Git to ignore. For example:

```
$ cat Some/Module/Testing/Data/Baseline/.ExternalData_SHA512_477e6028* |sha512sum
477e6028...  -
$ cat Some/Module/Testing/Data/Baseline/MyTest.png.sha512
477e6028...
```

Recover Data File

To recover the original file after running CMake but before committing, undo the operation:

```
$ cd Some/Module/Testing/Data/Baseline
$ mv .ExternalData_SHA512_$(cat MyTest.png.sha512) MyTest.png
```

pre-commit

While *committing* a new or modified content link the pre-commit hook moves the real data object from the `.ExternalData_SHA512_${hash}` file left by the ExternalData module to a local object repository stored in a `.ExternalData` directory at the top of the source tree.

The hook also uses Git plumbing commands to store the data object as a blob in the local Git repository. The blob is not referenced by the new commit but instead by `refs/data/SHA512/${hash}`. This keeps the blob alive in the local repository but does not add it to the project history. For example:

```
$ git for-each-ref --format="%(refname)" refs/data
refs/data/SHA512/477e6028...
$ git cat-file blob refs/data/SHA512/477e6028... | sha512sum
477e6028...  -
```

git gitlab-push

The `git gitlab-push` command is actually an alias for the git-gitlab-push script. In addition to pushing the topic branch to GitLab the script also detects content links added or modified by the commits in the topic. It reads the data object hashes from the content links and looks for matching `refs/data/` entries in the local Git repository.

The script pushes the matching data objects to your VTK GitLab fork. For example:

```
$ git gitlab-push --dry-run --no-topic
*       refs/data/SHA512/477e6028...:refs/data/SHA512/477e6028...   [new branch]
Pushed refs/data/SHA512/477e6028... and removed local ref.
```

A GitLab webhook that triggers whenever a topic branch is pushed checks for `refs/data/` in your VTK GitLab fork, fetches them, erases the refs from your fork, and uploads them to a location that we tell ExternalData to search in [vtkExternalData][] at build time.

To verify that the data has been uploaded as expected, you may direct a web browser to the location where ExternalData has uploaded the files. For VTK, that location is currently `http://www.vtk.org/files/ExternalData/SHA512/XXXX` where `XXXX` is the complete SHA512 hash stored in the content link file (e.g., the text in `MyTest.png.sha512`).

Publishing Data for an External Branch

The above *workflow* works well for developers working on a single machine to contribute changes directly to upstream VTK. When working in an external branch of VTK, perhaps during a long-term topic development effort, data objects need to be published separately.

The workflow for adding data to an external branch of VTK is the same as the above through the *commit* step, but diverges at the *push* step because one will push to a separate repository. Our ExternalData infrastructure intentionally hides the real data files from Git so only the content links (`.sha512` files) will be pushed. The real data objects will still be left in the `.ExternalData/SHA512` directory at the top of the VTK source tree by the *pre-commit* hook.

The `.ExternalData` directory must be published somewhere visible to other machines that want to use the data, such as on a web server. Once that is done then other machines can be told where to look for the data, e.g.

```
cmake ../VTK "-DExternalData_URL_TEMPLATES=https://username.github.io/VTK/ExternalData/
↪%(algo)/%(hash)
```

In this example we assume the files are published on a Github Pages `gh-pages` branch in `username`'s fork of VTK.

Within the `gh-pages` branch the files are placed at `ExternalData/SHA512/$sha512sum` where `$sha512sum` is the SHA512 hash of the content (these are the same names they have in the `.ExternalData` directory in the original source tree).

11.4 Dashboard Scripts

This page documents how to use the VTK `dashboard` branch in Git. See the README for more information.

11.4.1 Using the Dashboard Scripts

The `dashboard` branch contains a dashboard client helper script. Use these commands to track it:

```
$ mkdir -p ~/Dashboards/VTKScripts
$ cd ~/Dashboards/VTKScripts
$ git init
$ git remote add -t dashboard origin https://gitlab.kitware.com/vtk/vtk.git
$ git pull origin
```

The `vtk_common.cmake` script contains setup instructions in its top comments.

Update the `dashboard` branch to get the latest version of this script by simply running:

```
$ git pull
```

Here is a link to the script as it appears today: vtk_common.cmake.

11.4.2 Changing the Dashboard Scripts

If you find bugs in the hooks themselves or would like to add new features, the can be edited in the usual Git manner:

```
$ git checkout -b my-topic-branch
```

Make your edits, test it, and commit the result. Create a patch file with:

```
$ git format-patch origin/dashboard
```

And post the results in the Development category in the VTK Discourse forum.

11.5 Updating Third Party Projects

When updating a third party project, any changes to the imported project itself (e.g., the `zlib/vtkzlib` directory for zlib), should go through the `update.sh` framework. This framework ensures that all patches to the third party projects are tracked externally and available for (preferably) upstream or other projects also embedding the library.

The *Imported Third Party Projects* document lists all projects grouped by import method:

1. `update.sh` framework
2. `git submodule`
3. `copy`

Important: Any updates to projects imported through the `copy` method should first be converted over to the `update.sh` framework.

11.5.1 Updating a Project Upstream

Ideally, any code changes to third party code should first be submitted to the upstream project using whatever workflow they prefer or require. Once that is done, the changes can next be brought into VTK.

11.5.2 Updating the Import

Examine the project's `update.sh` script and note the value of the `repo=` field.

If it's referring to anything other than Kitware's GitLab, then skip to the next section.

Otherwise, you first need to bring in the upstream changes into the `third-party` repo. To do that, first fork and clone the repository named in the `repo=` field. Then use git commands to bring in a copy of the upstream changes.

Here's an example of updating the `twisted` project from tag 17.1.0 to 17.5.0:

```
$ cd twisted/
$ git checkout for/vtk
$ git fetch origin
$ git rebase --onto twisted-17.5.0 twisted-17.1.0
$ git push
```

When deciding what to rebase, you should generally use the first commit in the current history that isn't upstream.

11.5.3 Updating a Project into VTK

Bringing changes into VTK involves first deciding what to bring in. That is specified in the `update.sh` script under the `tag=` field. Usually this is a `for/vtk` branch, but may be `master`, or a tag, or any other Git reference.

If `update.sh` needs to be edited (the usual case), create a branch in the usual way and commit just those changes.

Next, run the `update.sh` script as below. This will update the local copy of the project to the version specified within.

```
$ cd vtk/ThirdParty/zlib
$ git checkout -b update_zlib_YYYY_MM_DD
$ ./update.sh
```

Appending the date to the branch name is not necessary, it just prevents any conflict in the event of you doing this procedure multiple times and inadvertently using the same branch name.

(All this requires a Git 2.5 or higher due the `worktree` tool being used to simplify the availability of the commits to the main checkout.)

Make sure to update the `SPDX_DOWNLOAD_LOCATION` in `CMakeLists.txt` to reflect the changes made to the project.

Now you can review the change and make a merge request from the branch as normal.

11.5.4 Porting a Project

When converting a project, if there are any local patches, a project should be created on Kitware's GitLab to track it (requests may be filed on the repo-requests repository). If the upstream project does not use Git, it should be imported into Git (there may be existing conversions available on Github already). The project's description should indicate where the source repository lives.

Once a mirror of the project is created, a branch named `for/foo` should be created where patches for the `foo` project will be applied (i.e., `for/vtk` for VTK's patches to the project). Usually, changes to the build system, the source code for mangling, the addition of `.gitattributes` files, and other changes belong here. Functional changes should be submitted upstream (but may still be tracked so that they may be used).

For mangling documentation, some guidelines are available.

The basic steps to import a project `twisted` based on the tag `twisted-17.1.0` looks like this:

```
$ git clone https://github.com/twisted/twisted.git
$ cd twisted/
$ git remote add kitware git@gitlab.kitware.com:third-party/twisted.git
$ git push -u kitware
$ git push -u kitware --tags
$ git checkout twisted-17.1.0
$ git checkout -b for/vtk
$ git push --set-upstream kitware for/vtk
```

Making the initial import involves filling out the project's `update.sh` script in its directory. The update-common.sh script describes what is necessary, but in a nutshell, it is basically metadata such as the name of the project and where it goes in the importing project.

The most important bit is the `extract_source` function which should subset the repository. If all that needs to be done is to extract the files given in the `paths` variable (described in the `update-common.sh` script), the `git_archive` function may be used if the `git archive` tool generates a suitable subset.

Make sure `update.sh` is executable before commit. On Unix, run:

```
$ chmod u+x update.sh && git add -u update.sh
```

On Windows, run:

```
$ git update-index --chmod=+x update.sh
```

Also add an entry to *Imported Third Party Projects* for the project, and `CMakeLists.txt` and `module.cmake` as appropriate.

11.5.5 Process

The basic process involves a second branch where the third party project's changes are tracked. This branch has a commit for each time it has been updated and is stripped to only contain the relevant parts (no unit tests, documentation, etc.). This branch is then merged into the main branch as a subdirectory using the `subtree` merge strategy.

Initial conversions will require a manual push by the maintainers since the conversion involves a root commit which is not allowed under normal circumstances. Please post a message on the VTK Discourse forum asking for assistance if necessary.

11.6 Imported Third Party Projects

This page provides an overview of the imported third-party projects that VTK depends on, grouped by import method.

The lists below references project directory name found in either the `ThirdParty` or `Utilities` source sub-directory available in the VTK GitLab repository where additional details may be found.

11.6.1 Using the `update.sh` framework

The following list shows third-party projects that were imported using the `update.sh` framework described in the *Updating Third Party Projects* document:

- cgns
- cli11
- diy2
- doubleconversion
- eigen
- exodusII
- expat
- exprtk
- fast_float
- fides
- fmt
- freetype
- gl2ps
- glew

- h5part
- hdf5
- ioss
- jpeg
- jsoncpp
- kissfft
- KWIML
- KWSys
- libharu
- libproj
- libxml2
- loguru
- lz4
- lzma
- MetaIO
- mpi4py
- netcdf
- nlohmannjson
- ogg
- pegtl
- png
- pugixml
- sqlite
- theora
- tiff
- utf8
- verdict
- xdmf3
- zfp
- zlib

11.6.2 Using `git submodule`

The following third-party project were imported as git submodules:

- vtkm

11.6.3 Using `copy`

The following third-party projects were imported by copying the files:

- vpic
- xdmf2

11.7 Deprecation Process

This page documents how to deprecate an API and mark it as no longer necessary for downstream consumers of VTK.

11.7.1 Deprecating classes and methods

Classes, functions, and methods may be deprecated using the deprecation macros.

```
#include "vtkDeprecation.h" // Include the macros.

// A deprecated class.
VTK_DEPRECATED_IN_X_Y_Z("reason for deprecation")
class oldClass {
public:
  // A deprecated method.
  VTK_DEPRECATED_IN_X_Y_Z("reason for deprecation")
  void oldMethod();
};

// A deprecated function.
VTK_DEPRECATED_IN_X_Y_Z("reason for deprecation")
void oldFunction();
```

The `X_Y_Z` should be the newest macro available in the `vtkDeprecation.h` header when the API is added.

Note that, unlike, the old `VTK_LEGACY_REMOVE` mechanism, the APIs are not deleted. This does interfere with various kinds of deprecations.

- *Changing the return type*: Don't do this. Use a new name for the function/method.
- *Deprecating macros*: Use `VTK_LEGACY_REMOVE`. New macro APIs should be highly discouraged.

Lifetime of deprecated APIs

Deprecated APIs should exist for at least one release with the deprecation warning active. This gives consumers of VTK at least one cycle to notice the deprecation and move off of it.

Upon branching for a release, `master` will soon after have all instances of deprecated symbols removed.

Avoiding warnings within VTK

VTK is providing the deprecated symbols and as such may still use them in tests or implementations. Since these generate warnings when compiling VTK itself, classes which define deprecated symbols must suppress them.

Sources which continue to use the deprecated macros should add a comment to the top of the source file to hide deprecation warnings in CI.

```
// Hide VTK_DEPRECATED_IN_X_Y_Z() warnings for this class.
#define VTK_DEPRECATION_LEVEL 0
```

If one already exists, please add another comment to it so that when deprecated symbols are removed, it shows up in the search.

11.7.2 Using `VTK_DEPRECATION_LEVEL`

When using VTK, the `VTK_DEPRECATION_LEVEL` macro may be set to a version number. APIs which have been deprecated after this point will not fire (as the API is not deprecated as of the level requested). It should be defined using the `VTK_VERSION_CHECK(major, minor, patch)` macro.

Note that APIs on the verge of deletion (those deprecated in at least one release) will always raise deprecation warnings.

If not set, its value defaults to the current level of VTK.

11.8 Release Process

This document provides a high-level overview of the VTK release cycle and associated release process.

11.8.1 Overview

We aim to release a new version of VTK every six months. However, we recognize that this schedule is flexible. The project is funded and developed by many different groups, each of which works towards their own particular sets of features.

VTK releases are named with a `Major.Minor.Patch` scheme.

11.8.2 Branching Scheme

The overall release history resembles a skinny tree. Development proceeds along the `master` branch, consisting of topic branches that start from and are merged into `master`. Every so often, a release is tagged and branched from it.

In general, no work takes place on the `release` branch, other than the handful of important patches that make up occasional patch releases.

Hint: Steps for contributing changes specific to the `release` branch are documented in *Create a Topic*.

On the `master` branch, bug fixes and new features are continuously developed. At release time, the focus temporarily shifts to producing a library that is as stable and robust as possible.

11.8.3 Steps

The process for cutting releases is as follows:

1. Announce upcoming release

   A few weeks before the intended `release` branch, announce on VTK Discourse that a new release is coming. This alerts developers to avoid making drastic changes that might delay the release and gives them a chance to push important and nearly completed features in time for the release. For example, see this post.

2. Polish the dashboards and bug tracker by addressing outstanding issue and coordinate effort with relevant developers.

   Persistent compilation and regression test problems are fixed. Serious outstanding bugs are fixed.

3. Create a new issue titled `Release X.Y.Z[rcN]` based of the new-release template.

   Important: Specific steps to create eiter the candidate or the official release are found in the newly created issue.

4. Perform the release candidate cycle

   1. Tag the release branch and create and publish release candidate artifacts and change summaries.
   2. Announce the release candidate and request feedback from the community, especially third-party packagers.

      Hint: Bug reports should be entered into the bug tracker with the upcoming release number as the milestone.

   3. If the community reports bugs, classify them in the bug tracker and ensure they are fixed.

      Only serious bugs and regressions need to be fixed before the release. New features and minor problems should be merged into `master` as usual.

      Patches for the release branch should start from the release branch, be submitted through GitLab, and then merged into `master`. Once fully tested there, the branch can be merged into the release branch.

      When the selected issues are fixed in the release branch, tag the tip of the release branch and release it as the next candidate, then the cycle continues.

   4. Distribution specific patches can accumulate over time. Consider reviewing the following distribution specific pages to identify potential fixes and improvements that could be integrated in VTK itself:

- Debian:
  - https://tracker.debian.org/pkg/vtk9
  - https://udd.debian.org/patches.cgi?src=vtk9
- Gentoo:
  - https://packages.gentoo.org/packages/sci-libs/vtk
  - https://gitweb.gentoo.org/repo/gentoo.git/tree/sci-libs/vtk/files
- openSUSE:
  - https://build.opensuse.org/package/show/openSUSE:Factory/vtk

5. Package the official release

   The official VTK package consists of tar balls and ZIP files of the source, Python Wheels, Doxygen documentation, and regression test data, all at the tag point.

   Volunteer third-party packagers create binary packages from the official release for various platforms, so their input is especially valuable during the release cycle.

   The release manager also compiles release notes for the official release announcement. Release notes are compiled from various standardized topic documents added to the `Documentation/release/dev` folder while features or issues are fixed. The aggregation of these topic files is done manually and results in the creation of a file named `Documentation/release/X.Y.md` for the current release.

11.8.4 GitLab and Releases

GitLab milestones are used for keeping track of branches for the release. They allow keeping track of issues and merge requests which should be "done" for the milestone to be considered complete.

For each release (including release candidates), a milestone is created with a plausible due date. The milestone page allows for an easy overview of branches which need wrangling for a release.

Merge Requests

Merge requests which need to be rebased onto the relevant release branch should be marked with the `needs-rebase-for-release` tag and commented on how the branch can be rebased properly:

~~~
This branch is marked for a release, but includes other commits in
`master`. Please either rebase the branch on top of the release branch and
remove the `needs-rebase-for-release` tag from the merge request:

```sh
$ git rebase --onto=origin/release origin/master $branch_name
$ git gitlab-push -f
```

or, if there are conflicts when using a single branch, open a new branch
and open a merge request against the `release` branch:

```sh
$ git checkout -b ${branch_name}-release $branch_name
$ git rebase --onto=origin/release origin/master ${branch_name}-release
$ git gitlab-push
~~~

(continues on next page)

(continued from previous page)

~~~
```

Thanks!
~~~

Wrangling Branches

Branches may be wrangled using the filters in the merge request page. Replace `$release` at the end with the relevant milestone name:

```
https://gitlab.kitware.com/vtk/vtk/-/merge_requests?state=all&milestone_title=$release
```

The following states of a merge request indicate where they are in the flow:

- open for `master`: get into `master` first
- open for `release`: ensure it is already in `master`
- open with `needs-rebase-for-release` tag: wait for contributor to rebase properly; ping if necessary
- `MERGED`: merge into `release`

There is currently no good way of marking a branch that went towards `master` is also in `release` already since tags cannot be added to closed merge requests. Suggestions welcome :) .

11.9 Coding Conventions

11.9.1 General

VTK is a large body of code with many users and developers. Coding in a consistent style eases shared development. VTK's style guidelines also ensure wide portability. All code that is contributed to VTK must conform to the following style guidelines. Exceptions are permissible, following discussion in code review, as long as the result passes the nightly regression tests. External code contributed into the ThirdParty directory is exempt from most of the following rules except for the rules that say "All code".

1. All code that is compiled into VTK by default must be compatible with VTK's BSD- style license.
2. Copyright notices should appear at the top of C++ header and implementation files using SPDX syntax.
3. All C++ code must be valid C++11 code.
4. The Java and Python wrappers must work on new code, or it should be excluded from wrapping.
5. Multiple inheritance is not allowed in VTK classes.

   Rationale: One important reason is that Java does not support it.
6. Only one public class per header file. Internal helper classes may be forward declared in header files, but can then only be defined in implementation files, ie using the PIMPL idiom.

   Rationale: helpful when searching the code and limits header inclusion bloat that slows compilation time.
7. Class names and file names must match, class names must be unique.

   Rationale: helpful when searching the code, includes are flattened at install.
8. The indentation style can be characterized as the modified Allman (https://en.wikipedia.org/wiki/Indent_style#Allman_style)style. Indentations are two spaces, and the curly brace (scope delimiter) is placed on the following line and indented to the same level as the control statement.

Rationale: Readability and historical

9. Conditional clauses (including loop conditionals such as for and while) must be in braces below the conditional. Ie, instead of `if (test) clause` or `if (test) { clause }`, use

```
if (test)
{
  clause
}
```

Rationale: helpful when running code through a debugger

10. Two space indentation. Tabs are not allowed. Trailing whitespace is not allowed.

Rationale: Removing tabs ensures that blocks are indented consistently in all editors.

11. Only alphanumeric characters in names. Use capitalization to demarcate words within a name (i.e., camel case). Preprocessor variables are the exception, and should be in all caps with a single underscore to demarcate words.

Rationale: Readability

12. Every class, macro, etc starts with either vtk or VTK. Classes should all start with lowercase vtk and macros or constants can start with either.

Rationale: avoids name clashes with other libraries

13. After the `vtk` prefix, capitalize the first letter of class names, methods and static and instance variables. Local variables are allowed to vary, but ideally should start in lower case and then proceed in camel case.

Rationale: Readability

14. Try to always spell out a name and not use abbreviations except in cases where the shortened form is obvious and widely understood.

Rationale: Readability, self-documentation

15. Classes that derive from `vtkObject` should have protected constructors and destructors, and privately declared but unimplemented copy constructor and assignment operator.

    1. Classes that don't derive from `vtkObject` should obey the rule of three. If the class implements the destructor, copy constructor or copy assignment operator they should implement all of them.

Rationale: VTK's reference counting implementation depends on carefully controlling each object's lifetime.

16. Following the copyright notice, the name and purpose of each class should be documented at the top of the header with standard doxygen markup.:

```
/**
 * @class vtkclassname
 * @brief one line description
 *
 * Longer description of class here.
*/
```

Rationale: Doxygen generated documentation uses this to describe each class.

17. Public methods must be documented with doxygen markup.

```
/**
 * Explanation of what the method/ivar is for
 */
```

Descriptions should do more than simply restate the method or ivar's name.

The documentation for each public ivar should document the default value.

The documentation style for SetGet macros should be a single comment for the pair and a brief description of the variable that is being set/get. Use doxygen group marking to make the comment apply to both macro expanded functions.

```
///@{
/**
 * Set / get the sharpness of decay of the splats.
 * This is the exponent constant in the Gaussian
 * equation. Normally this is a negative value.
 */
 */
vtkSetMacro(ExponentFactor,double);
vtkGetMacro(ExponentFactor,double);
///@}
```

The documentation style for vector macros is to name each of the resulting variables. For example comment

```
/**
 * Set/Get the color which is used to draw shapes in the image. The parameters are␣
↪SetDrawColor(red, green, blue, alpha)
 */
vtkSetVector4Macro(DrawColor, double);
vtkGetVector4Macro(DrawColor, double);
```

The description for SetClamp macros must describe the valid range of values.

```
/**
 * Should the data with value 0 be ignored? Valid range (0, 1).
 */
vtkSetClampMacro(IgnoreZero, int, 0, 1);
vtkGetMacro(IgnoreZero, int);
```

Rationale: Doxygen generated documentation (http://www.vtk.org/doc/nightly/html/) is generated from these comments and should be consistently readable.

18. Public and even Protected instance variables are allowed only in exceptional situations. Private variables should be used instead with public access given via Set/Get macro methods when needed. Rationale: Consistent API, ease of deprecation, and SetMacro takes part in reference counting.

19. Protected methods are allowed only when they are intended to be used by inheriting classes and overridden by inheriting classes. Private methods should be the default for any method. Please note this is not true in many classes but should be followed when adding new code. Rationale: Consistent API, ease of deprecation.

20. Accessors to `vtkObject` instance variables should be declared in the header file, and defined in the implementation file with the vtkCxxSetObjectMacro. Rationale: Reduces header file bloat and assists in reference counting.

21. Use `this->` inside of methods to access class methods and instance variables. Rationale: Readability as it helps to distinguish local variables from instance variables.

22. Header files should normally have just two includes, one for the superclass' header file and one for the class' module export header declaration. It is required that all but the superclass header have a comment explaining why the extra includes are necessary. Care should be taken to minimize the number of includes in public headers, with predeclaration/PIMPL preferred. Rationale: limits header inclusion bloat that slows compilation time.

23. Include statements in implementation files should generally be in alphabetical order, grouped by type. For example, VTK includes first, system includes, STL includes, and Qt includes. Rationale: avoid redundant includes, and keep a logical order.
24. All subclasses of `vtkObject` should include a `PrintSelf()` method that prints all publicly accessible ivars.

    Rationale: useful in debugging and in wrapped languages that lack sufficient introspection.
25. All subclasses of `vtkObject` should include a type macro in their class declaration.

    Rationale: VTK's implementation of runtime type information depends on it
26. Do not use `id` as a variable name in public headers, also avoid `min`, `max`, and other symbols that conflict with the Windows API.

    Rationale: `id` is a reserved word in Objective-C++, and against variable name rules. `min`, `max`, and less common identifiers listed in Testing/Core/WindowsMangleList.py are declared in the Windows API.
27. Prefer the use of vtkNew when the variable would be classically treated as a stack variable.
28. Eighty character line width is preferred.

    Rationale: Readability
29. Method definitions in implementation files should be preceded by // followed by 78 `-` characters.

    Rationale: Readability
30. New code must include regression tests that will run on the dashboards. The name of the file to test vtkClassName should be TestClassName.cxx. Each test should call several functions, each as short as possible, to exercise a specific functionality of the class. The `main()` function of the test file must be called TestClassName(int, char*[])

    Rationale: Code that is not tested can not be said to be working.
31. All code must compile and run without warning or error messages on the nightly dashboards, which include Windows, Mac, Linux and Unix machines. Exceptions can be made, for example to exclude warnings from ThirdParty libraries, by adding exceptions to CMake/CTestCustom.cmake.in
32. Namespaces should not be brought into global scope in any public headers, i.e. the `using` keyword should not appear in any public headers except within class scope. It can be used in implementations, but it is preferred to bring symbols into the global scope rather than an entire namespace.

    Rationale: Using VTK API should not have side-effects where parts of the std namespace (or the entire thing) are suddenly moved to global scope.
33. While much of the legacy VTK API uses integers for boolean values, new interfaces should prefer the bool type.

    Rationale: Readability.
34. Template classes are permitted, but must be excluded from wrapped languages.

    Rationale: The concept of templates doesn't exist in all wrapped languages.

11.9.2 Specific C++ Language Guidelines

C++ Standard Library

- Do not use vtkStdString in new API; prefer std::string

  Rationale: vtkStdString was introduced as a workaround for compilers that couldn't handle the long symbol name for the expanded std::string type. It is no longer needed on modern platforms.

- STL usage in the Common modules' public API is discouraged when possible, Common modules are free to use STL in implementation files. The other modules may use STL, but should do so only when necessary if there is not an appropriate VTK class. Care should be taken when using the STL in public API, especially in the context of what can be wrapped.

  Exception: std::string should be used as the container for all 8-bit character data, and is permitted throughout VTK.

  Rationale: limits header inclusion bloat, wrappers are not capable of handling many non-`vtkObject` derived classes.

- References to STL derived classes in header files should be private. If the class is not intended to be subclassed it is safe to put the references in the protected section.

  Rationale: avoids DLL boundary issues.

C++ Language Features Required when using VTK

- *nullptr* Use `nullptr` instead of `0` and `NULL` when dealing with pointer types
- *override* `VTK_OVERRIDE` will be replaced with the override keyword
- *final* `VTK_FINAL` will be replaced with the final keyword
- *delete* The use of delete is preferred over making default members private and unimplemented.

C++11 Features allowed throughout VTK

- *default* The use of default is encouraged in preference to empty destructor implementations
- *static_assert* Must use the static_assert (`bool_constexpr` , `message`) signature. The signature without the message in c++17
- *non static data member initializers*
- *strongly typed enums* VTK prefers the usage of strongly typed enums over classic weakly typed enums.

  Weakly typed enums conversion to integers is undesirable, and the ability for strongly typed enums to specify explicit storage size make it the preferred form of enums.

  strongly typed: `enum class Color { red, blue };`

  weakly typed: `enum Color { red, blue };`

  While VTK is aware that conversion of all enums over to strongly typed enums will uncover a collection of subtle faults and incorrect assumptions. Converting existing classes to use strongly typed enums will need to be investigated and discussed with the mailing list, as this will break API/ABI, potentially cause issues with VTK bindings, and possibly require changes to users VTK code.

C++11 Features acceptable in VTK implementation files, private headers, and template implementations

- *auto* Use auto to avoid type names that are noisy, obvious, or unimportant - cases where the type doesn't aid in clarity for the reader. auto is permitted when it increases readability, particularly as described below. **Never initialize an auto-typed variable with a braced initializer list.**

  Specific cases where auto is allowed or encouraged:

  - (Encouraged) For iterators and other long/convoluted type names, particularly when the type is clear from context (calls to find, begin, or end for instance).
  - (Allowed) When the type is clear from local context (in the same expression or within a few lines). Initialization of a pointer or smart pointer with calls to new commonly falls into this category, as does use of auto in a range-based loop over a container whose type is spelled out nearby.
  - (Allowed) When the type doesn't matter because it isn't being used for anything other than equality comparison.
  - (Encouraged) When iterating over a map with a range-based loop (because it is often assumed that the correct type is std::pair<KeyType, ValueType> whereas it is actually std::pair<const KeyType, ValueType>). This is particularly well paired with local key and value aliases for .first and .second (often const-ref).

  -
    ```
    for (const auto& item : some_map) {
    const KeyType& key = item.first;
    const ValType& value = item.second;
    // The rest of the loop can now just refer to key and value,
    // a reader can see the types in question, and we've avoided
    // the too-common case of extra copies in this iteration.
    }
    ```
  - (Discouraged) When iterating in integer space. `for (auto i=0; i < grid->GetNumberOfPoints(); ++i)`. Because vtk data structures usually contain more than 2 billion elements, iterating using 32bit integer is discouraged (and often doesn't match the type used)

- *braced initializer list* Braced initializer list are allowed as they prevent implicit narrowing conversions, and "most vexing parse" errors. They can be used when constructing POD's and other containers.

  Braced initializer lists are not allowed to be used as the right hand side for auto:

  ```
  auto a = { 10, 20 }; //not allowed as a is std::initializer_list<int>
  ```

- *lambda expressions*

  Usage of lambda expressions are allowed with the following guidelines.

  - Use default capture by value ([=]) only as a means of binding a few variables for a short lambda, where the set of captured variables is obvious at a glance. Prefer not to write long or complex lambdas with default capture by value.
  - Except for the above, all capture arguments must be explicitly captured. Using the default capture by reference ([&]) is not allowed. This is to done so that it is easier to evaluate lifespan and reference ownership.
  - Keep unnamed lambdas short. If a lambda body is more than maybe five lines long, prefer using a named function instead of a lambda.
  - Specify the return type of the lambda explicitly if that will make it more obvious to readers.

- *shared_ptr*

  - **Do not combine shared_ptr and vtk derived objects.** VTK internal reference counting makes the shared_ptr reference counting (and destructor tracking) pointless.

- *unique_ptr*
  - Do not combine unique_ptr and vtk derived objects. We prefer using vtkNew as VTK objects use internal reference counting and custom deletion logic, the ownership semantics of unique_ptr are invalid.
  - `make_unique` is not part of c++11
- *template alias*
  - The use of alias templates is preferred over using 'typedefs'. They provide the same language pattern of normal declarations, and reduce the need for helper template structs. For example (Scott Meyers, Effective Modern C++)

```
template<typename T> using MyAllocList = std::list<T, MyAlloc<T>>;
```

- universal references (&&) / std::move / std::forward
- *extern templates*
  - Note: This should be investigated as an update to the current infrastructure used to export explicit template instantiations used within VTK
- *unordered maps*
- *std::array*
  - The use of std::array is preferred over using raw fixed sized arrays. They offer compile time bounds checking without any runtime cost.
- *range based for loop*

C++11 Features allowed under certain conditions

- *concurrency*

  Concurrency inside of vtk should be handled by using or extending the already existing collection of support classes like vtkAtomic and vtkSMPThreadLocal.

  Instead of directly using new c++11 constructs such as std::compare_exchange_weak instead extend the functionality of vtk core concurrency classes.

  Note: Thread local storage has not been supported on OSX previously to XCode 8. VTK offers the following classes that should be used instead:

  - vtkSMPThreadLocalObject
  - vtkSMPThreadLocal
- *std::isnan*, *std::isfinite*, *std::isinf*

  These functions should not be called directly, instead the wrapped versions provided by vtk should be used instead.

  - vtk::isnan -> std::isnan
  - vtk::isfinite -> std::isfinite
  - vtk::isisinfnan -> std::isinf

The reason for these wrappings is to work around compiler performance issues. For example, some clang version would convert integral types to double and do the operation on the double value, instead of simply returning false/true.

- *std::future*/ *std::async*

  Future/Async based programming inside of vtk should be handled on a case by case basis. In general the use cases for this kind of execution model is best applied at the vtkExecutive / vtkPipeline level, or at the File IO level.

  In these cases the recommendation is to extending or adding support classes so that these design patterns can be utilized in the future.

- *variadic templates*

  Variadic Templates are not allowed in VTK unless they are the only solution to the given problem.

C++11 Features that are not allowed

- std::regex
  - Not supported by GCC 4.8 (can be used once GCC 4.9 is required)
- constexpr
  - Not supported by VS2013
- unicode string literals (n2442)
  - Not supported by VS2013
- universal character names in literals (n2170)
  - Not supported by VS2013
- user-defined literals (n2765)
  - Not supported by VS2013
- Extended sizeof (n2253)
  - Not supported by VS2013
- Unrestricted Unions (n2544)
  - Not supported by VS2013
- Noexcept (n3050)
  - Not supported by VS2013

Parts of this coding style are enforced by git commit hooks that are put in place when the developer runs the Setup-ForDevelopment script, other parts are enforced by smoke tests that run as part of VTK's regression test suite. Most of these guidelines are not automatically enforced. *VTK's commit hook enforced style checks Section* list the style checks that are in place.

VTK's commit hook enforced style checks

- Well formed commit message

  Every commit message should consist of a one line summary optionally followed by a blank line and further details. This is most easily approximated to the subject of an email, and the body in the form of paragraphs.

- Valid committer username and email address Every developer must have a valid name and email configured in git.\
- ASCII filename check All file names must contain only ASCII characters.
- No tabs
- No trailing whitespace
- No empty line at end of file
- Proper file access mode

  Files must be committed with sensible access modes.

- One megabyte maximum file size
- No submodules

  The VTK project does not allow submodules. For required third party dependencies, the recommended scheme is to use git's subtree merge strategy to reproducibly import code and thereby simplify eventual integration of upstream changes.

Additionally, new developers should be aware that the regression test machines have fairly strict compiler warnings enabled and usually have VTK_DEBUG_LEAKS configured on to catch leaks of VTK objects. Developers should be in the habit of doing the same in their own environments so as to avoid pushing code that the dashboards will immediately object to. With GCC, it is easiest to do so by turning on VTK_EXTRA_COMPILER_WARNINGS.

11.10 About this documentation

This website is hosted on readthedocs.io. It is generated using Sphinx together with the MyST parser. The Python API is extracted using autodoc2 extension while cmake API uses moderncmake-domain. The complete configuration along with custom helpers for auto-generating some of the content can be found here.

11.11 Quick Start Guide

This is a quick start guide so that you can start contributing to VTK easily. To understand the process more deeply, you can jump to the *workflow* section.

11.11.1 Initial Setup

Before you begin, perform your initial setup using the following steps:

1. Register GitLab Access to create an account and select a user name.
2. Fork VTK into your user's namespace on GitLab.
3. Follow the download instructions to create a local clone of the main VTK repository:

   ```
   $ git clone --recursive https://gitlab.kitware.com/vtk/vtk.git VTK
   ```

   The main repository will be configured as your `origin` remote.

4. Run the developer setup script to prepare your VTK work tree and create Git command aliases used below:

   ```
   $ ./Utilities/SetupForDevelopment.sh
   ```

   This will prompt you for your GitLab username and configure a remote called `gitlab` to refer to your fork. It will also setup a data directory for you. No need to do anything else.

11.11.2 Development

Create a local branch for your changes:

```
git checkout -b your_branch
```

Make the needed changes in VTK and use git locally to create logically separated commits. There is no strict requirements regarding git commit messages syntax but a good rule of thumb to follow is: `General domain:  reason for change`, General domain being a class, a module , a specific system like build or CI.

```
git commit -m "General domain: Short yet informative reason for the change"
```

Build VTK following the guide and fix any build warnings or issues that arise and seems related to your changes.

Add/Improve tests in order to ensure your changes are tested. Take a look in the `Testing` directory of the module you are making changes in to see how the tests are currently built and try to follow the same paradigms. Run your test locally from your build directory and check that they pass:

```
cmake . && cmake --build .
ctest -VV -R yourTest
```

11.11.3 Upload

Push your changes to the GitLab fork that you created in the *initial setup* stage:

```
git push gitlab
```

11.11.4 Data

If your test uses new data or baselines, you will need to add it to your fork. For data, add the file names to the list in your module `yourModule/Testing/CMakeLists.txt` and drop the files in `Testing/Data/`. For baselines, just drop the file in `yourModule/Testing/Data/Baselines` and run the following commands from your build directory:

```
cmake . && cmake --build .
```

This will transform your files into .sha512 files. Check your test is passing by running from your build directory:

```
ctest -VV -R yourTest
```

If it passes, add these .sha512 files and commit them, then push with:

```
git gitlab-push
```

11.11.5 Create a Merge Request

Once you are happy with the state of your development on your fork, the next step is to create a merge request back into the main VTK repository.

Open in a browser, select your branch in the list and create a Merge Request against master.

In the description, write an informative explanation of your added features or bugfix. If there is an associated issue, link it with the `#number` in the description.

Tag some VTK maintainers in the description to ensure someone will see it, see here for the complete *list*.

11.11.6 Robot Checks

Once the MR is created, our GitLab robot will check multiple things and make automated suggestions. Please read them and try to follow the instructions. The two standard suggestions are related to formatting errors and adding markdown changelog.

To fix the formatting, just add a comment containing:

```
Do: reformat
```

Then, once the robot has fixed the formatting, fetch the changes locally (this will remove any local changes to your branch)

```
git fetch gitlab
git reset --hard gitlab/your_branch
```

To fix the changelog warning, create, add, commit and push a markdown (.md) file in `Documentation/release/dev` folder. In this file, write a small markdown paragraph describing the development. See other .md files in this folder for examples. It may look like this:

```
## Development title

A new feature that does this and that has been introduced.
This specific issue has been fixed in this particular way.
```

Suggestions and best practices on writing the changelog can be found in the `Documentation/release/dev/0-sample-topic.md` file. This is an optional step but recommended to do for any new feature and user facing issues.

11.11.7 Reviews

VTK maintainers and developers will review your MR by leaving comments on it. Try to follow their instructions and be patient. It can take a while to get a MR into mergeable form. This is a mandatory step, and it is absolutely normal to get change requests.

Review comments can be resolved, please resolve a comment once you've taken it into account and pushed related changes or once you've reached an agreement with the commenter that nothing should be changed.

Once a reviewer is happy with your changes, they will add a `+X` comment. You need at least one `+2` or higher to consider merging the MR. Two `+1`s do not equal a `+2`. If a reviewer leave a `-1` comment, please discuss with them to understand what is the issue and how it could be fixed.

Once you have pushed new changes, please tag reviewers again so that they can take a look. If you do not tag reviewers, they may not know to revisit your changes. *Do not hesitate to tag them and ask for help*.

11.11.8 Continuous Integration

Before merging a MR, the VTK continuous integration (CI) needs to run and be green. For CI to be functional, please read and follow this guide.

To run the CI:

- Click on the Pipelines Tab
- Click on the last pipeline status badge
- Press the `Play all manual` arrows on top of the Build and Test stages

Do not hesitate to tag a VTK developer for help if needed.

You then need to wait for CI to run, it can take a while, up to a full day.

A successful CI should be fully green. If that is so, then your MR is ready !

If not, you need to analyse the issues and fix them. Recover the failure information this way:

Click on the pipelines tab, then on the last status badge, then on the `cdash-commit` job. It will take you to the related CDash report where you will find all information.

Everything in the CDash report should be green except the `NotRun` and `Time` column. Take a look into each issue and fix them locally. If there are issues in the pipeline but nothing is visible in the CDash, please ask a maintainer for help to figure out if anything should be done. You can always try to rerun the failed job by clicking on the arrow of the job in the pipeline.

Once you have fixed some issues locally, commit and push them to gitlab, run the CI again and tag reviewers again for follow-up reviews.

11.11.9 Merging

Once the MR has green CI and you have at least one `+2`, you can ask for a merge. Before that please make sure that:

- Your commit history is logical (or squashed into a single commit) and cleaned up with good commit messages
- You are rebased on a fairly recent version of master

If that is not the case, please rebase on master using the following commands:

```
git fetch origin
git rebase -i origin/master
git push gitlab -f
```

The interactive rebase will let you squash commits, reorganize commits and edit commit messages.

After the force push, make sure to run CI again.

Once all is done, tag a VTK developer so that they can perform the merge command.

Congratulations ! You just contributed to VTK !

CHAPTER
TWELVE

RESOURCES

For commercial or confidential consulting related to VTK or any of our other products and services, please contact Kitware's advanced support team for personalized assistance.

12.1 Links

Name	Description	
Book	Descriptions of important visualization algorithms, including example images and code that utilizes VTK	book.vtk.org
Discourse	Community forum	discourse.vtk.org
GitLab	Merge requests and issues take place here	gitlab.kitware.com/vtk/vtk
Examples	Examples, Tutorials, and guides for VTK in C++ and Python	examples.vtk.org
Doxygen	Documentation of VTK C++ classes updated daily	vtk.org/doc/nightly/html
CDash	Quality Dashboard	open.cdash.org/index.php?project=VTK

12.2 Python

Name	Description	
PyPI	Python Wheels	`pip install vtk`
`wheels.vtk.org`	See *Additional Python Wheels*	`pip install --extra-index-url https://wheels.vtk.org vtk`

12.3 Docker

The VTK Docker Repositories are a set of ready-to-run Docker images aiming to support development and testing of VTK-based projects.

Repository	Description	Dockerfile
kitware/vtk	Images with built dependencies to support the continuous integration of VTK	
kitware/vtk-for-ci	Images with installation of VTK (in `/opt/vtk/install`) to support building & testing your VTK-based projects. Learn more reading this blog.	
kitware/vtk-wasm	Static emscripten build of VTK to support building VTK-based WebAssembly applications. See Using WebAssembly	
kitware/vtkm	Images with built dependencies to support the continuous integration of VTK-m.	

CHAPTER THIRTEEN

RELEASE DETAILS

13.1 9.3

Released on 2023-11-09.

13.1.1 9.3.0 Release Notes

Changes made since VTK 9.2.0 include the following.

Changes

Build

- Compile fixes for C++20 builds with gcc11.
- Apply /utf-8 option for MSVC builds for standardization.
- Headers `vtkBlockSortHelper.h` from `VTK::RenderingVolume` and `vtkDIYKdTreeUtilities.h` from `VTK::FiltersParallelDIY2` are now installed.
- The `vtk-config.cmake` CMake package no longer permits unknown components to be listed and will report them as not found. This helps ensure the usability `VTK::Component` when `VTK_Component_FOUND` is set.
- The `vtk_encode_string` CMake API now supports the `ABI_MANGLE_SYMBOL_BEGIN`, `ABI_MANGLE_SYMBOL_END`, and `ABI_MANGLE_HEADER` arguments to specify a mangling mechanism. Previously (where mangling was supported), it was hard-coded to VTK's own mangling decisions.

Charts

- Uniformize the `vtkPlot` API for color setters/getter, in order to fit the API of `vtkPen` and `vtkBrush`. Methods using floating point parameters (e.g. `vtkPlot::SetColor(double r, double g, double b)`) are now suffixed with `F` to avoid confusion with equivalent functions using unsigned chars. The former ones are marked as deprecated.
- `vtkChartParallelCoordinates`'s default selection behavior has been simplified. Multiple selection is no longer supported in `SELECTION_DEFAULT`.

Copyright

- SPDX information have been added and replace all previous copyright declaration in all of VTK. See more information on the process used.

Core

- OSPRay has been disabled for older x86_64 processors which do not support SSE4.1.
- Removed hidden private dependency of `CommonCore` on `CommonDataModel`.
- Nested parallelism has been disabled by default for all backends except TBB, which should improve performance. Enabling nested parallelism is still possible when sub-task are coarse enough, using the `SetNestedParallelism` method or a `LocalScope`.
- Improved `vtkSMPTools` STDThread backend. A common, global, thread pool is now shared between all SMP calls, so they no longer create threads.

Data

- `vtkPolyLine::Clip` improved to generate polylines whenever possible.
- `vtkCompositeDataSet::ShallowCopy` now does an actual shallow copy up to array pointers.
- Fixed calculation of `vtkPyramid` centroid.

Filters

- VTK's interruption method has been updated to use `CheckAbort`. `CheckAbort` will check the current filter's `AbortExecute` flag as well as any upstream filter's `AbortExecute` flag. If any are set, the filter will output empty data and tell downstream filters to abort as well. Currently, `vtkContourGrid`, `vtkClipDataSet`, `vtkShrinkFilter`, and `vtkRTAnalyticSource`.
- `vtkmContour`'s `ComputeScalars` parameter has been fixed to behave like `vtkContourFilter`.
- `vtkExtractCells` has been relocated from `Filters/Extraction` to `Filters/Core`.
- `vtkTemporalDataSetCache` now deep copies data by default.

Geovis

- Moved `vtkCompassWidget` and `vtkCompassRepresentation` from `Geovis/Core` to `Interaction/Widgets`.

Interaction

- `vtkSelectPolyData` now passes cell data attributes to the selected and unselected outputs.
- Output support fixed for `vtkSelectPolyData` when `GenerateSelectionScalars` is enabled.

I/O

- `vtkPLYReader` changed to use new `vtkResourceStream` IO.
  - `vtkPLY::get_ascii_item` signature changed from `void(const char*, int, int*, unsigned int*, double*)` to `void(vtkResourceParser*, int, int*, unsigned int*, double*)`
  - `vtkPLY::ply_read` signature changed from `PlyFile*(std::istream*, int*, char***)` to `PlyFile*(vtkResourceStream*, int*, char***)`
  - `vtkPLY::get_words` signature changed from `void(std::istream* is, std::vector<char*>* words, char line_words[], char orig_line[])` to `void(vtkResourceParser* is, std::vector<char*>* words, char line_words[], char orig_line[])`
- `vtkPIOReader::GetTimeDataArray` now returns `nullptr` when the index is out-of-range.
- CGNS fixed compilation with HDF5 1.12.
- `vtkSTLReader` fixed to not consume new lines erroneously.

Python

- OSMesa VTK wheels are now provided. These are available on VTK's official channels (the VTK repository's Python index and `vtk.org`), but not PyPI because OSMesa conflicts with other OpenGL packages.
- The numpy adapter (`util.numpy_support`) converts `numpy.int8` arrays to `vtkSignedCharArray` rather than `vtkCharArray`, to ensure that signedness is preserved by the conversion.

Rendering

- Fix wireframe render shading issues for some GPUs.
- VTK previously exported a lot of its shader strings from its libraries. Now only those that are available through installed headers are available. These include:
  - `vtkTextureObjectVS` from `VTK::RenderingOpenGL2`
  - `vtkCompositeZPassFS` from `VTK::RenderingParallel`
- Volume label mapping improved to properly index upto 256 labels each with their own color, opacity and gradient transfer functions.

System

- `vtkExecutableRunner`'s argument splitting system has been overhauled. There are now 2 modes to execute a command using the `ExecuteInSystemShell` flag:
  - When `ExecuteInSystemShell` is `true` (default), the class will execute the given command in the system shell, leaving the actual argument split to the shell.
  - When `ExecuteInSystemShell` is `false`, you will have to split the command and its arguments yourself using the new `AddArgument` API.

Third Party

- VTK's vendored `zlib` library has been updated to 1.2.13.
- VTK's vendored `fmt` library has been updated to 9.1.0.
- VTK's vendored `ioss` library has been updated to the 2022-10-14 release.
- VTK's vendored `libtiff` library has been updated to 4.6.0. The new version fixes a number of CVEs.
- VTK's vendored `netcdf` library has been updated to 4.9.2.
- VTK's vendored `mpi4py` library has been updated to 3.1.4.
- VTK's vendored `expat` library has been updated to 2.4.8.
- VTK's vendored `libxml2` library has been updated to 2.10.1.
- VTK's vendored `PDAL` library has been updated to 2.1.
- Added fix for `Proj` compatibility with `windows.h` with the VTK `STRICT` definition.

New Features

ABI Namespace

- VTK is now wrapped in a customizable `inline namespace` (`VTK_ABI_NAMESPACE`). To wrap code in the ABI namespace use `VTK_ABI_NAMESPACE_BEGIN` and `VTK_ABI_NAMESPACE_END`. This change means you can now link different versions of VTK into the same runtime without generating conflicts between VTK symbols. Note: this does not prevent conflicts with third-part symbol (including VTK-m).
- Where to put namespaces:
  - Around classes, functions, variables, typedefs (optional).
  - Inner most named namespaces, there is no need to use the ABI namespace inside of an anonymous namespace.
    - ABI namespace should never be around a named namespace.
  - Forward declarations of classes/functions/variables/typedefs require ABI namespace if their implementation/declarion was inside the ABI namespace.
- Where not to put namespace:
  - Do not namespace around non-exported classes/functions/variables/typedefs (usually found in tests).
  - Do not namespace around main functions.
  - Python bindings cannot be namespaced.
  - Most Utilities are not namespaced, including vtksys/vtkmeta/ksys.
  - It doesn't hurt anything, but it is not required to namespace symbols that are compiled into a driver (such as Wrapping Tools).
- Some VTK modules have C interfaces that cannot be mangled:
  - VTK::CommonCore (GetVTKVersion)
  - VTK::IOXML (Provides a C API, `vtkXMLWriterC_-`)
  - VTK::WrappingPythonCore (Python Wrapping cannot have mangling)
- Thirdpary Libraries and their VTK module wrappers do not have mangling:
  - VTK::metaio
  - VTK::xdmf2
  - VTK::vpic
  - All C libraries (ie. HDF5, netCDF, etc.)
- VTKm CUDA Accelerators do not get mangled:
  - VTK::AcceleratorsVTKmCore
  - VTK::AcceleratorsVTKmDataModel
  - VTK::AcceleratorsVTKmFilters

Build

- VTK_LOGGING_TIME_PRECISION can be used to change the precision of loguru timing output (when VTK_ENABLE_LOGGING is ON).
- VTK_ZSPACE_USE_COMPAT_SDK can be used to control runtime search for zSpace Core Compatibility libraries. Default is ON, disabling the search.
- VTK_GENERATE_SPDX can be used to generate SPDX files for each VTK module. Default is OFF. The generation of SPDX files is considered experimental.
- Added VTK_USE_FUTURE_BOOL configure-time variable. The codebase contains many variables typed as int that really should be bool. But changing them breaks backwards compatibility, and so a vtkTypeBool typedef was introduced which is defined to either int or bool depending on the new VTK_USE_FUTURE_BOOL configure-time variable. This allows for the piecemeal changing of many int variables to vtkTypeBool.

Charts

- vtkChartParallelCoordinates now has a chart legend which can be toggled with the SetShowLegend method. This legend can be customized using the vtkChartLegend API.
- vtkPlotParallelCoordinates now has the option to set a preconfigured color array using SetColorModeToDefault.
- Fixed bug where calling vtkPlotBar.GetLookupTable caused a segmentation fault when no data had been plotted.
- You can now set an array name for the vtkPlotHistogram2D. This allows you to set an array that is not scalar, ie. an array with a number of components greater than 1.

Core

- vtkMath::GetPointAlongLine can be used to compute a point along a line defined by two points and an offset.
- vtkValueFromString is a new low-level function that converts a string to an integer, a floating-point value or a boolean. vtkValueFromString is faster than standard library functions such as the std::strto* function family.
- VTK now provides a way to obtain type names at compile time in the Common/Core/vtkTypeName.h header:

```
#include "vtkTypeName.h"

// ...
std::string typeName = vtk::TypeName<vtkImageData>();
std::cout << typeName << std::endl;
```

- The vtkStringToken class introduces a utility for hashing strings at either compile or run-time and using the resulting integers as tokens. Additional utilities regarding compile-time hashing have also been added:
  - vtkStringManager holds strings hashed at runtime. This makes it possible for the string-token class to return the original string to you in some cases. Because the manager holds a map from string-hash to string, only a single copy of the string is stored no matter how many copies of the token exist.

- Added new `vtk::literals` namespace for creating hashes and tokens at compile time.
  - `""_hash` - returns a 32-bit integer hash of the given string.
  - `""_token` - returns a `vtkStringToken` instance of the given string. Note that because the hash is computed during compilation, you may not call the token's `Data()` method to retrieve the string unless it is inserted at run time by some other code.
- Since hashing is performed at build time, the following example is possible:

```
#include "vtkStringToken.h"
using namespace vtk::literals;
vtkStringToken t;
switch (t.GetId())
{
  case "foo"_hash: foo(); break;
  case "bar"_hash: bar(); break;
  default: vtkErrorMacro("Unknown token " << t.Data()); break;
}
```

- VTK now provides a way to iterate over a class and all its ancestor types (as long as they inherit `vtkObjectBase` and use the `vtkTypeMacro()` to define `Superclass` type-aliases). The `vtk::ParentClasses<T>::enumerate()` function will invoke a functor you pass on `T` and each superclass of `T`. This is used by a new `vtk::Inheritance<T>()` function that inserts the name of each class inherited by `T` into a container you pass to it. See `Common/Core/Testing/Cxx/TestInherits` for example usage.
- `vtkThreadedCallbackQueue` can be used to run functions in the background on different threads. Use the `Push` method to add functions to the queue. The `Push` method returns a `vtkSmartPointer<vtkThreadedCallbackQueue::vtkSharedFutureBase>`, which lets users synchronize tasks.

Data

- Added `vtkImplicitArray` template class that implements a read-only `vtkGenericDataArray` interface which transforms an implicit function mapping integers to values into a practically zero cost `vtkDataArray`. This is helpful in cases where one needs to attach data to data sets and memory efficiency is paramount.
  - Additional backends have been added in the `vtkImplicitArray` framework:
    - `vtkAffineArray` that gets constructed with a slope and intercept and then returns values linearly depending on the queried index.
    - `vtkCompositeArray` that takes an `std::vector<vtkDataArray*>` at construction and returns values as if the list has been concatenated into one array.
    - `vtkConstantArray` that gets constructed with a given value and then returns that same value regardless of the index queried.
    - `vtkStdFunctionArray` which uses a `std::function<ValueType(int)>` backend capable of covering almost any function one might want to use.
    - `vtkIndexedArray` that takes an indexing array (either `vtkIdList` or `vtkDataArray`) and a base `vtkDataArray` at construction and returns values indirected using the indexing array to give access to a shuffled array without the memory cost.
  - Read more about `vtkImplicitArrays` here.

- `ProcessIds` data array is now accessible directly from `vtkDataSetAttributes` just like any other data array (e.g GlobalIds or Normals).
- Added `vtkCellGrid`. It exists to support finite element techniques using novel function spaces, which violate vtkDataSet's assumptions – especially discontinuous Galerkin (DG) elements.
- `vtkPolyhedronUtilities` added to support polyhedron decomposition into tetrahedra. Improves downstream filter results (e.g. contours) on polyhedrons with concave faces.
- Added new `vtkPolyhedron::TriangulateFaces` method.
- Added new `vtkStaticFaceHashLinksTemplate` templated class that can be used to group faces of an unstructured grid and eliminates duplicates.
- Added `vtkHyperTreeGridGeometricLocator` which is a geometric locator for `vtkHyperTreeGrid` datasets.
- `vtkHyperTreeGrid` has a new type of cursor, called `unlimited` cursors. `vtkHyperTreeGridNonOrientedUnlimitedMooreSuperCursor` and `vtkHyperTreeGridNonOrientedUnlimitedGeometryCursor` have been added.
- `vtkDataSet::GetCellNumberOfFaces` can be used to get the number of faces in a given cell.
- `vtkBoundingBox::ComputeBounds` added to compute the bounds for a set of points. This is used for the `GetCellBounds` method in `vtkUnstructuredGrid`, `vtkPolyData`, and `vtkExplicitStructuredGrid`.
- Added `vtkCompositeDataSet::CompositeShallowCopy` which shallow copies up to dataset pointers only.
- Add new `vtkNonLinearCell::StableClip` method and `vtkQuadraticTetra::StableClip` implementation. The goal of this clip is to only decompose a cell if its actually clipped, otherwise keep the non-linear cell in its entirety. Note: this clipping approach will lead to topological holes between decomposed cells and the remaining non-linear cells.
- `vtkPolyData::BuildCells` has been multithreaded.
- Improved performance of `vtkUnstructuredGrid`'s `IsCellBoundary/GetCellNeighbors` methods.
- Improved stability of `vtkCellLocator::FindClosestPointWithinRadius`.

Documentation

- The VTK documentation has undergone a major update and consolidation to enhance its usefulness for developers. The `/Documentation/docs` directory now contains the contents and configuration for the Sphinx-based website, published on the ReadTheDocs platform at https://docs.vtk.org. This consolidates all existing documentation for VTK, including the newly added list of supported data formats, API of all VTK public CMake modules, the VTK formats specification (previously part of vtk-examples), and the general information about the VTK project. Software process and conventions documentation has also been moved from docs.google.com to the new website.
- In addition to the documentation website, two new resources have been introduced: the VTK book, which hosts the markdown version of the VTK book at https://book.vtk.org and VTK examples at https://examples.vtk.org, which contain many examples with redirects put in place to ensure the previous URL remains functional. Many other updates to the documentation have also been made, including improved documentation structure, removal of obsolete documents, and addition of imported third-party projects to the developer guide. VTK documentation now follows a versioning system and is actively maintained alongside the code.
- Contributions and feedback are welcome for all three websites to ensure that the VTK documentation remains up-to-date.
- **Next steps**

- The next steps for the VTK documentation project include setting up a versioning system for docs, doxygen, and the book. Work is also underway to include a description of each `Modules` (as well as a `README.md` file) in `docs.vtk.org`. And there are plans to explore the possibility of using merge request previews for the documentation so that contributors don't have to compile it themselves.
- For `examples.vtk.org`, the plan is to consolidate examples from VTK and `vtk-examples`.
- Pages on the `mediawiki` site will be marked as deprecated, and a link to `docs.vtk.org` will be included.
- These efforts will help ensure that VTK documentation remains user-friendly and accessible to all developers.

Filters

- Added Filters/GeometryPreview module which include filters for creating a preview of the geomertry of a dataset. Current GeometryPreview filters are:
  - `vtkPointSetToOctreeImageFilter`, used to convert a `vtkPointSet` into an image with a number of points per cell target and an `unsigned char` octree cell array.
  - `vtkOctreeImageToPointSetFilter`, used to convert an image with an `unsigned char` octree cell array to a `vtkPointSet`.
  - `vtkPointSetStreamer`, used to stream points as buckets.
- `vtkImageReslice` now supports oriented images, and can reslice an image into a new orientation via the new `SetOutputDirection()` method.
- `vtkDistancePolyDataFilter` can now output directions in conjunction with the (signed/unsigned) distances. This is enabled using `ComputeDirection` (default:off).
- `vtkVortexCore` now outputs 2 extra arrays, `vorticity` and `vorticity_magnitude`.
- `vtkQuadricDecimation` has the following changes:
  - Added new `MapPointData` property to which maps input point data to its decimated output.
  - Added regularization mode. This is enabled by setting `vtkQuadricDecimation::SetRegularize(true)` and `vtkQuadricDecimation::SetRegularization(value)` where `value` is the standard deviation used in the Gaussian distribution.
- `vtkHyperTreeGridGradient` has added support for vector fields. The resulting gradient has 3 times the number of components as the input field. Additionally, vorticity, divergence and Q-Criterion can now be computed.
- `vtkHyperTreeGridContour` now has 2 contour strategies in the 3D case: the former behavior called `USE_VOXELS`, and the new `USE_DECOMPOSED_POLYHEDRA` which can produce better contour results when the generated dual cells used for contouring appear to be concave. Note: `USE_DECOMPOSED_POLYHEDRA` is much slower than the former strategy.
- `vtkExtractCells` has new flags `PassThroughCellIds` and `OutputPointsPrecision`.
- `vtkProbeFilter` has new flag `SnapToCellWithClosestPoint` which can be used with `vtkPointSet` inputs to snap the probe points to the cell with the closest point.
- `vtkPlaneCutter` has new flags `OutputPointsPrecision` and `MergePoints`.
- `vtkPCANormalEstimation` has two new search modes used for the selection of neighbor points: KNN and RADIUS.
- Various optimizations for `vtkGeometryFilter`:

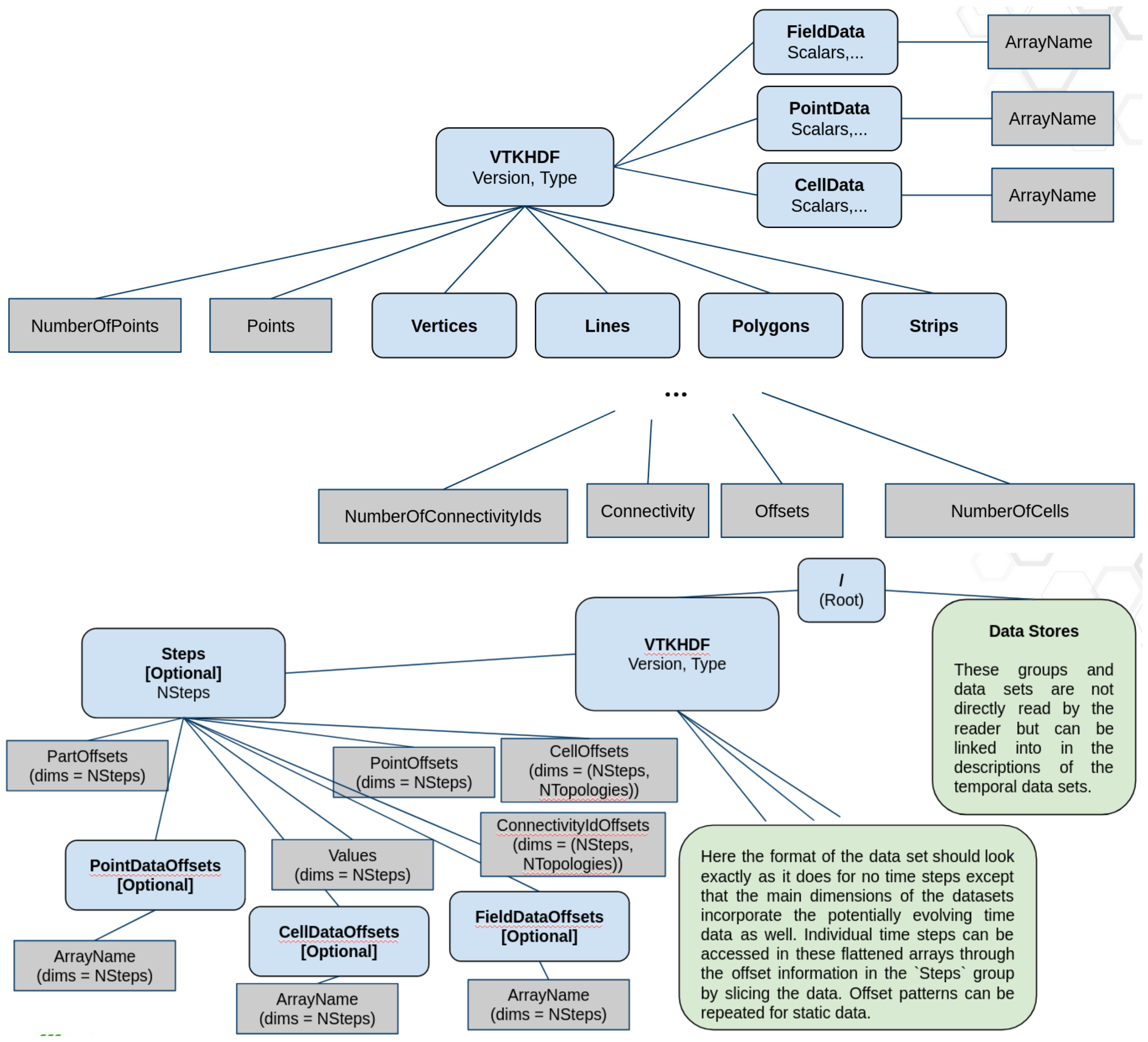
  - Significant memory reduction (x5) with the introduction of `vtkStaticFaceHashLinksTemplate`.
  - Significant speedup (x100) for `vtkGeometryFilter`'s conversion of `vtkUnstructuredGrid` to `vtkPolyData` if the `vtkUnstructuredGrid` has only either vertices, or lines, or polys, or strips.
- Improved performance of `vtkResampleToImage`.
- Improved performance of `vtkDistancePolyDataFilter`.
- Improved performance of `vtkFrustumSelector`.
- Improved performance of `vtkExtractSelection`.
- Improved memory performance for `vtkCellDataToPointData`.
- Added more VTK-m accelerated filter overrides. If the `VTK::AcceleratorsVTKmFilters` is enabled and the CMake option `VTK_ENABLE_VTKM_OVERRIDES` is ON, the following filters will be overridden:
  - `vtkGradientFilter` -> `vtkmGradient`
  - `vtkTableBasedClipDataSet` -> `vtkmClip`
  - `vtkCutter` -> `vtkmSlice`
  - `vtkThreshold` -> `vtkmThreshold`
  - `vtkCellDataToPointData` -> `vtkmAverageToPoints`
  - `vtkPointDataToCellData` -> `vtkmAverageToCells`
- The following filter components have been multithreaded:
  - `vtkRectilinearGrid::GetPoints`
  - `vtkExtractCells`
  - `vtkExtractSelection::ExtractSelectedCells`
  - `vtkExtractSelection::ExtractSelectionPoints`
  - `vtkExtractGeometry`
  - `vtkPolyDataNormals`
  - `vtkProbeFilter::ProbeEmptyPoints`
  - `vtkTableBasedClipDataSet`
  - `vtkThreshold`
- New filter `vtkHyperTreeGridPProbeFilter` can be used to probe a `vtkHyperTreeGrid` using `vtkDataSet`.
- New filter `vtkFieldDataToDataSetAttribute` provides a way to efficiently pass FieldData single-value arrays to other AttributeData. This is useful for composite data, where FieldData can be used to store a single scalar, varying at block level only. Moving this scalar, for instance, to PointData, allows to use it in your pipeline.
- New filter `vtkTensorPrincipalInvariants` computes principal values and vectors from 2D and 3D symmetric tensors.
- The new `vtkYieldCriteria` filter computes different yield criteria from given 2D or 3D symmetric tensors. Available yield criteria currently include:
  - Tresca criterion
  - Von Mises criterion
- Added support for `vtkHyperTreeGrid` with `vtkValueSelector`, `vtkLocationSelector` and `vtkFrustumSelector`. The selections generate can also now be extracted with the `vtkExtractSelection` filter.

- Added `HyperTreeGridToUnstructuredGrid` boolean flag to `vtkExtractSelection` filter to control whether to output an unstructured grid (when `true`) or a hyper try grid (when `false`, the default).
- Fix `vtkHyperTreeGridAxisClip` when `insideout` is `true`.
- The `vtkHyperTreeGridGeometry` filter now provides `PassThroughCellIds` (default `false`) to pass through original cell IDs from the input `vtkHyperTreeGrid` to the output `vtkPolyData`.
- Added support for `vtkHyperTreeGrid` resampling with `vtkResampleWithDataSet` and `vtkPResampleWithDataSet` filters.
- `vtkPolyDataToUnstructuredGrid` is a new multithreaded filter that converts `vtkPolyData` to `vtkUnstructuredGrid`.
- Added `vtkAttributeDataToTableFilter` filter to VTK from ParaView. It serves to turn a data object into a table by shallow copying its attributes into row data. This replaces `vtkDataObjectToTable`, which has been deprecated.
- `vtkBoundaryMeshQuality` filter added to compute quality metrics for boundary meshes.
- `vtkGenerateProcessIds` filter added to generate process ids for both PointData and CellData, and store it via ProcessIds attribute. This filter replaces `vtkProcessIdScalars`, which has been deprecated.
- Added `PointDataWeighingStrategy` option to `vtkCleanUnstructuredGrid` for choosing how to collapse point data. Previously, when merging duplicate points, the point with the lowest index had its data transported to the merged output point. With this new option, you can now choose between:
  - `vtkCleanUnstructuredGrid::FIRST_POINT` (for backwards compatibility): where the point with the lowest index in the input gets the ownership of the merged point.
  - `vtkCleanUnstructuredGrid::AVERAGING`: where the data on the merged output point is the number average of the input points.
  - `vtkCleanUnstructuredGrid::SPATIAL_DENSITY`: where the merged point data is averaged using a partition of the volumes in the cells attached to each point being merged.
- `vtkTableFFT` no longer adds or squeezes certain arrays, like those starting with `vtk`, when the input and the output have a different size.
- `vtkTableFFT` now supports complex valued FFTs.
- Added hemispherical capping support along with texture coordinates for `vtkCylinderSource` to replace the `vtkCapsuleSource` which has now been deprecated.
- `vtkClipClosedSurface` now provides the ability to clip on the reverse side of the clipping planes and also provides the new triangulated geometry created by clipping as a second output.

I/O

- VTK now supports URI parsing, resolution and loading through the two new classes `vtkURI` and `vtkURILoader`. URI support as been implemented to enable resource stream support in readers that need to access multiple resources. For more information about URI usage and loading, please refer to the `vtkURILoader` documentation.
- `vtkResourceStream` added as customizable replacement for standard istreams. Added `vtkFileResourceStream` and `vtkMemoryResourceStream` implementations.
- `vtkResourceParser` added as a high-performance formatted input parser. `vtkResourceParser` parses strings, floats, integers and booleans from any `vtkResourceStream`. Most `std::istream` common features have equivalent methods in `vtkResourceParser`, making migration mostly trivial.

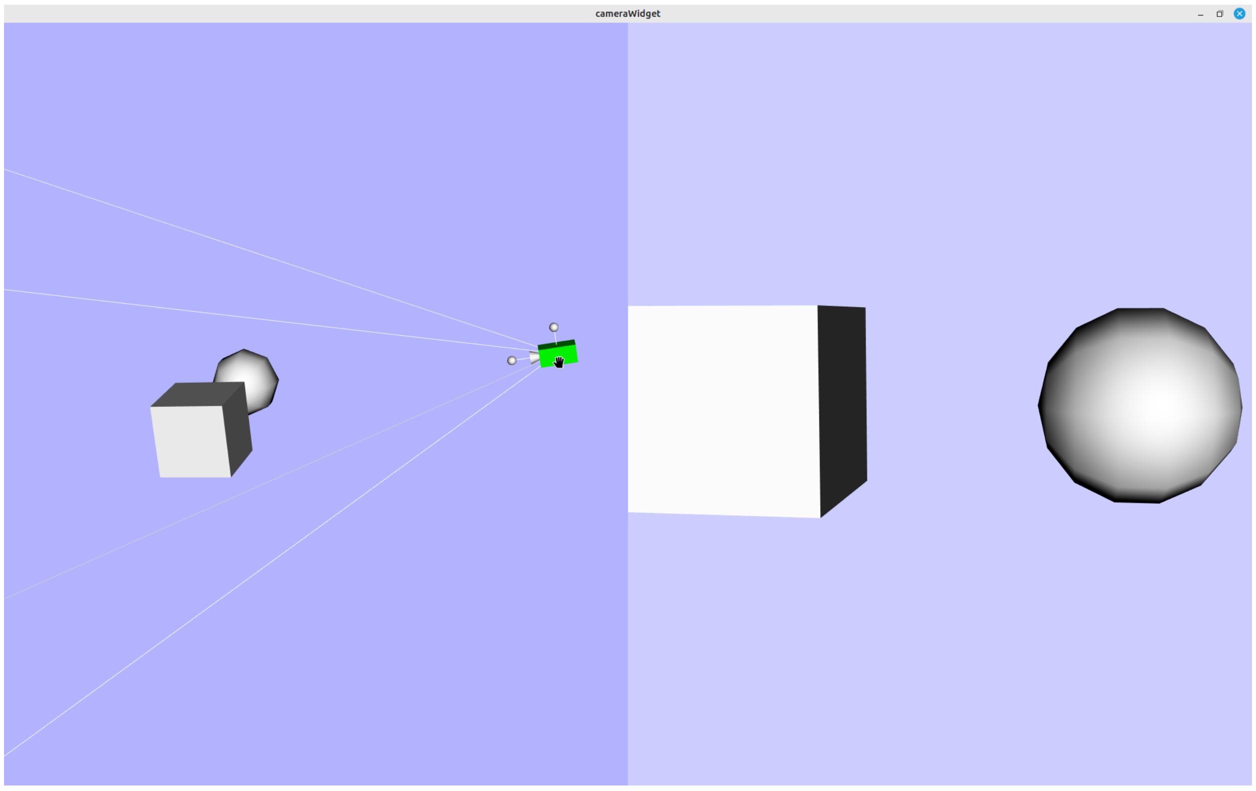
- Added vtkPLYReader, vtkGLTFReader and vtkOBJReader support for reading from vtkResourceStream.
- vtkMemoryResourceStream can now own a streamed buffer, meaning you can free the source buffer after setting it. You can now set source buffer as a std::string, a std::vector or a vtkBuffer*.
- Added vtkNek5000Reader to support NEK5000 data format.
- Added vtkOpenVDBReader in the IOOpenVDB module read to .vdb files.
- Added vtkIOSSWriter writer for the Exodus file format implemented using the IOSS library. Input can be vtkPartitionedDataSetCollection, vtkPartitionedDataSet or vtkDataSet. vtkIOSSWriter can be executed in parallel.
- Added support for higher-order Lagrange cells with vtkIOSSReader.
- vtkIOSSReader now supports mixed-order, 12-node wedge elements.
- Added flag ReadAllFilesToDetermineStructure to vtkIOSSReader which toggles reading all files or only reading the first file to determine mesh structure.
- Added vtkNetCDFUGRIDReader support for reading 2D meshes from NetCDF UGRID files.
- VTKHDF's major version has been incremented to 2 due to the following additions:
  - Added VTKHDF support for both static and transient vtkPolyData files. The metadata schematic for how transient data is read is shown below (first image).
  - Added VTKHDF support for transient ImageData and UnstructuredGrid data. The metadata schematic for how transient data is read is shown below (second image).
  - Specific documentation related to the evolution of the VTKHDF format can be found here.
- Added ANSYS Fluent CFF Reader (Common Fluid Format) into its own dedicated module VTK::IOFLUENTCFF, which currently supports the new vtkFLUENTCFFReader. See further documentation here.
- Added vtkNumberToString::SetHighExponent and vtkNumberToString::SetLowExponent to control the exponent range for scientific or fixed notation.
- Fixed reading fault on vector fields with vtkXMLHyperTreeGridReader.
- Added support for mixed cell unstructured grids in vtkConduitSource. See ValidateMeshTypeMixed and ValidateMeshTypeMixed2D tests in IO/CatalystConduit/Testing/Cxx/TestConduitSource.cxx for more details.
- vtkDataObjectToConduit now supports polygons, vtkPolyData and mixed shapes vtkUnstructuredGrid topologies.


FieldData
Scalars,...
ArrayName
PointData
Scalars,...
ArrayName
CellData
Scalars,...
ArrayName
VTKHDF
Version, Type
NumberOfPoints
Points
Vertices
Lines
Polygons
Strips
...
NumberOfConnectivityIds
Connectivity
Offsets
NumberOfCells
/
(Root)
Steps
[Optional]
NSteps
VTKHDF
Version, Type
Data Stores
These groups and data sets are not directly read by the reader but can be linked into in the descriptions of the temporal data sets.
PartOffsets
(dims = NSteps)
PointOffsets
(dims = NSteps)
CellOffsets
(dims = (NSteps, NTopologies))
ConnectivityIdOffsets
(dims = (NSteps, NTopologies))
Values
(dims = NSteps)
PointDataOffsets
[Optional]
ArrayName
(dims = NSteps)
CellDataOffsets
[Optional]
ArrayName
(dims = NSteps)
FieldDataOffsets
[Optional]
ArrayName
(dims = NSteps)
Here the format of the data set should look exactly as it does for no time steps except that the main dimensions of the datasets incorporate the potentially evolving time data as well. Individual time steps can be accessed in these flattened arrays through the offset information in the `Steps` group by slicing the data. Offset patterns can be repeated for static data.

- Add `vtkMPICommunicator` support for MPI message lengths > MAX_INT, which can now occur in MPI 4.X and later.
- Added `vtkMPICommunicator::NoBlockSend` method that allows for dynamic MPI types.
- Fixed bugs in `vtkMPICommunicator::Test*` and `vtkMPICommunicator::Wait*` that prevented them from being called repeatedly.
- `vtkIOSSReader` can now merge entity blocks into a single block for the exodus format using the flag `MergeExodusEntityBlocks` which is off by default. This is useful e.g. for cases where the entity blocks just represent different cell types but they actually describe the same block.
- Incorrect `vtkEnSightWriter` output has been fixed for `VTK_POLYGON`, `VTK_WEDGE`, `VTK_QUADRATIC_WEDGE`, `VTK_QUADRATIC_EDGE` or `VTK_CONVEX_POINT_SET` cell types. Support for `VTK_POLYHEDRON` has also been added.
- Added flag `WriteNodeIDs` to `vtkEnSightWriter`, which toggles writing node and element IDs to the EnSight data. This makes the output geometry file significantly smaller.
- Added property `SizeAverageCellToPoint` to `vtkOpenFOAMReader` that allows the user to weigh the cell point averaging operation by cell size.

Interaction

- Add support for removing intermediate layers with `vtkExpandMarkedElements`. Added boolean flags `RemoveSeed` and `RemoveIntermediateLayers`. Using these flags will remove their respective layers, keeping only the final expansion layer. This functionality has been extended for use in `vtkSelectionSource` and `vtkSelector`.
- `vtkAppendSelection` `SetColorArray`, `SetInputColor` and `GetInputColor` methods added to associate colors to selections which are used to generate a color array.
- `vtkCameras` can now be oriented with the `vtkCamera3DWidget` and its representation `vtkCamera3DRepresentation`. The representation allows you to move the camera position, target position, to rotate the view up and to update its view angle. See example:



- Added `vtk3DCursorWidget` and `vtk3DCursorRepresentation` to track mouse in a scene. The 3D cursor follows the mouse and is placed on the surface of the actor's scene. Note: this behavior does not currently support volumes.
- Added `SetForce3DArcPlacement` API to `vtkAngleRepresentation2D` which allows users to force correct the 3D placement of arcs that may be misalligned.
- Moved `vtkCompassWidget` and `vtkCompassRepresentation` from `Geovis/Core` to `Interaction/Widgets`. Previously these classes were in a non-working state, but have been fixed with the following changes:
  - `vtkSliderRepresentation` and subclasses: Fixes were applied to correctly calculate the local coordinate for the slider position. They also now honor their `Visibility` parameter.
  - In `vtkCompassWidget` you can now adjust the update `TimerDuration`, `TiltSpeed` and `DistanceSpeed` when clicking on the slider end caps.

- Added standardized color setters (`SetForegroundColor`, `SetHandleColor` and `SetInteractionColor`) to several widgets used by ParaView. These widgets include `vtkBoxRepresentation`, `vtkCurveRepresentation`, `vtkLineRepresentation`, `vtkSphereRepresentation`, `vtkImplicitCylinderRepresentation`, `vtkImplicitPlaneRepresentation`, `vtkDisplaySizedImplicitPlaneRepresentation` and `vtkPointHandleRepresentation3D`. Description of added methods: The intended use of these colors is as follows:

Color	Description
`HandleColor`	Widget handles that are available to interact with via click+drag.
`InteractionColor`	Widget handles the user is interacting with (via a click+drag) or hovering over.
`ForegroundColor`	Widget elements meant to contrast with the background and which are not interactive.

- Added `vtkOrientationWidget` and its representation `vtkOrientationRepresentation` which are used to rotate any actor. The appearance of widget controls are customizable through the representation. See examples:

Math

- Add `GetOctaveFrequencyRange` computation to `vtkFFT` which gets lower/upper frequencies of octaves. Parameters include `octaveSubdivision`, from which you can choose one-third, half, or full octave frequency ranges (default is full) as well as `baseTwo` which toggles between base 2 and base 10 power (default is base 2).

Module System

- Added `vtk_module_wrap_python(HEADERS_DESTINATION)` argument. This argument adds a header into the install tree that initializes the builtin module table for statically built Python modules. This header had not been installed previously.

Python

- Added `ModernizePythonImports.py` script that parse Python scripts and replaces "import vtk" with module specific imports for performance.
- `vtkDataObject`'s now support pickling by the Python `pickle` module.
  - To use this new feature in python, you must first run:

    ```
    import vtkmodules.util.pickle_support
    ```

  - Once you have imported the module the pickling of data objects is straightforward:

    ```
    from vtkmodules.vtkFiltersSources import vtkSphereSource
    import vtkmodules.util.pickle_support
    import pickle
    ```

(continues on next page)

(continued from previous page)

```
sphereSrc = vtkSphereSource()
sphereSrc.Update()

pickled = pickle.dumps(sphereSrc.GetOutput())
unpickled = pickle.loads(pickled)

print(unpickled)
```

- Python 3.12 wheels are now provided for the following platforms:
  - Linux x86_64
  - Linux x86_64 (with OSMesa)
  - macOS x86_64
  - macOS arm64
  - Windows x86_64
  - Windows x86_64 (with OSMesa)

Qt

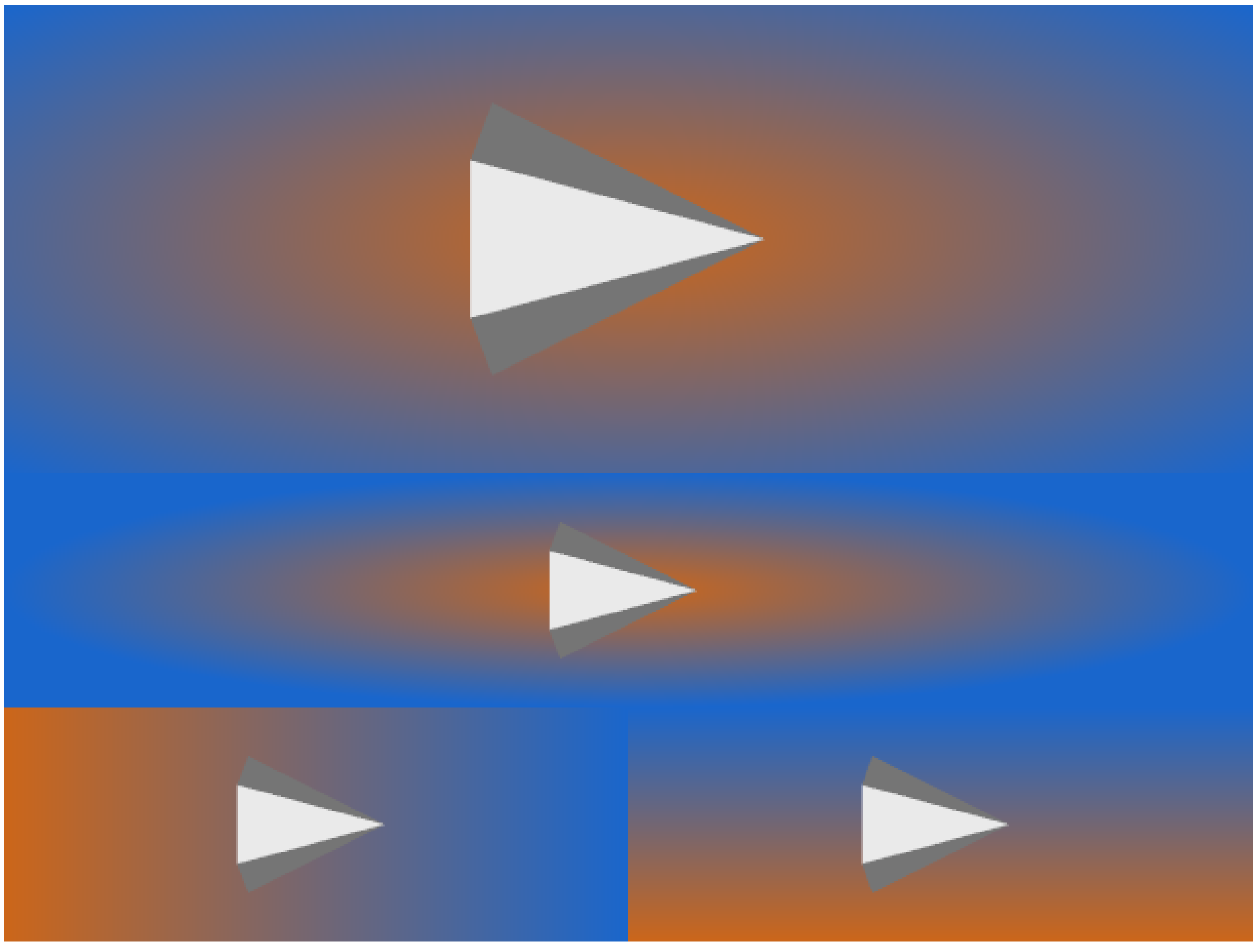
- Added minimal Qt/VTK example application `MinimalQtVTKApp`.
- QML integration support has been upgraded to allow a vtkRenderWindow per QQuick item.
- Added custom cursor methods to `QVTKOpenGLStereoWidget` and `QVTKOpenGLNativeWidget` that get/set the cursor shape.

Rendering

- Added `VTK_USE_WIN32_OPENGL` option to disable Win32 API in `VTK::RenderingOpenGL2` on Windows. This enables OSMesa support on Windows.
- Improved performance of `vtkTupleInterpolator`.
- Added new module providing `zSpace` support to VTK, implementing render window, interactor style, camera, etc. Supports both the "Core zSpace API" (legacy) and the "Core Compatibility zSpace API" (latest).
- `vtkAxisActor2D` labels now use `UseFontSizeFromProperty`, which was formerly used exclusively by the title.
- `vtkImageResliceMapper` now fully supports oriented images, in the same manner as `vtkImageSliceMapper`. This allows the display of arbitrary oblique slices of oriented images, including those where the orientation matrix has a negative determinant.
- Improved performance and consistency of `vtkRenderWindowInteractor::ProcessEvents` across all platforms.
- `vtkCompositePolyDataMapper` can now color separate blocks with different scalar arrays. To use this functionality, turn on `ScalarVisibility` and select a `ScalarMode` and/or a `ColorMode`.
- Improved performance of `vtkContext2D` for rendering large numbers of points.

- vtkCompositePolyDataMapper can now use separate lookup tables and interpolation modes for different blocks in a composite dataset. You can override lookup table and other related attributes like scalar interpolation and scalar ranges. Refer to vtkMapper documentation. Here's a summary:
  - ScalarVisibility: True/False
  - UseLookupTableScalarRange: When true, the mapper shall import the range from the lookup table.
  - InterpolateScalarsBeforeMapping: Applies when mesh is colored using point scalars. This flag decides whether point colors are sampled using texture maps instead of interpolating colors on the GPU after scalars are mapped to colors.
  - ColorMode: Specifies whether to map scalars to colors or directly use the scalars as RGB(A) values.
  - ScalarRange: Specifies a range of scalars for color mapping.
  - LookupTable: Specifies a lookup table.
- vtkCompositePolyDataMapper in VTK::RenderingCore has been improved to efficiently render large datasets. It now performs as well as vtkCompositePolyDataMapper2 in the VTK::RenderingOpenGL2 module, which has now been deprecated. This refactor has significantly impacted the following VTK modules:
  - vtkCompositePolyDataMapper now has an API similar to vtkCompositePolyDataMapper2.
  - vtkCompositeSurfaceLICMapper derives vtkCompositePolyDataMapper instead of vtkCompositePolyDataMapper2.
  - The OSPRay module uses vtkCompositePolyDataMapper instead of vtkCompositePolyDataMapper2.
  - vtkVtkJSSceneGraphSerializer uses vtkCompositePolyDataMapper instead of vtkCompositePolyDataMapper2.
- Added new vtkOpenGLES30PolyDataMapper supports polydata and composite dataset rendering with OpenGL ES 3.0. If VTK was configured with VTK_OPENGL_USE_GLES=ON, this mapper is an override for vtkPolyDataMapper.
- Fixed vtkSurfaceLICMapper crash when rendering lines as tubes or points as spheres.
- vtkTextureObject can now be used to create texture buffers on all OpenGL implementations that support 2D textures.
- Fixed vtkCamera CAVE bugs for head tracking and volume rendering.
- Fixed compositing artifacts when volume rendering in parallel with the OSPRay raycaster. This fix adds a new VolumeSamplingRate parameter to vtkOSPRayRendererNode.
- vtkPolarAxesActor has a number of new features.
  - Radial/polar axes and arc ticks are now customizable. SetRequestedNumberOfRadialAxes, SetRequestedDeltaAngleRadialAxes, SetArcTickMatchesRadialAxes, SetRequestedNumberOfPolarAxes, SetRequestedDeltaAnglePolarAxes, SetArcTickMatchesPolarAxes, SetDeltaAngleMajor, and SetDeltaAngleMinor are all new methods.
  - Tick size is now computed as a ratio of maximum radius by default. You can specify a value for this ratio using SetTickRatioRadiusSize, default is 0.02.
  - You can now change polar arcs resolution per degree. See SetPolarArcResolutionPerDegree, default is 0.2.
  - Text offsets are now customizable with SetPolarTitleOffset, SetRadialTitleOffset, SetPolarLabelOffset and SetPolarExponentOffset.

- `vtkMultiVolume` now supports RGBA volume inputs in a similar way to the existing single-input volume rendering. When turning off the `IndependentComponent` flag of the `vtkVolumeProperty` and providing 4-components to the mapper values are interpreted as RGBA.
- Added new gradient background modes. You can select from various gradient background modes with `vtkViewport::SetGradientMode`. The following modes are available:
  - `VTK_GRADIENT_VERTICAL` Background color is used at the bottom, Background2 color is used at the top.
  - `VTK_GRADIENT_HORIZONTAL` Background color on the left, Background2 color on the right.
  - `VTK_GRADIENT_RADIAL_VIEWPORT_FARTHEST_SIDE` Background color in the center, Background2 color on and beyond the circle ellipse edge. Circle/Ellipse touches all sides of the square/rectangle viewport.
  - `VTK_GRADIENT_RADIAL_VIEWPORT_FARTHEST_CORNER` Background color in the center, Background2 color on and beyond the circle/ ellipse edge. Circle/Ellipse touches all corners of the square/rectangle viewport. See gradient background examples:



Third Party

- `fast_float` added as a vendored package. It is available using the `VTK::fast_float` module.

VTK-m

- VTK-m submodule has been updated to the latest release, VTK-m 2.0.0. Being a major update, it significantly breaks compatibility with the API provided by VTK-m 1.X. Thus, many changes were needed in VTK to make it compatible with VTK-m 2.0.0.
  - All VTK-m cmake targets are now prefixed with `vtkm_`. Exceptions have been made for `vtkm::cuda` and `vtkm::kokkos_hip`, for compatibility with external VTK-m imports.
  - VTK-m VTK module is now called `vtk::vtkvtkm` as opposed to `vtk::vtkm`.
  - `vtkmlib` functions that translate VTK to VTK-m data structures now respect coordinates system changes. Coordinates systems are now represented as a field inside the VTK-m dataset rather than a special/unique component.
- The Fides library has been updated upstream to ensure compatibility with VTK-m 2.0.0. This update in upstream has been brought to VTK to enable using VTK-m 2.0.0 and Fides through VTK.

Deprecated and Removed Features

Legacy

The following APIs were deprecated in 9.1 or earlier and are now removed:

- Python 2 support has been removed.
- Threading types (use C++ `std` classes instead):
  - `vtkSimpleConditionVariable` (`std::condition_variable`)
  - `vtkConditionVariable` (`std::condition_variable`)
  - `vtkMutexType`
  - `vtkSimpleMutexLock` (`std::mutex`)
  - `vtkMutexLock` (`std::lock_guard`)
  - `vtkCritSecType`
  - `vtkSimpleCriticalSection` (`std::mutex`)
- The `EvaluateLocationProjectedNode` method has been removed on the following classes; use `EvaluateLocation` instead:
  - `vtkBezierCurve`
  - `vtkBezierHexahedron`

  - vtkBezierQuadrilateral
  - vtkBezierTetra
  - vtkBezierTriangle
  - vtkBezierWedge
- vtkBezierInterpolation::flattenSimplex has been renamed to ::FlattenSimplex
- vtkBezierInterpolation::unflattenSimplex has been renamed to ::UnFlattenSimplex
- vtkBezierInterpolation::deCasteljauSimplex has been renamed to ::DeCasteljauSimplex
- vtkBezierInterpolation::deCasteljauSimplexDeriv has been renamed to ::DeCasteljauSimplexDeriv
- vtkHigherOrderHexahedron::getEdgeCell has been renamed to ::GetEdgeCell
- vtkHigherOrderHexahedron::getFaceCell has been renamed to ::GetFaceCell
- vtkHigherOrderHexahedron::getInterp has been renamed to ::GetInterpolation
- vtkHigherOrderQuadrilateral::getEdgeCell has been renamed to ::GetEdgeCell
- vtkHigherOrderTetra::getEdgeCell has been renamed to ::GetEdgeCell
- vtkHigherOrderTetra::getFaceCell has been renamed to ::GetFaceCell
- vtkHigherOrderTriangle::eta has been renamed to ::Eta
- vtkHigherOrderTriangle::deta has been renamed to ::Deta
- vtkHigherOrderTriangle::getEdgeCell has been renamed to ::GetEdgeCell
- vtkHigherOrderQuadrilateral::getBdyQuad has been renamed to ::GetBoundaryQuad
- vtkHigherOrderQuadrilateral::getBdyTri has been renamed to ::GetBoundaryTri
- vtkHigherOrderQuadrilateral::getEdgeCell has been renamed to ::GetEdgeCell
- vtkHigherOrderQuadrilateral::getInterp has been renamed to ::GetInterpolation
- vtkIncrementalOctreeNode::InsertPoint without numberOfNodes is removed for the variant with it
- vtkLine::Intersection3D has been replaced by vtkLine::Intersection
- vtkPointData::NullPoint has been replaced by vtkFieldData::NullData
- vtkSelectionNode::INDEXED_VERTICES has been removed
- vtkReaderExecutive has been removed
- vtkThreadMessager has been removed; use C++ std threading support instead
- vtkPassThroughFilter has been replaced by vtkPassThrough
- vtkXMLPPartitionedDataSetWriter has been replaced by vtkXMLPartitionedDataSetWriter
- vtkBlueObeliskData::GetWriteMutex has been replaced by ::LockWriteMutex and ::UnlockWriteMutex
- vtkThreshold::ThresholdByLower has been replaced by ::SetLowerThreshold or ::SetThresholdFunction
- vtkThreshold::ThresholdByUpper has been replaced by ::SetUpperThreshold or ::SetThresholdFunction
- vtkThreshold::ThresholdBetween has been replaced by ::SetLowerThreshold and ::SetUpperThreshold or ::SetThresholdFunction

- `vtkMultiBlockFromTimeSeriesFilter` has been replaced by `vtkGroupTimeStepsFilter`
- `vtkDataSetGhostGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkDataSetSurfaceFilter` methods `::GetUseStrips`, `::SetUseStrips`, `::UseStripsOn`, and `::UseStripsOff` have been removed
- `vtkStructuredGridGhostDataGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkUniformGridGhostDataGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkUnstructuredGridGhostCellsGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkPDataSetGhostGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkPStructuredGridGhostDataGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkPUniformGridGhostDataGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkPUnstructuredGridGhostCellsGenerator` has been replaced by `vtkGhostCellsGenerator`
- `vtkOpenGLRenderer::HaveApplePrimitiveIdBug` has been removed as no supported macOS release has the issue anymore
- `vtkOpenGLRenderWindow` has removed the following methods:
  - `::GetBackLeftBuffer`
  - `::GetBackRightBuffer`
  - `::GetFrontLeftBuffer`
  - `::GetFrontRightBuffer`
  - `::GetBackBuffer`
  - `::GetFrontBuffer`
- `vtkOpenGLRenderWindow::GetOffScreenFramebuffer` has been replaced by `::GetRenderFramebuffer`
- `vtkDataEncoder::PushAndTakeReference` has been replaced by `::Push`
- `vtkGenericOpenGLRenderWindow::IsDrawable` is removed
- `vtkIOSRenderWindow::IsDrawable` is removed
- `vtkCocoaRenderWindow::IsDrawable` is removed
- `vtkRenderWindow::IsDrawable` is removed
- `vtkDIYUtilities::GetDataSets` is replaced by `vtkCompositeDataSet::GetDataSets`
- `vtkCurveRepresentation::*DirectionalLine*` methods have been renamed to `::*Directional*`
- `vtkSimpleImageFilterExample` has been removed
- `vtkExodusIIReaderPrivate::PrintData` has been renamed to `::PrintSelf`
- `vtkEnSightReader::ReplaceWildcards` has been replaced by `vtkGenericEnSightReader::ReplaceWildcardsHelper`
- `vtkQtSQLDatabase::*Port` has been renamed to `::*DbPort` to avoid Windows SDK macro collisions
- The `vtkDataSetSurfaceFilter::GetInterpolatedPointId` overload without `weights` has been replaced by the one with it

Charts

- `vtkPlot` color setter/getter methods with floating point parameters are now suffixed with `F`. The former methods without the suffix have been deprecated. For example:
  - `vtkPlot::SetColor(double r, double g, double b)` has been moved to `vtkPlot::SetColorF(double r, double g, double b)`.

Core

- The `vtkStdString` implicit conversion to `const char*` is deprecated. Instead, call `.c_str()` explicitly on the instance.
- `vtkVariant::ToX` and related string parsing no longer supports `[-]infinity` as a valid float conversion. Only `[-]inf` is now supported.

Data

- Deprecated `vtkCompositeDataSet::RecursiveShallowCopy`, use `vtkCompositeDataSet::ShallowCopy` instead.
- `vtkUnstructuredGrid::GetCellLinks` has been deprecated, `vtkUnstructuredGrid::GetLinks` instead.
- `vtkAbstractCellLinks::BuildLinks(vtkDataSet*)` has been deprecated, use `vtkAbstractCellLinks::BuildLinks()` instead.

Filters

- `vtkDataObjectToTable` is deprecated in favor of `vtkAttributeDataToTableFilter`, which has the same functionality.
- `vtkProcessIdScalars` is deprecated in favor of `vtkGenerateProcessIds`. The following is a migration example:
  - Deprecated `vtkProcessIdScalars` code:

```
vtkNew<vtkProcessIdScalars> processIdsGenerator;
processIdsGenerator->SetInputConnection(someData->GetOutputPort());
processIdsGenerator->SetScalarModeToCellData();
processIdsGenerator->Update();

vtkDataSet* pidGeneratorOutput = processIdsGenerator->GetOutput();
vtkIntArray* pidArray = vtkIntArray::SafeDownCast(pidGeneratorOutput->
↪GetCellData()->GetArray("ProcessId"));
```

  - New `vtkGenerateProcessIds` code:

```
vtkNew<vtkGenerateProcessIds> processIdsGenerator;
processIdsGenerator->SetInputConnection(someData->GetOutputPort());
processIdsGenerator->GeneratePointDataOff();
processIdsGenerator->GenerateCellDataOn();
processIdsGenerator->Update();

vtkDataSet* pidGeneratorOutput = processIdsGenerator->GetOutput();
vtkIdTypeArray* pidArray = vtkIdTypeArray::SafeDownCast(pidGeneratorOutput->
↪GetCellData()->GetProcessIds());
```

- `vtkCapsuleSource` is deprecated in favor of `vtkCylinderSource::SetCapping(true)` and `vtkCylinderSource::SetCapsuleCap(true)`, which has the same functionality.

I/O

- `vtkOpenFOAMReader` support for polyhedral decomposition, `SetDecomposePolyhedra`, has been deprecated.
- `vtkNumberToString::operator()` has been deprecated in favor of `vtkNumberToString::Convert`.

Python

- The VTK wheels no longer provide the `VTK::PythonInterpreter` module as it is unnecessary in such situations.

Rendering

- `vtkXOpenGLRenderWindow::SetSizeNoXResize()` has been deprecated due to structural RenderingUI changes in VTK 9.0.
- `vtkOutputWindowCleanup` has been deprecated as it is no longer used.
- `vtkCompositePolyDataMapper2` has been deprecated in favor of `vtkCompositePolyDataMapper` following the latter's performance improvements.

Other Changes

13.2 9.2

Released on 2022-09-27.

13.2.1 9.2.0 Release Notes

Changes made since VTK 9.1.0 include the following.

Changes

Build

- The `VTK_USE_MPI` and `VTK_USE_TK` options are more lenient and will not force any modules in the `MPI` or `Tk` group, respectively, to be built. Instead, affected modules may be disabled if they are unwanted.
- VTK's packages now hint OpenVR locations (for the build tree or `VTK_RELOCATABLE_INSTALL=OFF` installations).
- Installation destinations for Python modules is now fixed under MinGW.
- Compile fixes for older compilers, mainly GCC 4.8.

Core

- `vtkVector`'s += and -= operators now return a `vtkVector&` as expected. Previously they returned uninitialized `vtkVector` instances which is of little use to anyone.
- `vtkSetGet.h` macros which create setters now have `*Override` variants to use the `override` keyword instead of repeating `virtual`.
- `vtkObject` instances may now be assigned a name used in reporting. It is not copied by `ShallowCopy` or `DeepCopy` copies.
- `vtkAbstractArray::CreateArray` now prefers creating sized integer arrays rather than arrays of basic C types. This is intended to help readers get the correct size instead of having to remember whether `long` is 32 or 64 bits on the given platform.

Data

- The `vtkArrayListTemplate` helper class for `vtkDataSetAttributes` incorrectly held a `vtkDataArray*`. This meant that filters using the class could not support other arrays such as `vtkStringArray`. Now, it holds a `vtkAbstractArray*` to support these types. Users may adapt by using `vtkArrayDownCast` to obtain a `vtkDataArray*` if needed.

Filters

- `vtkArrayCalculator`, `vtkmodules.numpy_interface.dataset_adapter`, and `vtkProgrammableFilter` support for `vtkHyperTreeGrid` has been improved.
- `vtkUnstructuredGridQuadricDecimation::NO_ERROR` has been renamed to `::NON_ERROR` to avoid conflicts with Microsoft Windows SDK headers.
- `vtkImprintFilter::ABSOLUTE` has been renamed to `::ABSOLUTE_TOLERANCE` to avoid conflicts with Microsoft Windows SDK headers.
- `vtkMeshQuality` and `vtkCellTypes` now use a `enum class QualityMeasureTypes` instead of `#define` symbols for metrics.
- `vtkCheckerboardSplatter` no longer has nested parallelism.
- `vtkmProbe` filters now return probed fields as point data rather than cell data.
- `vtkDescriptiveStatistics`'s Kurtosis formula had a mistake which is now corrected.
- `vtkDescriptiveStatistics` previously supported toggling kurtosis, skewness, and variance over sample or population individually. Now, sample or population can be selected using the `SampleEstimate` boolean (on by default). This simplifies interactions with the filter and avoids confusion by mixing and matching. The previous APIs still exist, but do not do anything.
- `vtkPlaneCutter` now frees the sphere trees if the input changes and can handle `vtkUniformGridAMR` inputs.
- `vtkPlaneCutter` now uses `vtkAppendPolyData` internally to merge internal results. This avoids complex `vtkMultiBlockDataSet` inputs from creating complicated sets of `vtkMultiPieceDataSet`. Inputs and outputs are now transformed as follows:
  - `vtkMultiBlockDataSet` input becomes `vtkMultiBlockDataSet`
  - `vtkUniformGridAMR` input becomes `vtkPartitionedDataSetCollection` (previously `vtkMultiBlockDataSet`)
  - `vtkPartitionedDataSetCollection` input becomes `vtkPartitionedDataSetCollection`
  - `vtkPartitionedDataSet` input becomes `vtkPartitionedDataSet`
  - `vtkDataSet` input becomes `vtkPolyData` (previously `vtkPartitionedDataSet`)
- `vtkCellTreeLocator` has moved from `VTK::FiltersGeneral` to `VTK::CommonDataModel`
- `vtkArrayCalculator` no longer calls `Modified()` on any value setting because it causes multi-threaded contention.
- The `vtkArrayRename` filter may be used to rename data arrays within a data set.
- `vtkGeometryFilter` no longer supports the `vtkUnstructuredGrid::FastMode` using the `Degree` flag.
- `vtkTemporalDataSetCache` no longer crashes when a nonexistent timestep is requested.
- `vtkTableFFT` no longer prefixes output array names with "FFT_" like it was in 9.1 and just keep the same name as the input like it was doing before 9.1. A new API has been added to keep 9.1 behavior when needed.
- `vtkContourTriangulator` polygon bounds checking now factors in the tolerance.
- `vtkImageDifference` calculations have been fixed. Note that this may affect testing results.
- `vtkLagrangianParticleTracker` caching invalidation logic fixed.

Interaction

- `vtkFrustumSelection` has been optimized.
- Selection extraction on `vtkUniformGridAMR` has been fixed.

I/O

- The `vtkIOSSReader` now provides `DisplacementMagnitude` to scale point displacement.
- The `vtkIOSSReader` now turns off the `LOWER_CASE_VARIABLE_NAMES` IOSS property.
- `vtkIOSSReader` now reads side sets correctly by avoiding a false positive hit in its internal cache.
- FFmpeg 5.0 is now supported.
- `vtkXdmfReader` no longer caches internal `XdmfGrid` instances to avoid wasting memory. See #19633.
- `vtkJSONSceneExporter` no longer overwrites existing files.
- `vtkGLTFExporter` now exports the correct camera transformation matrix. Imported scenes may use `vtkGLTFImporter::SetCamera(0)` prior to `Update()` to use the original camera location.
- `vtkPLYWriter` may now write the point normals of input meshes, if present.
- `vtkPIOReader` now requires dump files to begin with the problem name. This avoids using an unrelated file for partially written dumps.
- `vtkNetCDFCAMReader` now properly extracts level data.

Rendering

- `vtkProp3D` actors may now be added using different coordinate frames: `WORLD` (the default), `PHYSICAL` (in VR, the physical room's coordinates, in meters), and `DEVICE` (relative to the device). When using `PHYSICAL` or `DEVICE`, a renderer must be provided via the new `SetCoordinateSystemRenderer()` and `SetCoordinateSystemDevice()` methods. Such props should typically use `UseBoundsOff()` to ignore their bounds when resetting the camera.
- Unstable volume rendering configurations are detected.
- Volume rendering now supports more than 6 lights.
- `vtkXOpenGLRenderWindow` and `vtkXRenderWindowInteractor` now properly disconnects from the display when it is not owned.
- Add a missing GIL lock in `vtkMatplotlibMathTextUtilities`.
- Avoid a hard-coded translation when resetting the camera in VR.

Python

- SDKs for each weekly wheels are now available on `vtk.org`. Releases will also have them available.
- `vtkmodules.qt` now supports PyQt6.
- Python 3.10 is now supported by `vtkpython`.
- Python 3.10 wheels are now supported.
- VTK's wrapped classes may now be interposed by using the class' `override` decorator:

```python
from vtkmodules.vtkCommonCore import vtkPoints

@vtkPoints.override
class foo(vtkPoints):
  pass

o = vtkPoints() # o is actually an instance of foo
```

- Note that Python subclasses still cannot override C++ `virtual` functions, cannot alter the C++ class hierarchy, is global, and is ignored when the class uses `vtkObjectFactory` to provide a subclass from its `::New()` method.
- `.pyi` files for autocompletion and hinting in editors are now available in VTK builds and wheels. Note that Windows wheels older than 3.8 do not provide `.pyi` files for platform-specific reasons.
- Starting with 9.2.3, Python 3.11 is supported and newly-deprecated APIs are avoided.
- Starting with 9.2.3, Matplotlib 3.6 is now supported.
- Starting with 9.2.3, `vtk[web]` is required to get web dependencies with the wheels.

Web

- Fix a memory leak in `vtkWebApplication`.
- The render window serializers were updated to better map VTK options to VTK.js options. This includes font coloring for scalar bars and color transfer function discretization.
- `vtkDataSetSurfaceFilter` is used in place of `vtkGeometryFilter`
- The generic mapper serializer now uses `vtkDataSetSurfaceFilter` to always extract a surface from the input dataset.
- Python `print` statements were changed to DEBUG logging statements.

Third Party

- VTK's vendored HDF5 library has been updated to 1.13.1.
- VTK's vendored `verdict` library has been updated.
- VTK's vendored `freetype` library has been updated to 2.12.0.
- VTK's vendored `mpi4py`'s Cython updated to support Python 3.11.
- Avoidance of deprecated APIs in new FFmpeg releases.
- VTK's vendored `libproj` better supports cross-compilation.

Infrastructure

- Modules may now specify license files for the module in their `vtk.module` file. It will automatically be installed.

New Features

Animation

- Animations may now be played in reverse using `vtkAnimationCue`'s direction to `vtkAnimationCue::PlayDirection::BACKWARD`

Build

- When `VTK::AcceleratorsVTKmFilters` is enabled, the `VTK_ENABLE_VTKM_OVERRIDES` option may be turned on to provide factory overrides for other VTK filters. Note that for these overrides to be used, the relevant VTKm modules must be linked (for C++) or imported (for Python) to be effective.

Core

- `vtkMath::Convolve1D` can be used to compute the convolution of two 1D signals using `full`, `same`, or `valid` boundary conditions.
- `vtkReservoirSampler` may be used to perform reservoir sampling. It is intended for selecting random fixed-size subsets of integer sequences (e.g., array indices or element IDs).

Charts

- The parallel coordinates chart now supports multiple selections on the same axis. This includes addition, subtraction, and toggle actions.

Filters

- The `vtkGenerateTimeSteps` filter may be used to add timesteps to shallow-copied data within a pipeline.
- The `vtkHyperTreeGridGradient` filter may be used to compute a gradient over a scalar field. The edges of the dual is used, so all neighbors are considered, but coarse cells are ignored.
- The `vtkExtractParticlesOverTime` filter may extract particles over time that pass through a given volumetric dataset.
- `vtkMultiObjectMassProperties` now also computes the centroids of each object. Centroids are calculated using tetrahedron centroids and uniform density.
- `vtkJoinTables` may perform a SQL-style `JOIN` operation on two `vtkTable` objects. The columns to keep depend on the mode: `intersection` (keep columns common to both tables), `union` (keeps columns present in either table), and `left` and `right` (keeping the keys present in the respective input table).
- `vtkComputeQuantiles` has been split out of `vtkComputeQuartiles` as a new superclass. It supports arbitrary numbers of buckets.
- `vtkMeshQuality` and `vtkCellQuality` have:
  - been multithreaded
  - improved documentation
  - no longer supports the `AspectBeta` tetrahedron metric
  - improved pyramid cell metrics:
    - `EquiangleSkew`
    - `Jacobian`
    - `ScaledJacobian`
    - `Shape`
    - `Volume`
  - improved wedge cell metrics:
    - `Condition`
    - `Distortion`
    - `EdgeRatio`
    - `EquiangleSkew`
    - `Jacobian`
    - `MaxAspectFrobenius`
    - `MaxStretch`
    - `MeanAspectFrobenius`
    - `ScaledJacobian`

    * `Shape`
    * `Volume`
  - new triangle cell metrics:
    * `EquiangleSkew`
    * `NormalizedInradius`
  - new quadrilateral cell metrics:
    * `EquiangleSkew`
  - new tetrahedron cell metrics:
    * `EquiangleSkew`
    * `EquivolumeSkew`
    * `MeanRatio`
    * `NormalizedInradius`
    * `SquishIndex`
  - new hexahedron cell metrics:
    * `EquiangleSkew`
    * `NodalJacobianRatio`

- The new `vtkLinearTransformCellLocator` is a cell locator adaptor which can calculate a transformation matrix from a base dataset to another dataset. This matrix is then used to perform cell locator operations. The `UseAllPoints()` method may be used to use either all dataset points (if the transformation might not be linear) or, at most, 100 sample points sampled uniformly from the dataset's point array.
- `vtkCellLocator`, `vtkStaticCellLocator`, `vtkCellTreeLocator`, `vtkModifiedBSPTree`, and `vtkLinearTransformCellLocator` each have numerous improvements:
  - support for `ShallowCopy()`
  - caching cell bounds has been multithreaded
  - `InsideCellBounds` checks are now cached
  - new `IntersectWithLine` methods sorted by a parametric `t`; this also provides `FindCellsAlongLine` for each locator
  - the `tolerance` parameter may be used to check cell bound intersections
  - The `UseExistingSearchStructure` parameter may be used to not rebuild locators when component data changes, but the geometry stays the same; use `ForceBuildLocator` to rebuild as needed in this case
- `vtkCellTreeLocator` supports 64bit IDs.
- `vtkCellTreeLocator::IntersectWithLine()` and `vtkModifiedBSPTree::IntersectWithLine()` are now thread-safe.
- `vtkCellLocator` is now fully thread-safe.
- The `vtkAlignImageDataSetFilter` has been added which can align image datasets to share a single global origin and offset extents in each component image accordingly. All images must use the same spacing.
- The new `vtkLengthDistribution` filter may be used to estimate the range of geometric length scales preset in a `vtkDataSet`.

- `vtkImageMathematics` can now perform operations on more than two images. Rather than connecting a second image to port 1, all connections are made to port 0 instead. This unifies behavior with other repeatable image filters such as `vtkImageAppend`.
- VTKm's `vtkmContour` filter may be used as a factory override for `vtkContourFilter`.
- VTKm filter factory overrides may be toggled using `vtkmFilterOverrides::SetEnabled()`.
- The `vtkExtractHistogram` filter has been moved from ParaView into VTK.
- `vtkPointDataToCellData` now handles categorical data using multiple threads.
- `vtkSuperquadricSource` now creates pieces using multiple threads.
- Particle traces now support `vtkDataObjectTree` objects to define seed points rather than only `vtkDataSet` objects.
- `vtkStreamTracer` now uses SMP when multiprocessing is not in use.
- `vtkStreamTracer` performance and quality have been improved.
- `vtkFindCellStrategy::FindClosestPointWithinRadius()` has been added.
- `vtkCompositeInterpolatedVelocityField::SnapPointOnCell()` has been refactored from the `vtkInterpolatedVelocityField` and `vtkCellLocatorInterpolatedVelocityField` subclasses.
- `vtkParticleTracerBase` is now multithreaded (with one MPI rank or more than 100 particles).
- `vtkParticleTracerBase` can now use either use a cell locator (the default) or point locator for interpolation.
- `vtkParticleTracerBase` supports different levels of mesh changes over time:
  - `DIFFERENT`: the mesh changes on every timestep.
  - `SAME`: the mesh is the same on every timestep.
  - `LINEAR_TRANSFORMATION`: the mesh is a linear transformation of the prior timesteps (partially applies to point locators as only cell links are preserved).
  - `SAME_TOPOLOGY`: the mesh data changes, but its topology is the same every timestep (only applies to point locators).
- `vtkTemporalInterpolatedVelocityField` can now use the `FindCellStrategy` because it now preserves higher numerical accuracy internally.
- `vtkGeometryFilter` is now multi-threaded over more data types including:
  - `vtkUnstructuredGrid`
  - `vtkUnstructuredGridBase`
  - `vtkImageData` (3D)
  - `vtkRectilinearGrid`
  - `vtkStructuredGrid`
- `vtkGeometryFilter` can now handle ghost and blank cells and points.
- `vtkGeometryFilter` can now remove ghost interfaces using the `RemoveGhostInterfaces` flag (default on).
- The `vtkFiniteElementFieldDistributor` filter can now visualize Discontinuous Galerkin (DG) finite element fields of type `H(Grad)`, `H(Curl)`, and `H(Div)`.
  - Note that all cells must be of the same type and the field data must contain a `vtkStringArray` describing the DG fields, basis types, and reference cells.

Interaction

- `vtkDisplaySizedImplicitPlaneWidget` is now provided. Compared to `vtkImplicitPlaneWidget2`:
  - the outline is not drawn by default
  - the intersection edges of the outline and the plane may be drawn
  - the normal arrow and plane size are relative to the viewport
    - their sizes may be bounded by the widget bounds
  - the origin may be moved freely rather than constrained to the bounding box
  - the handle sizes are larger
  - the plane is represented as a disc
  - the only option for the perimeter is to be tubed or not
  - the perimeter may be selected and resized to change the disc radius
  - the actors are highlighted only when hovered
    - except the plane surface which is highlighted when any actor is hovered over
  - a new plane origin may be picked using `P` or `p`
    - the `ctrl` modifier will snap to the closest point on an object or the camera plane focal point otherwise
  - a new plane normal may be picked using `N` or `n`
    - the `ctrl` modifier will snap to the closest normal on an object or the camera plane normal otherwise
  - picking tolerance is relative to the viewport size
- `vtkResliceImageViewer` may now apply a factor when scrolling.
- `vtkResliceCursorWidgetLineRepresentation` supports `alt+leftclick` to translate along a single axis.
- `vtkVRInteractorStyle` now supports the `Grounded` movement style. The existing movement style is called `Flying`. `Grounded` movement is constrained to an `xy` plane in four directions on one joystick. The other joystick changes elevation.
- `vtkSelection` now supports the `xor` boolean operator.
- `vtkSelectionSource` now supports multiple selection nodes.
- `vtkSelectionSource` may now define the field option using either `FieldType` or `ElementType`.
- `vtkSelectionSource` now defines the `ProcessId` of the selection.
- `vtkAppendSelection` can now append multiple selections through an expression using selection input names.
- `vtkConvertSelection` may now convert `BLOCK` and `BLOCK_SELECTORS` nodes to `INDICES`.

I/O

- vtkFidesReader reader can now use the Inline engine for in-situ processing.
- vtkCatalystConduit may be used to adapt Conduit datasets via the Catalyst library's conduit interactions. This module requires an external catalyst library to be provided. This module includes:
  - vtkConduitSource: a source filter which generates a vtkPartionedDataSet or vtkPartitionedDataSetCollection from a Conduit node (it may also generate vtkMultiBlockDataSet if needed for historical reasons).
  - vtkDataObjectToConduit to convert any vtkDataObject into a Conduit node
  - a Catalyst implementation
- The vtkHDFReader filter now supports overlapping AMR datasets. The specification can be found in the VTK File Formats documentation.
- vtkCGNSReader now support reading cell- or face-centered data arrays for meshes with 3D cells. Note that connectivity must be defined using NGON_n in face-based meshes. Data arrays are then defined with a GridLocation_t of either CellCenter or FaceCenter. The behavior may be selected by setting vtkCGNSReader::DataLocation to vtkCGNSReader::CELL_DATA (the default and previous behavior) or vtkCGNSReader::FACE_DATA.
- vtkPIOReader can now read restart block and even/odd dumps.
- vtkPIOReader will now add the xdt, ydt, zdt, and rho derived variables and calculate them if they are not already present in the restart file.
- vtkIOSSReader now caches time values internally to avoid filesystem contention on HPC systems.
- The new vtkIOSSWriter can write Exodus files using the IOSS library. For now, only element blocks, node sets, and side sets are supported.

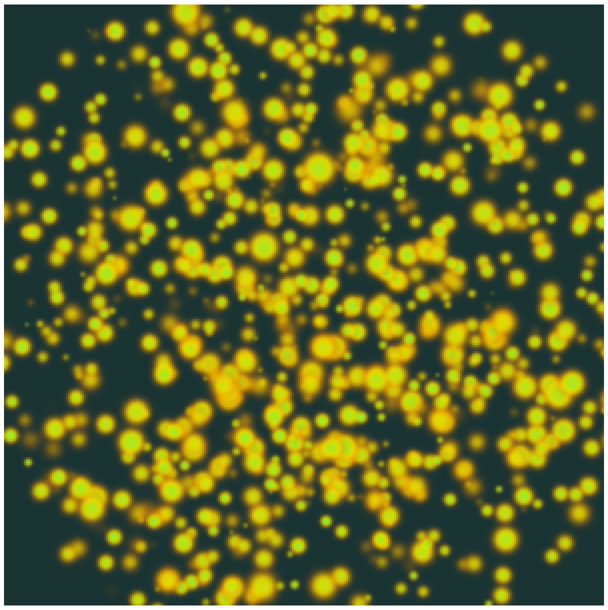
Qt

- QVTKTableModelAdapter may be used to provide a QAbstractItemModel model as a vtkTable to use in a pipeline.

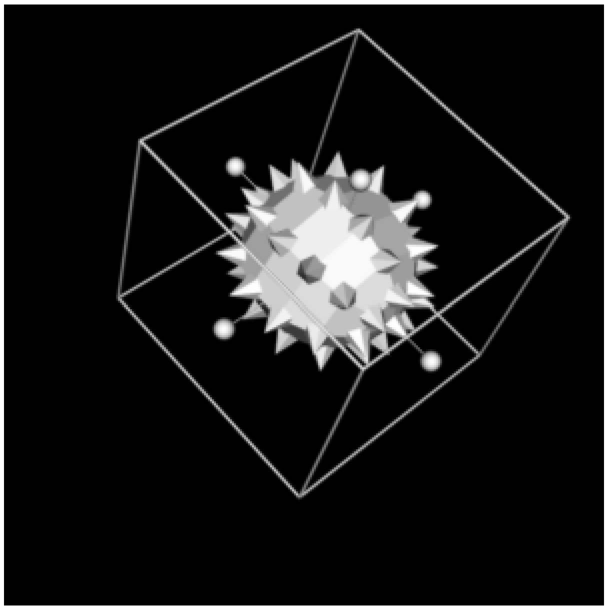
Rendering

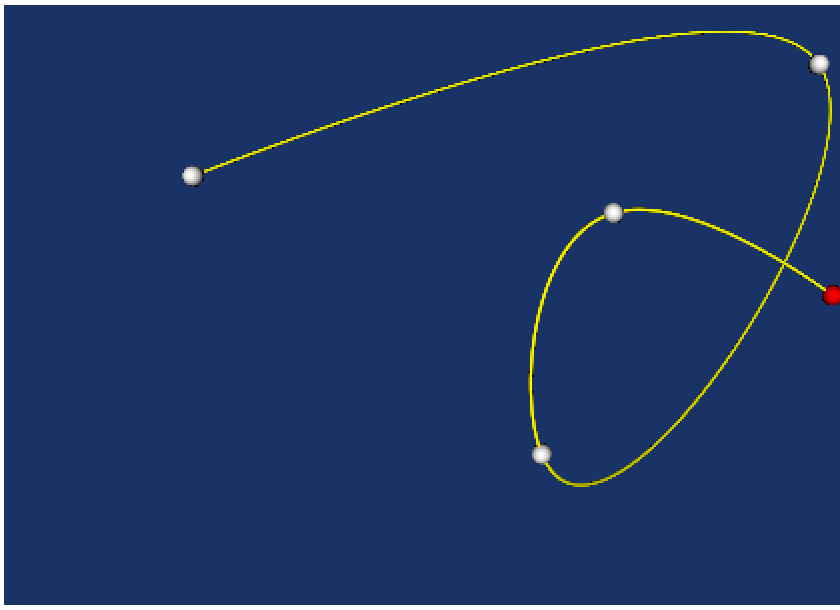
- Basic OpenXR support is supported for virtual reality rendering.
- The vtkHyperTreeGridMapper mapper may be used to render only visible parts of a vtkHyperTreeGrid in 2D.
- Rendering point sets may now use OSPRay's "Particle Volume" when using vtkPointGaussianMapper's ray tracing backend.
- vtkColorTransferFunction::AddRGBPoints may now be called with points and colors in batches for much better performance.
- The WindowLocation API has moved from vtkTextRepresentation to vtkBorderRepresentation so that it can be used by more classes.

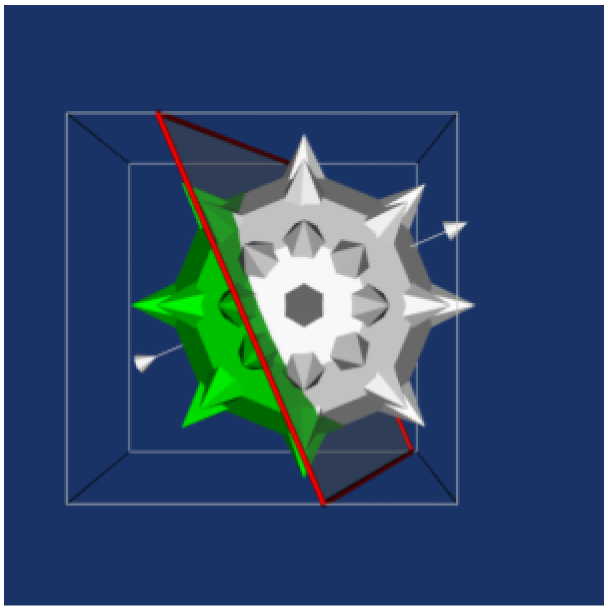
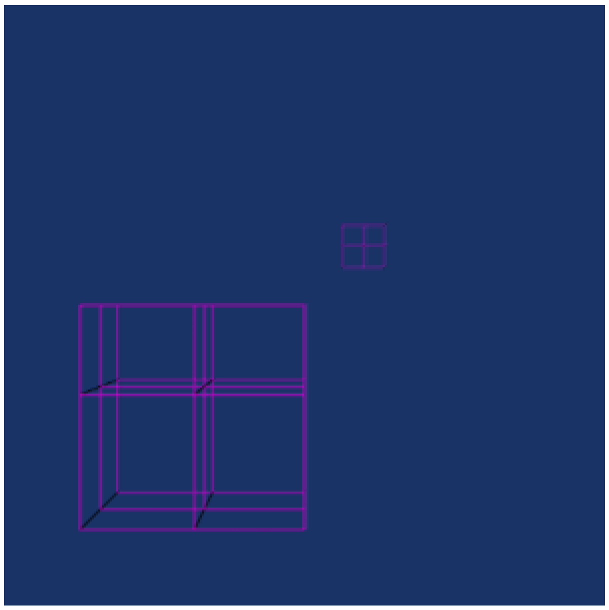
- Volumetric shadows is now supported which allows for a volumetric model to cast shadows on itself. Requires volumetric shading to be enabled. An illumination reach parameter controls how accurate the shadows will be, `0` meaning only local shadows and `1` for shadows across the entire volume.



- Interactive rendering of most widgets are now supported with OSPRay.

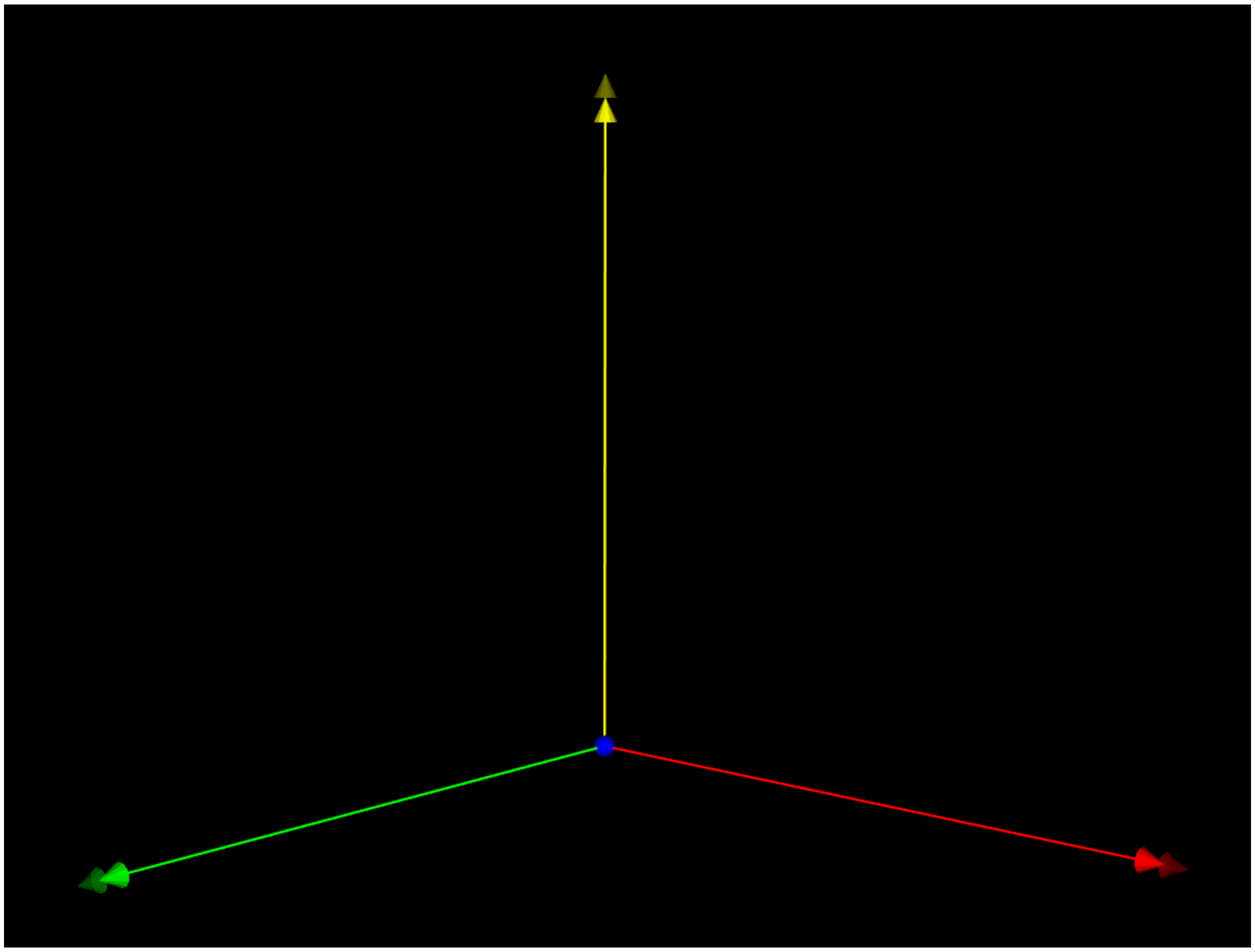




Widgets

- The `vtkCoordinateFrameWidget` controls 3 orthogonal right-handed planes. Axes are rendered proportionally to the viewport size (and is configurable). Interaction may pick a basis point and choose alignment with a surface normal or another point. See this Discourse thread for discussion.
- `vtkHardwarePicker` may be used to pick a point and normal by intersection with a mesh cell or nearest mesh point.



Testing

- The `vtkHyperTreeGridPreConfiguredSource` may be used to generate different `vtkHyperTreeGrid` datasets for testing purposes instead of hand-crafting them.

Wrapping

- Wrapping tools now support Unicode command line arguments.
- `vtkSmartPointer<T>` parameters and return values are now supported in wrapped Python APIs. `std::vector<vtkSmartPointer<T>>` is also supported by appearing as a `tuple` in Python and conversion from any sequence when converting to C++.

Module System

- `vtk_module_sources` is now provided as a wrapper around `target_sources` for VTK module targets.
- `vtk_module_add_module` now supports a `NOWRAP_TEMPLATE_CLASSES` keyword for template classes which should not be wrapped.

Deprecated and Removed Features

Legacy

The following APIs were deprecated in 9.0 or earlier and are now removed:

- `vtkPlot::GetNearestPoint(const vtkVector2f&, const vtkVector2f&, vtkVector2f*)`
- `vtkPlot::LegacyRecursionFlag` (used to help subclasses implement the replacement for the prior method)
- The following APIs have been replaced by `vtkOutputWindow::SetDisplayMode()`:
  - `vtkOutputWindow::SetUseStdErrorForAllMessages()`
  - `vtkOutputWindow::GetUseStdErrorForAllMessages()`
  - `vtkOutputWindow::UseStdErrorForAllMessagesOn()`
  - `vtkOutputWindow::UseStdErrorForAllMessagesOff()`
  - `vtkWin32OutputWindow::SetSendToStdErr()`
  - `vtkWin32OutputWindow::GetSendToStdErr()`
  - `vtkWin32OutputWindow::SendToStdErrOn()`
  - `vtkWin32OutputWindow::SendToStdErrOff()`
- `vtkArrayDispatcher`, `vtkDispatcher`, `vtkDoubleDispatcher` have been replaced by `vtkArrayDispatch`
- Fetching edge and face points via `int` rather than `vtkIdType`:
  - `vtkConvexPointSet::GetEdgePoints(int, int*&)`
  - `vtkConvexPointSet::GetFacePoints(int, int*&)`
  - `vtkHexagonalPrism::GetEdgePoints(int, int*&)`
  - `vtkHexagonalPrism::GetFacePoints(int, int*&)`
  - `vtkHexahedron::GetEdgePoints(int, int*&)`
  - `vtkHexahedron::GetFacePoints(int, int*&)`
  - `vtkPentagonalPrism::GetEdgePoints(int, int*&)`
  - `vtkPentagonalPrism::GetFacePoints(int, int*&)`
  - `vtkPolyhedron::GetEdgePoints(int, int*&)`
  - `vtkPolyhedron::GetFacePoints(int, int*&)`
  - `vtkPyramid::GetEdgePoints(int, int*&)`

  - `vtkPyramid::GetFacePoints(int, int*&)`
  - `vtkTetra::GetEdgePoints(int, int*&)`
  - `vtkTetra::GetFacePoints(int, int*&)`
  - `vtkVoxel::GetEdgePoints(int, int*&)`
  - `vtkVoxel::GetFacePoints(int, int*&)`
  - `vtkWedge::GetEdgePoints(int, int*&)`
  - `vtkWedge::GetFacePoints(int, int*&)`
- Querying point cells with an `unsigned short` count of cells:
  - `vtkPolyData::GetPointCells(vtkIdType, unsigned short&, vtkIdType*&)`
  - `vtkUnstructuredGrid::GetPointCells(vtkIdType, unsigned short&, vtkIdType*&)`
- `vtkAlgorithm::SetProgress()` has been replaced by `vtkAlgorithm::UpdateProgress()`
- The following APIs have been replaced by `vtkResourceFileLocator::SetLogVerbosity()`:
  - `vtkResourceFileLocator::SetPrintDebugInformation()`
  - `vtkResourceFileLocator::GetPrintDebugInformation()`
  - `vtkResourceFileLocator::PrintDebugInformationOn()`
  - `vtkResourceFileLocator::PrintDebugInformationOff()`
- `vtkIdFilter::SetIdsArrayName()` has been replaced by `vtkIdFilter::SetPointIdsArrayName()` and `vtkIdFilter::SetCellIdsArrayName()`
- `vtkExtractTemporalFieldData` has been replaced by `vtkExtractExodusGlobalTemporalVariables`
- `vtkTemporalStreamTracer` and `vtkPTemporalStreamTracer` have been replaced by `vtkParticleTracerBase`, `vtkParticleTracer`, `vtkParticlePathFilter`, or `vtkStreaklineFilter`
- `vtkHyperTreeGridSource::GetMaximumLevel()` and `vtkHyperTreeGridSource::SetMaximumLevel()` have been replaced by `vtkHyperTreeGridSource::GetMaxDepth()` and `vtkHyperTreeGridSource::SetMaxDepth()`
- `QVTKOpenGLNativeWidget`, `QVTKOpenGLStereoWidget`, `QVTKOpenGLWindow` methods have been removed:
  - `::SetRenderWindow()` is now `::setRenderWindow()`
  - `::GetRenderWindow()` is now `::renderWindow()`
  - `::GetInteractor()` and `GetInteractorAdaptor()` have been removed
  - `::setQVTKCursor()` is now `QWidget::setCursor()`
  - `::setDefaultQVTKCursor()` is now `QWidget::setDefaultCursor()`
- `QVTKOpenGLWidget` is replaced by `QVTKOpenGLStereoWidget`
- `vtkJSONDataSetWriter::{Get,Set}FileName()` is now `vtkJSONDataSetWriter::{Get, Set}ArchiveName()`
- `vtkLineRepresentation::SetRestrictFlag()` has been removed
- The following `vtkRenderWindow` methods have been removed:
  - `GetIsPicking()`
  - `SetIsPicking()`
  - `IsPickingOn()`

  - `IsPickingOff()`
- The following APIs have been replaced by `vtkShaderProperty` methods of the same names:
  - `vtkOpenGLPolyDataMapper::AddShaderReplacement()`
  - `vtkOpenGLPolyDataMapper::ClearShaderReplacement()`
  - `vtkOpenGLPolyDataMapper::ClearAllShaderReplacements()`
  - `vtkOpenGLPolyDataMapper::ClearAllShaderReplacements()`
  - `vtkOpenGLPolyDataMapper::SetVertexShaderCode()`
  - `vtkOpenGLPolyDataMapper::GetVertexShaderCode()`
  - `vtkOpenGLPolyDataMapper::SetFragmentShaderCode()`
  - `vtkOpenGLPolyDataMapper::GetFragmentShaderCode()`
  - `vtkOpenGLPolyDataMapper::SetGeometryShaderCode()`
  - `vtkOpenGLPolyDataMapper::GetGeometryShaderCode()`
- The following APIs have been removed (they supported the legacy shader replacements):
  - `vtkOpenGLPolyDataMapper::GetLegacyShaderProperty()`
  - `vtkOpenGLPolyDataMapper::LegacyShaderProperty`
- The following APIs have been removed since only `FLOATING_POINT` mode is now supported:
  - `vtkValuePass::SetRenderingMode()`
  - `vtkValuePass::GetRenderingMode()`
  - `vtkValuePass::SetInputArrayToProcess()`
  - `vtkValuePass::SetInputComponentToProcess()`
  - `vtkValuePass::SetScalarRange()`
  - `vtkValuePass::IsFloatingPointModeSupported()`
  - `vtkValuePass::ColorToValue()`
- `vtkPythonInterpreter::GetPythonVerboseFlag()` has been replaced by `vtkPythonInterpreter::GetLogVerbosity()`
- `vtkUnicodeString` and `vtkUnicodeStringArray` have been removed. The `vtkString` and `vtkStringArray` classes are now fully UTF-8 aware. UTF-16 conversion is no longer possible through VTK APIs.
- `vtkVariant` support for `__int64` and `unsigned __int64` has been removed. They have returned `false` for years.

Core

- `vtkCellTypes` no longer uses `LocationArray`. It was used for `vtkUnstructuredGrid` but is now stored with the class instead. As of this deprecation, all supported APIs are now only `static` methods.
- `vtkUnstructuredGrid::GetCellTypes` is deprecated. Instead, `vtkUnstructuredGrid::GetDistinctCellTypesArray` should be used to access the set of cell types present in the grid.
- `vtkHyperTreeGrid::GetNumberOfVertices()` is now `vtkHyperTreeGrid::GetNumberOfCells()` to align with VTK's usage of the terminology.
- Classes may now opt into the garbage collection mechanism by overriding the `UsesGarbageCollector()` method to return `true` instead of via the `Register()` and `UnRegister()` methods.
- `vtkCriticalSection` is deprecated. `vtkCriticalSection` was intended to be deprecated in VTK 9.1.0, but a warning was never added to it. VTK now has the warning present as it was originally intended.

Filters

- `vtkChemistryConfigure.h` has been deprecated. It previously only provided information to VTK's test suite which is now routed internally instead. There is no replacement and any usage can simply be removed.
- `vtkMFCConfigure.h` has been deprecated. It used to provide information used during the module's build that is now passed through command line flags instead. There is no replacement and any usage can simply be removed.
- `vtkMeshQuality`'s mechanism to run the filter in legacy mode is deprecated. In particular the `CompatibilityMode` and `Volume` members are no longer necessary with the new mode and should not be used anymore.
- `vtkMeshQuality` method renames:
  - `SetQuadQualityMeasureToMaxEdgeRatios` to `SetQuadQualityMeasureToMaxEdgeRatio`
  - `SetHexQualityMeasureToMaxEdgeRatios` to `SetHexQualityMeasureToMaxEdgeRatio`
  - `QuadMaxEdgeRatios` to `QuadMaxEdgeRatio`
  - `TetShapeandSize` to `TetShapeAndSize`
- `vtkDescriptiveStatistics::UnbiasedVariance`, `vtkDescriptiveStatistics::G1Skewness`, and `vtkDescriptiveStatistics::G2Kurtosis` are now deprecated in favor of a single `vtkDescriptiveStatistics::SampleEstimate` instead.
- `vtkCellLocator`, `vtkStaticCellLocator`, `vtkCellTreeLocator`, `vtkModifiedBSPTree`, and `vtkLinearTransformCellLocator` all have deprecated their `LazyEvaluation` flag due to thread-safety issues. `BuildLocatorIfNeeded` is also deprecated for those that supported it.
- `vtkStaticCellLocator:UseDiagonalLengthTolerance()` has been deprecated because it no longer uses `Tolerance`.
- `vtkParticleTracerBase::StaticMesh` is deprecated in preference to `vtkParticleTracerBase::SetMeshOverTime` (an enumeration rather than a boolean).
- `vtkCachingInterpolatedVelocityField`, `vtkCellLocatorInterpolatedVelocityField`, and `vtkInterpolatedVelocityField` filters have been deprecated. Instead, use:
  - `vtkCellLocatorInterpolatedVelocityField` becomes `vtkCompositeInterpolatedVelocityField` with `vtkCellLocatorStrategy`.

   - `vtkInterpolatedVelocityField` becomes `vtkCompositeInterpolatedVelocityField` with `vtkClosestPointStrategy`.
   - `vtkCachingInterpolatedVelocityField` becomes `vtkCompositeInterpolatedVelocityField` with the appropriate strategy.

Interaction

- `vtkExtractSelectedThresholds`, `vtkExtractSelectedPolyDataIds`, `vtkExtractSelectedLocations`, `vtkExtractSelectedIds`, and `vtkExtractSelectedBlock` can now be replaced by `vtkExtractSelection`.
- `vtkHierarchicalBoxDataIterator` is now deprecated in favor of `vtkUniformGridAMRDataIterator`.

Rendering

- `vtkOSPRayRendererNode::VOLUME_ANISOTROPY`, `vtkOSPRayRendererNode::GetVolumeAnisotropy()`, and `vtkOSPRayRendererNode::SetVolumeAnistropy()` are deprecated in favor of`vtkVolumeProperty::SetScatteringAnisotropy()` and `vtkVolumeProperty::GetScatteringAnisotropy()`.

Other Changes

13.3 9.1

Released on 2021-11-04.

13.3.1 9.1.0 Release Notes

Changes made since VTK 9.0.0 include the following.

Changes

Charts

- vtkChartXYZ now applies matrix transformations in the right order (see issue 17542)

Data

- The node numbering for VTK_LAGRANGE_HEXAHEDRON has been corrected to match the numbering of VTK_QUADRATIC_HEXAHEDRON when the Lagrange cell is quadratic. Readers can internally convert data to the new numbering with XML file version 2.2 and legacy file version 5.1. The (0, 1) edge is swapped with the (1, 1) edge:

```
       quadratic                         VTK_QUADRATIC_HEXAHEDRON
VTK_LAGRANGE_HEXAHEDRON                  VTK_LAGRANGE_HEXAHEDRON
    before VTK 9.1                           VTK 9.1 and later

       +_____+_____+                        +_____+_____+
       |\          :\                       |\          :\
       | +         : +                      | +         : +
       |  \     19 +  \                     |  \     18 +  \
    18 +   +-----+-----+                 19 +   +-----+-----+
       |   |       :   |                    |   |       :   |
       |__ | _+____:   |                    |__ | _+____:   |
       \   +        \  +                    \   +        \  +
        +  |         + |                     +  |         + |
         \ |          \|                      \ |          \|
           +_____+_____+                        +_____+_____+
```

- vtkPolyData::ComputeBounds() used to ignore points that do not belong to any cell which was not consistent with other vtkPointSet subclasses. See [ParaView issue #20354][paraview-issue20354]. The previous behavior is available through vtkPolyData::ComputeCellsBounds() and vtkPolyData::GetCellsBounds() (usually for rendering purposes).

Filters

- The VTK::FlowPaths and VTK::ParallelFlowPaths filters now use a vtkSignedCharArray rather than a vtkCharArray since the latter is ambiguous as to whether it is signed or unsigned. This affects the protected API available to subclasses and the usage of the output's ParticleSourceId point data array.
- vtkStaticCellLocator::FindCellsAlongLine() now uses a tolerance for intersections with boxes.
- The tolerance used in vtkStaticCellLocator is now retrieved from the locator's tolerance rather than relying on the size of the dataset. Previous behavior may be restored using the UseDiagonalLengthTolerance property.
- vtkTubeFilter now increases the radius of tubes linearly with respect to the norm of the input vector when the radius is selected to vary by the vector's norm.
- vtkArrayCalculator has been updated to use C++ containers rather than raw pointers.
- vtkArrayCalculator no longer supports the old dot syntax by default and the dot() function must be used instead.

- `vtkArrayCalculator` can now use `exprtk` to parse expressions (the new default). This brings in functionality, but cannot define functions that return vectors.
- `vtkArrayCalculator`'s input variable names must now be sanitized or quoted.

I/O

- There is a new `VTK::IOChemistry` module which contains chemistry-related readers. Moved classes:
  - `vtkCMLMoleculeReader`: from `VTK::DomainsChemistry`
  - `vtkGaussianCubeReader`: from `VTK::IOGeometry`
  - `vtkGaussianCubeReader2`: from `VTK::IOGeometry`
  - `vtkMoleculeReaderBase`: from `VTK::IOGeometry`
  - `vtkPDBReader`: from `VTK::IOGeometry`
  - `vtkVASPAnimationReader`: from `VTK::DomainsChemistry`
  - `vtkVASPTessellationReader`: from `VTK::DomainsChemistry`
  - `vtkXYZMolReader`: from `VTK::IOGeometry`
  - `vtkXYZMolReader2`: from `VTK::IOGeometry`
- `vtkOpenFOAMReader` no longer supports the `include` compatibility keyword (deprecated in 2008).
- `vtkOpenFOAMReader` no longer treats `uniformFixedValue` as special. Proper support requires logic that would require VTK to use OpenFOAM libraries.
- `vtkOpenFOAMReader` no longer supports OpenFOAM 1.3 cloud naming (deprecated in 2007).

Rendering

- `vtkVolumeMapper` and its subclasses now accept `vtkDataSet` input, but ignore any type that is not a `vtkImageData` or `vtkRectilinearGrid` (or their subclasses). Derived classes may need to `SafeDownCast` if the input is assumed to be `vtkImageData`.
- OpenGL framebuffers are now handled using a `RenderFramebuffer` that is internally managed rather than destinations such as `BackLeft` and `Front`. If `SwapBuffers` is on, then the `RenderFramebuffer` will be blitted to a `DisplayFramebuffer` (twice for stereo rendering). The `vtkOpenGLRenderWindow::BlitToRenderFramebuffer` may be used to blit the current read framebuffer to the render framebuffer to initialize color and depth data. The `vtkOpenGLRenderWindow::FrameBlitMode` property may be set to control the following behavior:
  - `BlitToHardware`: blit to hardware buffers (such as `BACK_LEFT`)
  - `BlitToCurrent`: blit to the bound draw framebuffer
  - `NoBlit`: blitting will be handled externally
- `vtkTexture`'s API now more closely matches OpenGL's API. Instead of `Repeat` and `EdgeClamp` properties, `Wrap` is provided using values such as `ClampToEdge`, `Repeat`, `MirroredRepeat`, and `ClampToBorder`. A border color may be selected when using `ClampToBorder`.

Java

- The `byte`, `short`, `long`, and `float` types may now be exposed in the wrapped Java APIs. The Java API now uses types as close as possible to the C++ types used in the wrapped API.

Python

- VTK now defaults to Python 3.x rather than Python 2.
- The `VTK::WebPython` module no longer supports Python 2.
- Python 2.6, 3.2, and 3.3 are no longer supported. Python 3.6 or higher is recommended.
- VTK's web support now requires `wslink>=1.0.0`. This slims down the dependency tree by dropping Twisted in favor of `asyncio` and `aiohttp`.
- VTK's wheels are now built via CI (rather than by hand). Wheels are available for:
  - Python versions 3.6, 3.7, 3.8, and 3.9
  - Platforms `manylinux2014`, `macos10.10`, and `windows`
  - Python 3.9 also supports `macos11.0 arm64`
  - Official wheels do not use any external dependencies, but see `build.md` for instructions on building custom wheels.
- VTK object `repr()` now shows the address of the underlying C++ `vtkObjectBase` and the Python object itself:
  - `<vtkmodules.vtkCommonCore.vtkFloatArray(0xbd306710) at 0x69252b820>`.
- VTK objects which are not derived from `vtkObjectBase` now have a `repr()` that shows the construction method (though this is dependent on how the backing type serializes itself, so it may not be 100% accurate; please file issues for types which look "odd"):
  - `vtkmodules.vtkCommonCore.vtkVariant("hello")`
  - `vtkmodules.vtkCommonDataModel.vtkVector3f([1.0, 2.0, 3.0])`

Rendering

- `vtkTextProperty::LineSpacing` now defaults to `1.0` rather than `1.1`.

Infrastructure

- VTK's deprecation mechanism now marks specific APIs as deprecated so that compilers may warn about its usage. Clients still requiring older APIs can suppress warnings by defining `VTK_DEPRECATION_LEVEL` to `VTK_VERSION_CHECK(x, y, z)` to suppress warnings about APIs deprecated after `x.y.z`.

New Features

Algorithms

- `vtkFFT` is now available to perform discrete Fourier transformations.

Core

- The `vtkGaussianRandomSequence::GetNextScaledValue()`, `vtkMinimalStandardRandomSequence::GetNextRangeVa`
  and `vtkRandomSequence::GetNextValue()` APIs have been added to avoid the `->Next()`, `->GetValue()` ping-ponging.
- `vtkVariant::ToString()` and `vtkVariant::ToUnicodeString()` now support formatting and precision arguments when processing numerical values or arrays.

Charts

- `vtkChartMatrix` may now share `x` and/or `y` axes between charts using the `Link(c1, c2)` and `Unlink(c1, c2)` methods. Labels for the axes may be set using the `LabelOuter(leftBottom, rightTop)` method.
- `vtkChartMatrix` now supports nested `vtkChartMatrix` items.
- `vtkChartMatrix::Paint` and `vtkChartMatrix::GetChartIndex` methods have been refactored to use an iterator-based API.
- `vtkChartXYZ` now resizes dynamically with the scene by managing its margins). Manual calls to `SetGeometry` is no longer necessary.
- `vtkChartXYZ` now supports removing plots.
- `vtkChartXYZ` can now turn off its clipping planes to avoid disappearing plots when zooming in.
- `vtkChartXYZ` can now zoom the axes along with the data.
- `vtkChartXYZ` now supports axes labels.
- `vtkChartXYZ` now uses `vtkTextProperty` for its text rendering.

Data

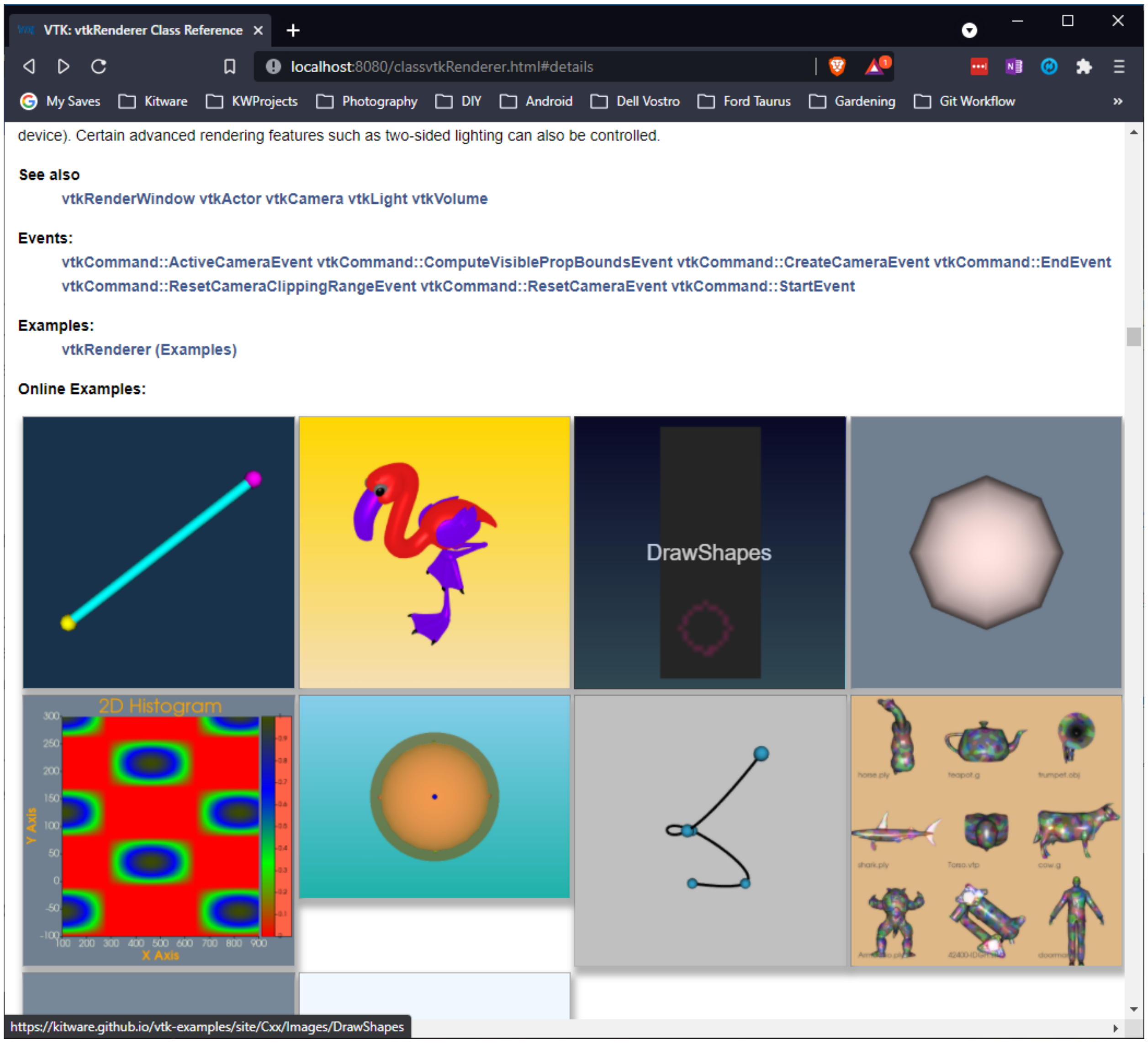
- `vtkDataObjectTypes::TypeIdIsA` may be used to determine if a type is the same as or a specialization of another type.
- `vtkDataObjectTypes::GetCommonBaseTypeId` may be used to find the first common base class of two types.
- `vtkPartitionedDataSetCollection` and `vtkDataAssembly` has been introduced to represent hierarchical datasets rather than `vtkMultiBlockDataSet` and `vtkMultiPieceDataSet`. The new `vtkConvertToPartitionedDataSetCollection` filter can be used to convert any dataset

into a `vtkPartitionedDataSetCollection` with `vtkDataAssembly` representing any hierarchical organization within a `vtkMultiBlockDataSet`. Converting back may be performed with `vtkPConvertToMultiBlockDataSet` or `vtkConvertToMultiBlockDataSet`.

- `vtkUnstructuredGrid::GetCell` is now thread-safe.

Documentation

- VTK's Doxygen documentation now cross-references pages with the vtk-examples website to examples using the class. Images for the classes are now embedded into the class documentation as well.


VTK: vtkRenderer Class Reference
localhost:8080/classvtkRenderer.html#details
My Saves
Kitware
KWProjects
Photography
DIY
Android
Dell Vostro
Ford Taurus
Gardening
Git Workflow
DrawShapes
2D Histogram
Y Axis
X Axis
https://kitware.github.io/vtk-examples/site/Cxx/Images/DrawShapes

device). Certain advanced rendering features such as two-sided lighting can also be controlled.

See also
vtkRenderWindow vtkActor vtkCamera vtkLight vtkVolume

Events:
vtkCommand::ActiveCameraEvent vtkCommand::ComputeVisiblePropBoundsEvent vtkCommand::CreateCameraEvent vtkCommand::EndEvent vtkCommand::ResetCameraClippingRangeEvent vtkCommand::ResetCameraEvent vtkCommand::StartEvent

Examples:
vtkRenderer (Examples)

Online Examples:

Geometry

- The `vtkCell` API now includes support for the 19-node-pyramid named `vtkTriQuadraticPyramid`. Along with the addition of this API, several filters, readers and tests have been updated to incorporate `vtkTriQuadraticPyramid`:
  - Filters:
    - `vtkCellValidator`
    - `vtkUnstructuredGridGeometryFilter`
    - `vtkReflectionFilter`
    - `vtkBoxClipDataset`
    - `vtkCellTypeSource`
  - Readers:
    - `vtkIossReader`

Filters

- `vtk3DLinearGridPlaneCutter` has been updated to also handle cell data. Each triangle of the output dataset contains the attributes of the cell it originated from. Using this class should be faster than using the `vtkUnstructuredGridCutter` or the `vtkDataSetCutter` and should avoid some small projection error. The `vtkCutter` has also been updated to benefit from these changes.
- `vtkMergeVectorComponents` has been added to the `VTK::FiltersGeneral` module which supports `vtkDataSet` objects and may be used to combine three arrays of components into a new output array. This filter supports multithreading via `vtkSMPTools`.
- `vtkArrowSource` now has an option to be placed and oriented around its center which makes placing it after scaling and rotation much easier.
- `vtkDataSetSurfaceFilter` can now extract surfaces from all structured data sets including `vtkImageData`, `vtkStructuredGrid`, and `vtkRectilinearGrid` when they have blanked cells marked using a ghost array.
- `vtkDataSetSurfaceFilter` has a "fast mode" which only considers the outer-most surface without considering external faces within the outer shell which is intended for rendering purposes.
- `vtkExtractExodusGlobalTemporalVariables` now supports field data arrays.
- `vtkExtractEdges` now supports a `UseAllPoints` to use a non-`Locator`-based strategy to skip selecting only the points which are used and instead assumes that all points will be present in the output.
- The `vtkGhostCellsGenerator` filter is now available to generate ghost cells. It uses DIY for MPI communication internally and can handle any `vtkMultiBlockDataSet`, `vtkPartitionedDataSet`, and `vtkPartitionedDataSetCollection` that is filled with the supported input dataset types including `vtkImageData`, `vtkRectilinearGrid`, `vtkStructuredGrid`, `vtkUnstructuredGrid`, and `vtkPolyData`. Ghost cells are only exchanged with elements of the same type and are treated as the "closest" supported common ancestor class. Note that `vtkHyperTreeGrid` is not supported and `vtkHyperTreeGridGhostCellsGenerator` should be used for it instead.
- `vtkGroupDataSetsFilter` may be used to combine input datasets into a `vtkMultiBlockDataSet`, `vtkPartitionedDataSet`, or `vtkPartitionedDataSetCollection`. Each input is added as an individual block and may be named using `SetInputName`.

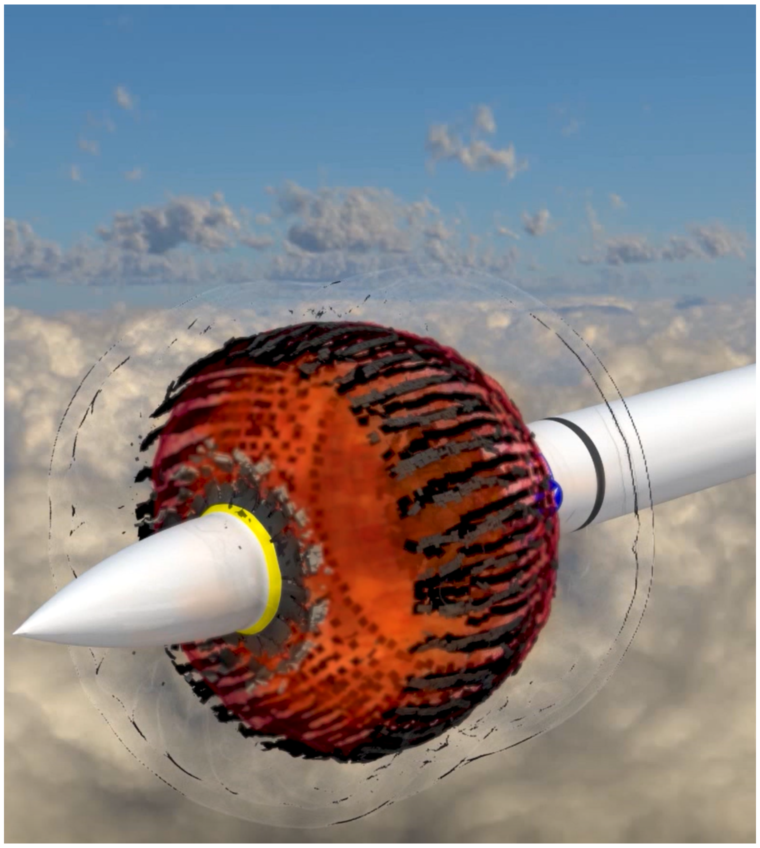
- vtkGroupTimeStepsFilter may be used to turn temporal data into a single vtkMultiBlockDataSet or vtkPartitionedDataSetCollection with all of the temporal data. The output type is vtkPartitionedDataSetCollection unless the input is vtkMultiBlockDataSet in which case another vtkMultiBlockDataSet is output.
- vtkVortexCore's output points now include interpolated variables of the input points.
- vtkVortexCore is now fully multithreaded using vtkSMPTools.
- vtkMergeTimeFilter may be used to synchronize multiple input temporal datasets. The output is a vtkMultiBlockDataSet with one block per input element. The output timestep may be either a union or intersection of the input timestep lists (which may be de-duplicated with either absolute or relative tolerances).
- vtkResizingWindowToImageFilter may be used as an alternative to vtkWindowToImageFilter to create screenshots of any size and aspect ratio using the SetSize(width, height) method regardless of the window size. Note that offscreen buffers are used and therefore memory usage is higher than vtkWindowToImageFilter. Memory usage may be limited using the SetSizeLimit(width, height) method (defaulting to (4000, 4000)). For images larger than the limit the filter will fallback to vtkWindowToImageFilter in order to save memory.
- vtkExtractVectorComponents can now work multithreaded using vtkSMPTools.
- vtkPartitionedDataSetCollectionSource is now available to programmatically produce a vtkPartitionedDataSetCollection.
- vtkThresholdPoints::InputArrayComponent has been added to enable selection of a single component within the active data array. If a value larger than the number of components is used, the magnitude of each array tuple will be used.
- vtkTubeBender is now provided in order to generate better paths for tubes in vtkTubeFilter. This is particularly visible when generating tubes around acute angles.
- vtkArrayCalculator now supports multithreading via vtkSMPTools.
- vtkVectorFieldTopology may be used to compute the topological skeleton of a 2D or 3D vector field given as a set of critical points and separatrices. Separatrices are lines in 2D and surfaces in 3D (computed via vtkStreamSurface).
- vtkTableFFT now offers a frequency column in the output table.
- vtkTableFFT can now compute the FFT per block and then average these results.
- vtkTableFFT can now apply a windowing function before computing the FFT.
- vtkMergeCells can now merge points using double precision tolerances.
- vtkTemporalPathLineFilter can now manage backwards times using its SetBackwardTime() method. When using backwards time, each vtkDataObject::DATA_TIME_STEP() from subsequent ::RequestData() method calls will decrease.
- vtkPlaneCutter used to always generate a vtkMultiBlockDataSet regardless of input type. Now vtkPlaneCutter decides what the output type will be based on the input type.
  - If input type is vtkUniformGridAMR or vtkMultiBlockDataSet, the output type will be vtkMultiBlockDataSet.
  - If input type is vtkPartitionedDataSetCollection, the output type will be vtkPartitionedDataSetCollection.
  - If input type is vtkDataSet or vtkPartitionedDataSet, the output type will be vtkPartitionedDataSet.
- The RemapPointIds functor of vtkRemoveUnusedPoints has now been multithreaded properly. (#18222)

Imaging

- The vtkImageProbeFilter works like vtkProbeFilter, but is designed for image data. It uses vtkImageInterpolator rather than cell and point interpolations. The filter supports SMP acceleration.

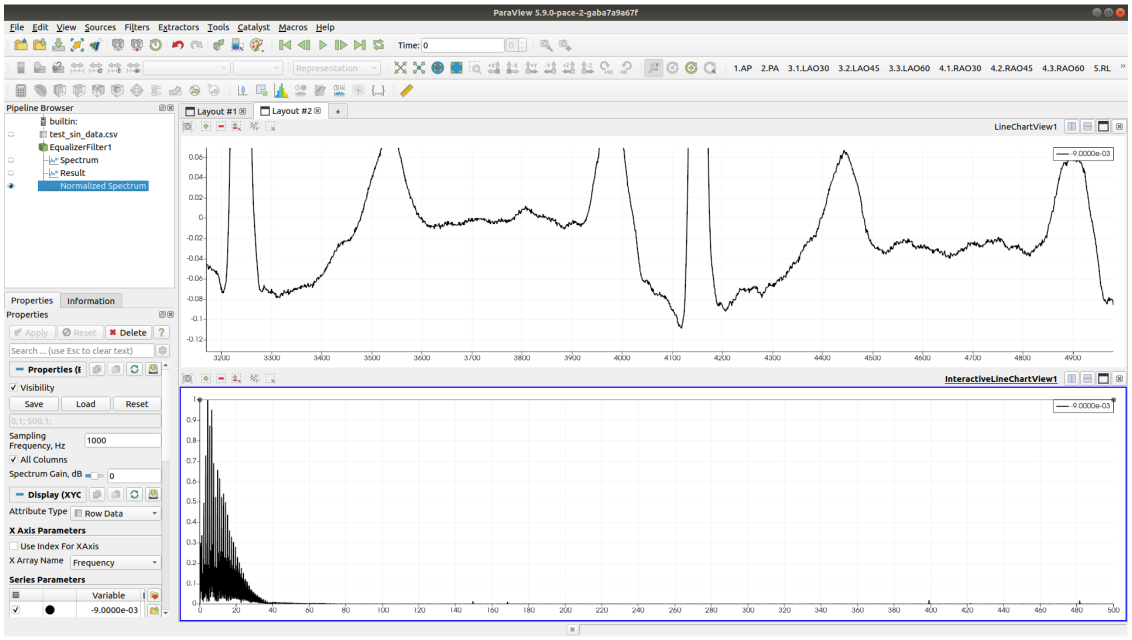
I/O

- VTK can now read ADIOS2 files or data streams using Fides. This can be provided as a JSON file containing the data model or Fides can generate its own data model automatically (see Fides documentation for details). Fides converts the ADIOS2 data to a VTK-m dataset and the vtkFidesReader creates partitioned datasets that contain either native VTK datasets or VTK VTK-m datasets. Time and time streaming is supported. Note that the interface for time streaming is different and requires calling PrepareNextStep() and Update() for each new step.
- The vtkCONVERGECFDReader has been added to read CONVERGE CFD (version 3.1) files containing meshes, surfaces, and parcels as well as support for time series data. Each stream is considered its own block. Cell and point data arrays may be selected using the CellArrayStatus and ParcelArrayStatus APIs. Note that parallel support is not yet available.
- vtkEnSightGoldBinaryReader now supports reading undefined and partial variables from per-node and per-element files.
- VTK's EnSight Gold support can now read asymmetric tensors. This is not supported in EnSight6 files yet.
- The vtkIossReader has been added which supports reading CGNS and Exodus databases and files. The output is provided as a vtkPartitionedDataSetCollection with vtkDataAssembly representing the logical structure. Note that not all CGNS files are supported; only the subset supported by the backing IOSS library. Eventually, the vtkExodusIIReader will be deprecated in preference for this reader.
- The vtkMP4Writer writer may be used to write H.264-encoded MP4 files on Windows using the Microsoft Media Foundation API. The Rate property is available to set the framerate and the BitRate property may be used to set the quality of the output.
- vtkOMFReader may be used to read Open Mining Format files. The output is a vtkPartitionedDataSetCollection with each vtkPartitionedDataSet is one OMF element (point set, line set, surface, or volume).
- vtkOpenVDBWriter may be used to write OpenVDB files. MPI is supported by writing separate files for each rank. Temporal data is also written to a separate file for each timestep.
- vtkTGAReader may be used to read TGA images.
- vtkPDBReader now supports reading PDB files with multiple models.
- vtkPDBReader now generates an array containing the model each atom belongs to.
- vtkVelodyneReader may be used to read Velodyne AMR files.

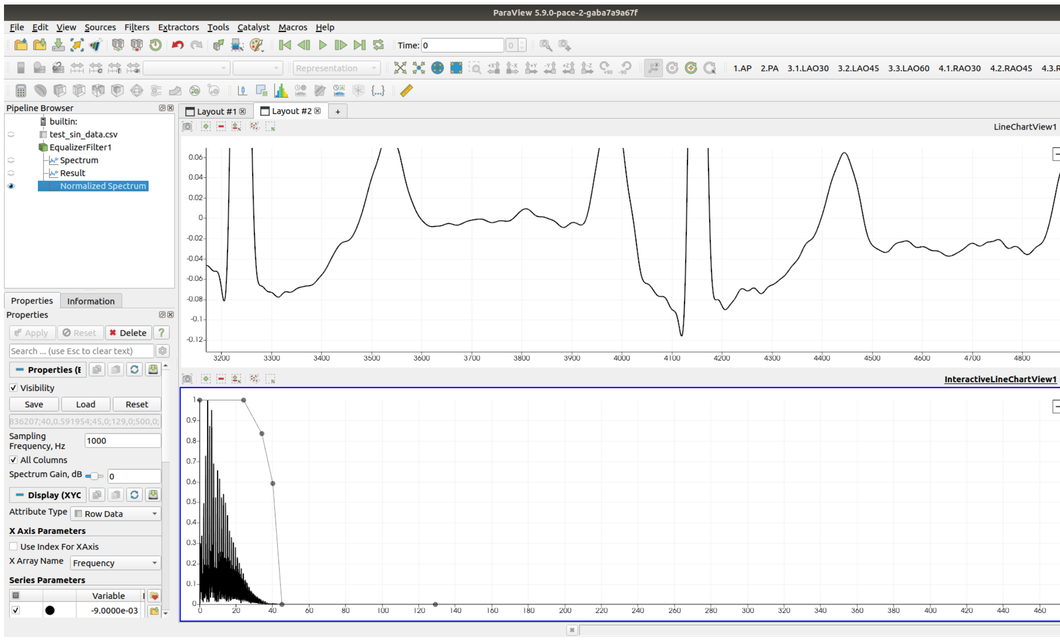


- `vtkNrrdReader` can now read gzip-encoded compressed NRRD files.
- `vtkOpenFOAMReader` now supports reading internal dimensioned fields.
- `vtkOpenFOAMReader` now supports string expansion syntaxes from OpenFOAM v1806 (`#include`, `<case>`, `<constant>`, `<system>`).
- `vtkOpenFOAMReader` now handles mixed-precision workflows much more robustly.
- `vtkOpenFOAMReader` now handles multi-region cases without a default region.
- `vtkOpenFOAMReader` now respects the `inGroups` boundary entry for selection of multiple patches by group.
- `vtkOpenFOAMReader` now properly handles empty zones and has basic support for face zones.
- `vtkOpenFOAMReader` respects point patch value fields suing the correct visitation order.
- `vtkOpenFOAMReader` no longer has hard-coded limits on polyhedral size.
- `vtkOpenFOAMReader` now preserves uncollated Lagrangian information.
- `vtkOpenFOAMReader` now avoids naming ambiguities for Lagrangian and region names using a `/regionName/` prefix for non-default regions.

Interaction

- A new 3D camera orientation widget. The widget's representation is synchronized with the camera of the owning renderer. The widget may be used to select an axis or using its handles.
- `vtkSelectionNode` now supports `BLOCK_SELECTORS` to select whole blocks from a composite dataset using a selector expression for hierarchy or an associated `vtkDataAssembly` for `vtkPartitionedDataSetCollections`.
- Interactive 2D widgets have been added (ported from ParaView). The `vtkEqualizerFilter` and `vtkEqualizerContextItem` are now available using this feature.
  - Source data: 

Sampling Frequency, Hz: 1000
All Columns
Spectrum Gain, dB: 0
Attribute Type: Row Data
X Axis Parameters
Use Index For XAxis
X Array Name: Frequency
Series Parameters
Variable: -9.0000e-03

  - After applying the filter: 

Sampling Frequency, Hz: 1000
All Columns
Spectrum Gain, dB: 0
Attribute Type: Row Data
X Axis Parameters
Use Index For XAxis
X Array Name: Frequency
Series Parameters
Variable: -9.0000e-03

- VTK's OpenVR's input model has been updated to be action-based and supports binding customization within the OpenVR user interface. This includes controller labeling and user configuration.

- vtkEventData now understands an “Any” device so that handlers can look for all events and do filtering internally. See merge request 7557 for an example of how to update custom 3D event handling.
- vtkResliceCursorWidget now refreshes when scrolling.
- Frustum selection of lines and polylines now only considers the line itself, not their containing area (i.e., they were treated as polygons during selection).
- Selections of vtkDataAssembly may be limited to a subset of blocks using a collection of xpath selectors for the dataset.
- vtkChartXYZ can now be controlled using the arrow keys for rotation.
- vtkScalarBarActor now supports custom tick locations via vtkScalarBarActor::SetCustomLabels() and vtkScalarBarActor::SetUseCustomLabels().

Java

- Java 1.7 is now required (bumped from 1.6).
- The wrapped Java API now handles strings more efficiently by handling encoding in the Java wrappers directly. No APIs are affected.

Python

- vtkPythonInterpreter::SetRedirectOutput can be used to disable Python output to vtkOutputWindow.
- VTK now offers two CMake options for deployments that can help to handle Python 3.8’s DLL loading policy changes on Windows. This allows import vtkmodules to handle PATH manipulations to ensure VTK can be loaded rather than relying on vtkpython to do this work.
  - VTK_UNIFIED_INSTALL_TREE: This option can be set to indicate that VTK will share an install tree with its dependencies. This allows VTK to avoid doing extra work that doesn’t need to be done in such a situation. This is ignored if VTK_RELOCATABLE_INSTALL is given (there’s no difference there as VTK assumes that how VTK is used in such a case is handled by other means).
  - VTK_DLL_PATHS: A list of paths to give to Python to use when loading DLL libraries.
- Python wrappers will now generate deprecation warnings when the underlying VTK API is deprecated. Since Python silences DeprecationWarning by default, the warnings must be allowed via:

```python
import warnings
warnings.filterwarnings("default", category=DeprecationWarning)
```

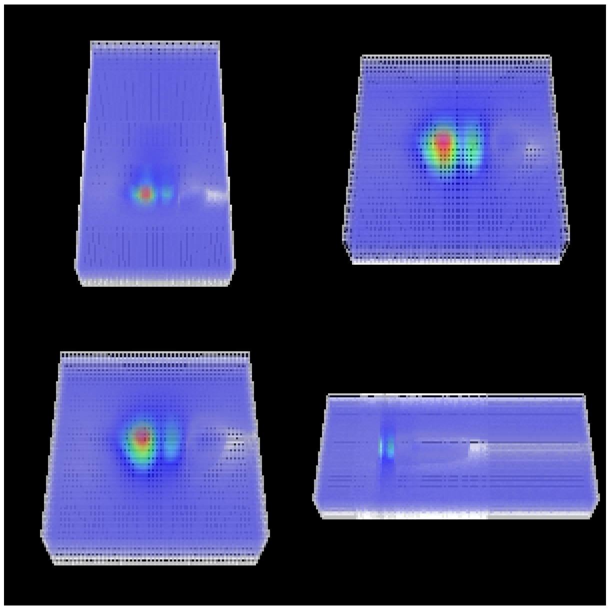
- With Python 3.6 and newer, VTK APIs marked with attributes that indicate that paths are expected will now support pathlike objects.
- Wrapped Python APIs now contain docstrings with type hints as described in PEP 484. This will help with IDE tab completion and hinting.
- VTK’s vtkmodules package and vtk module now provide __version__ attributes.
- The vtkmodules.web.render_window_serializer module may be used to additionally serialize vtkPolyData, vtkImageData, and mergeToPolyData optionally using the requested_fields=['*'] argument.

Qt

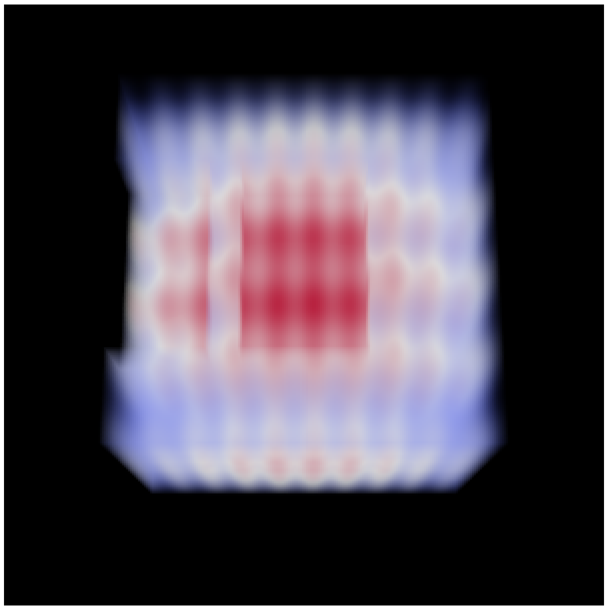
- VTK now supports Qt6 using the `VTK_QT_VERSION` variable. This may be set to `Auto` in which case the first of Qt6 or Qt5 found will be used.
- The `VTK::GUISupportQtQuick` module has been added which supports the necessary integration classes as well as the QML module infrastructure required to import VTK into a QtQuick application.

Rendering

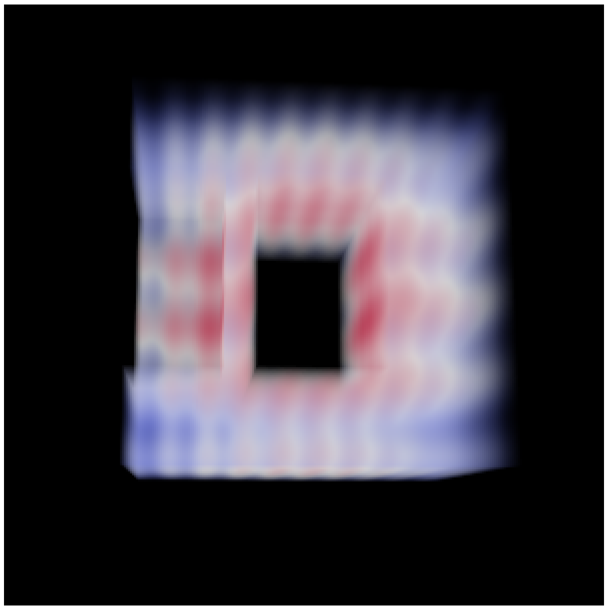
- The `vtkOutlineGlowRenderPass` renders a scene as a glowing outline. Combined with layered renderers this creates a very visible highlight without altering the highlighted object.
- VisRTX and OptiX now offer trace-level debugging information to determine why they may not be available.
- `vtkMultiBlockUnstructuredGridVolumeMapper` has been added to volume render the entirety of a multi-block unstructured grid without merging them.
- The physical-based render shader now supports anisotropic materials. This may be modified using the `Anisotropy` and `AnisotropyRotation` properties. Support for these values from a texture is also available.
- `vtkBlockItem` may now resize itself based on the label specified. Additionally, options for the brush, pen, and text, padding, margins are available.
- Multi-volume ray-casting now supports shading.
- `vtkMatplotlibMathTextUtilities` now supports multi-line (using `\n`) and multi-column (using `|` separators) text. `vtkTextProperty` now has a `CellOffset` property to control the spaces between columns (in pixels).
- The GPU-based ray-casting volume mapper now supports rendering non-uniform rectilinear grids. Volume streaming via block partitioning is not yet supported.



- `vtkOpenGLMovieSphere` may be used to display spherical movies using OpenGL. Both regular 360° video and stereo 360° video is supported where stereo streams are split into left and right eye rendering passes. The video is sent to the graphics card as YUV textures which are decoded to RGB in the associated shaders.
- VTK's VisRTX support is now compatible with OSPRay 2.0.
- The OpenGL `vtkPolyData` mappers now provide a way to handle jitter introduced by rendering with single-precision vertex coordinates far from the origin. The `PauseShiftScale` parameter may be used to suspend updates during user interactions.
- `vtkResliceCursorRepresentation::BoundPlane()` is now offered to show the entire resliced image when rotating.
- `vtkOpenGLPolyDataMapper` now supports a `vtkSelection` object to display selected ids or values directly from the mapper itself. Selections are rendered in a second pass as a wireframe using the `vtkProperty::SelectionColor` color.
- The GPU-based ray-casting volume mapper now supports direct volume rendering with the blanking of cells and points defined by individual ghost arrays.
  - Uniform grid with blanking:



- Image data with ghost cells and points:



- Volume rendering may now use independent scalar arrays for the `x` and `y` dimensions of 2D transfer functions.

Web

- `render_window_serializer.py` now supports serialization of a `vtkRenderWindow` that contains `vtkVolume`, `vtkVolumeProperty`, or `vtkVolumeMapper`.

SMP

- `vtkSMPTools::For()` can now be used on iterators and ranges. This can be especially useful for containers that are not indexed such as `std::map` and `std::set`.
- `vtkSMPTools::LocalScope` may be used to call a `vtkSMPTools` method with a specific configuration within a scope. The configuration structure takes a maximum thread number and/or a backend.
- `vtkSMPTools` now has an `STDThreads` backend which uses C++'s `std::thread`.
- `vtkSMPTools::Transform` and `vtkSMPTools::Fill` may be used as replacements for `std::transform` and `std::fill`.
- Multiple backends may now be compiled into VTK at build time rather than a separate build per backend. The default may be selected using the `VTK_SMP_IMPLEMENTATION_TYPE` CMake variable or the `VTK_SMP_BACKEND_IN_USE` environment variable at runtime. Enabling a backend is controlled by the `VTK_SMP_ENABLE_<backend>` CMake variable at build time.
- The `VTK_SMP_MAX_THREADS` environment variable is now available to limit the number of threads any SMP task will use.
- `vtkSMPTools` now supports nested parallelism using the `NestedParallelism` property (defaults to `false`) and the `IsParallelScope` query to detect such scoping.

Wrapping

- Wrapped classes no longer require a `vtk` prefix.
- Hierarchy files are used exclusively for type checking.
- The wrapping tools now keep an internal cache of which header files exist where on the system to avoid repeated lookups when resolving `#include` search paths.

Module System

- The `vtk_module_add_module(NOWRAP_HEADERS)` argument may be used to list public and installed headers which should not be exposed for wrapping purposes.
- The `vtk_module_add_module(NOWRAP_CLASSES)` argument may be used to list class names for which the headers are treated as `NOWRAP_HEADERS` arguments.
- The `vtk_module_add_module(HEADER_DIRECTORIES)` argument may be used to indicate that the relative path of headers from the current source or binary directory should be preserved upon installation.
- The `vtk_module_install_headers(USE_RELATIVE_PATHS)` argument may be used to indicate that the relative path of headers from the current source or binary directory should be preserved upon installation.
- The `vtk_module_build`, `vtk_module_wrap_python`, and `vtk_module_wrap_java` APIs now support a `UTILITY_TARGET` argument. The target named using this argument will be privately linked into every library created under these APIs. This may be used to provide compile or link options to every target. Note that the target given must be installed, but it may be installed with the same export set given to any of these APIs.
- The module system now supports target-specific components using the `vtk_module_build(TARGET_SPECIFIC_COMPONENTS)` argument.

Infrastructure

- VTK plans to hold to a new minor (or major) release every six months with releases in or around April and October each year.
- VTK now uses GitLab-CI for testing.
- OSPRay support now detects Apple's Rosetta translation environment and refuses to run due to the environment not supporting instructions used within OSPRay.
- `vtkGetEnumMacro` and `vtkSetEnumMacro` are now available to work with `enum class` properties.
- VTK now requires CMake 3.12 to build and to consume. This is mainly due to the usage of `SHELL:` syntax to properly support some MPI use cases.
- `vtkSOADataArrayTemplate` compilation should use less memory now that its template instantiations are split into separate TU compilations.
- The `VTK_TARGET_SPECIFIC_COMPONENTS` option may be specified to provide target-specific installation components.

Third Party

- An external `ioss` library may now be provided to VTK's build.
- An external `pegtl` library may now be provided to VTK's build.
- `exprtk` is now included in VTK (an external copy is supported).
- `fmt` is now included in VTK (an external copy is supported, though VTK requires patches which have been merged upstream but not yet released).
- VTK's embedded third party packages have been updated:
  - `cli11` 2.0.0
  - `eigen` 3.3.9
  - `exodusII` 2021-05-12
  - `expat` 2.4.1
  - `freetype` 2.11.0
  - `glew` 2.2.0
  - `hdf5` 1.12.1
  - `jpeg` 2.1.0
  - `libxml2` 2.9.12
  - `lz4` 1.9.3
  - `lzma` 5.2.5
  - `mpi4py` 3.0.3
  - `netcdf` 4.8.0
  - `ogg` 1.3.5
  - `pugixml` 1.11.4
  - `sqlite` 3.36.0
  - `tiff` 4.3.0

Deprecated and Removed Features

Legacy

Some APIs had been deprecated for a long time. The following APIs are now removed.

- `vtkDataArrayTemplate` (deprecated since Dec 2015)
- `vtkObjectBase::PrintRevisions` and `vtkObjectBase::CollectRevisions` (deprecated since 2012)
- `VTK___INT64` and `VTK_UNSIGNED___INT64` (deprecated since Mar 2017)
- `vtkArrayCalculator::SetAttributeMode*` and `VTK_ATTRIBUTE_MODE_*` macros (deprecated in Jun 2017)

- vtkContourGrid::ComputeGradients (deprecated in Dec 2018)
- vtkSMPContourGridManyPieces, vtkSMPTransform, vtkThreadedSynchronizedTemplates3D, and vtkThreadedSynchronizedTemplatesCutter3D (deprecated in Sep 2017)
- vtkAbstractImageInterpolator::GetWholeExtent (deprecated in Mar 2016)
- vtkImageStencilData::InsertLine (an overload) (deprecated in Nov 2014)
- The RemoveBlockVisibilites method from vtkCompositeDataDisplayAttributes, vtkCompositeDataDisplayAttributesLegacy, and vtkCompositePolyDataMapper2 (deprecated in Jul 2017)
- vtkOpenVRPropPicker (deprecated in Apr 2018)

Core

- vtkLegacy.h and VTK_LEGACY_REMOVE are now deprecated and vtkDeprecation.h and its mechanisms should be used instead.
- Building with kits is no longer supported with static builds. Since the goal of a kit build is to reduce the number of runtime libraries needed at startup, a static kit build does not make much sense. Additionally, some dependency setups could not be resolved in such a build (as witnessed by ParaView) and a proper fix is not easy, so disabling the support makes more sense at this time.
- The vtkToolkits.h header provided preprocessor definitions indicating some features within VTK's build. However, these were inaccurate in single-configure builds since the information was not available when the header was configured.
  - VTK_USE_VIDEO_FOR_WINDOWS: now available in vtkIOMovieConfigure.h
  - VTK_USE_VFW_CAPTURE: now available in vtkIOVideoConfigure.h as VTK_USE_VIDEO_FOR_WINDOWS_CAPTURE, but the old name is given for compatibility.
- vtkUnicodeString and vtkUnicodeStringArray are now deprecated since VTK, since 8.2, has assumed UTF-8 for all string APIs. As such, vtkUnicodeString and vtkUnicodeStringArray did not provide any additional information. Users which used them to convert UTF-8 to UCS-2 for Windows API usages should instead use VTK::vtksys's SystemTools APIs for converting such strings.
- vtkAtomic is removed in favor of std::atomic. As such, vtkAtomic.h and vtkAtomicTypeConcepts.h are no longer provided.
- Threading-related classes are now deprecated in favor of C++11 standard library mechanisms.
  - vtkConditionVariable: std::condition_variable_any
  - vtkCriticalSection: std::mutex
  - vtkMutexLock: std::mutex
  - vtkSimpleConditionVariable: std::condition_variable_any
  - vtkSimpleCriticalSection: std::mutex
  - vtkSimpleMutexLock: std::mutex
  - vtkThreadMessanger: std::mutex and std::condition_variable_any
- vtkSetGet.h no longer includes <math.h>.
- vtkVariant.h methods Is__Int64 and IsUnsigned__Int64 were marked deprecated, though they have been unconditionally returning false since 2015.

Filters

- `vtkDataSetSurfaceFilter` no longer supports generation of triangle strips. The `vtkStripper` filter may be used to generate them if needed.
- `vtkConfigure.h` is now deprecated and split into more focused headers. The headers which now contain the information:
  - `vtkBuild.h`: `VTK_BUILD_SHARED_LIBS`
  - `vtkCompiler.h`: Compiler detection and compatibility macros.
  - `vtkDebug.h`: `VTK_DEBUG_LEAKS` and `VTK_WARN_ON_DISPATCH_FAILURE`
  - `vtkDebugRangeIterators.h`: `VTK_DEBUG_RANGE_ITERATORS` and `VTK_ALWAYS_OPTIMIZE_ARRAY_ITERATORS`
  - `vtkEndian.h`: `VTK_WORDS_BIGENDIAN`
  - `vtkFeatures.h`: `VTK_ALL_NEW_OBJECT_FACTORY` and `VTK_USE_MEMKIND`
  - `vtkLegacy.h`: `VTK_LEGACY_REMOVE`, `VTK_LEGACY_SILENT`, and `VTK_LEGACY`
  - `vtkOptions.h`: `VTK_USE_64BIT_IDS` and `VTK_USE_64BIT_TIMESTAMPS`
  - `vtkPlatform.h`: `VTK_REQUIRE_LARGE_FILE_SUPPORT` and `VTK_MAXPATH`
  - `vtkSMP.h`: `VTK_SMP_${backend}` and `VTK_SMP_BACKEND`
  - `vtkThreads.h`: `VTK_USE_PTHREADS`, `VTK_USE_WIN32_THREADS`, `VTK_MAX_THREADS`
    - Also includes `VTK_THREAD_RETURN_VALUE` and `VTK_THREAD_RETURN_TYPE`, but `vtkMultiThreader.h` is guaranteed to provide this.
- Old ghost cell filters are deprecated in favor of `vtkGhostCellsGenerator`.
  - `vtkUnstructuredGridGhostCellsGenerator`
  - `vtkPUnstructuredGridGhostDataGenerator`
  - `vtkStructuredGridGhostDataGenerator`
  - `vtkPStructuredGridGhostDataGenerator`
  - `vtkUniformGridGhostDataGenerator`
  - `vtkPUniformGridGhostDataGenerator`
- `vtkThreshold`'s `ThresholdByLower()`, `ThresholdByUpper()`, and `ThresholdBetween()` methods are deprecated in favor of `vtkThreshold::LowerThreshold`, `vtkThreshold::UpperThreshold`, and `vtkThreshold::ThresholdFunction` properties.

Python

- Python 2 support, which reached its end-of-life in January 2020 is deprecated.

Soft deprecations

Some groundwork has been laid down to deprecate existing classes with new features in this release, but have not been formally deprecated yet. Users are encouraged to use the new APIs to help ensure that the transition when they are deprecated is smoother.

- `vtkMultiBlockDataGroupFilter` usage should be replaced by `vtkGroupDataSetsFilter`.
- `vtkMultiBlockFromTimeSeriesFilter` usage should be replaced by `vtkGroupDataSetsFilter`.
- `vtkExodusIIReader` usage should be replaced by `vtkIossReader`.
- The random number APIs from `vtkMath` should be moved to `vtkGaussianRandomSequence` or `vtkMinimalStandardRandomSequence` as needed.

Other Changes

13.4 9.0

Released on 2020-05-01.

13.4.1 9.0.0

See Discourse for release notes.

13.4.2 9.0.2

VTK 9.0.2 collects fixes to 9.0.1 which have been made since its release. Of particular interest are the fixes to macOS rendering, support for the macOS arm64 platform, and updates for API changes in external libraries.

13.4.3 New classes

- Added a `vtkImageProbeFilter` which works like `vtkProbeFilter`, but for `vtkImageData`

13.4.4 New support

- `enum class` setters and getters are now supported via `vtk{Get,Set}EnumMacro`

13.4.5 Fixes

- The `QVTKRenderWidget.h` is now installed.
- `vtk3DLinearGridPlaneCutter` guards against `nullptr` points and cells
- The composite date mapper now iterates over data blocks properly
- `vtkStringArray::Resize` takes tuple elements into account
- `vtkArrowSource` now supports scalong and rotation around the origin or the arrow's center point
- The `VTK::DomainsChemistryOpenGL2`, `VTK::RenderingContextOpenGL2`, and `VTK::RenderingOpenGL2` modules are added to the `Rendering` group to avoid missing implementations of rendering components
- `vtkCutter` enables point merging when requested through a `vtkPointLocator` which merges points
- `vtkAxesActor` bounds calculations improved to avoid assumptions about range values
- `vtkWindowLevelLookupTable` out-of-range colors are now initialized properly
- `vtkImageReslice::RequestInformation` is refactored handle common image information passing
- `vtkImageReslice` creates a new interpolator in `::GetInterpolator`; this new interpolator now uses the same interpolation mode as `vtkImageReslice` itself

macOS

- macOS wheels are now built and uploaded by VTK's CI
- Fixes for macOS OpenGL state tracking (related to `GL_SCISSOR`)
- Multisampling on macOS with Intel graphics turned off for volume rendering
- OpenGL state tracking on macOS with layers is improved (rather than using the wrong context between layers)
- OSPRay is disabled when running under macOS Rosetta

Third Party

- HDF5 has been updated to address errors on newer Xcode compilers
- HDF5 macOS universal2 compilation fixes
- `VTK::mpi` now disables C++ bindings for SGI MPT as well
- Usage of `numpy.character` is removed (deprecated in NumPy 1.19)
- Avoidance of APIs deprecated in Python 3.9
- Compilation with newer libfreetype resolved (`FT_CALLBACK_DEF` usage removed)

13.4.6 9.0.3

A minor patchset on top of 9.0.2 to fix problems with the new release process' configuration when building the wheels.

Wheels

- Disable `VTK_DEBUG_LEAKS` in wheel builds
- Remove long-deprecated API usage in the Python bindings

13.5 8.2

Released on 2019-01-30.

Release notes for version 8.2 can be found at https://www.kitware.com/vtk-8-2-0.

13.6 8.1

Released on 2017-12-22.

Release notes for version 8.1 can be found at https://www.kitware.com/vtk-8-1-0.

13.7 8.0

Released on 2017-06-26.

Release notes for version 8.0 can be found at https://www.kitware.com/vtk-8-0-0.

13.8 7.1

Released on 2016-11-14.

Release notes for version 7.1 can be found at https://www.kitware.com/vtk-7-1-0.

13.9 7.0

Released on 2016-02-16.

Release notes for version 7.0 can be found at https://www.kitware.com/vtk-7-0-0.

13.10 6.3

Released on 2015-08-31.

Release notes for version 6.3 can be found at https://www.kitware.com/vtk-6-3-0.

13.11 6.2

Released on 2015-03-03.

Release notes for version 6.2 can be found at https://www.kitware.com/vtk-6-2-0.

13.12 6.1

Released on 2014-01-22.

Release notes for version 6.1 can be found at https://www.kitware.com/vtk-6-1-0.

13.13 6.0

Released on 2013-06-21.

Release notes for version 6.0 can be found at https://www.kitware.com/vtk-6-0-0.

13.14 5.10

Released on 2012-05-14.

Release notes for version 5.10 can be found at https://www.kitware.com/vtk-5-10-now-available.

13.15 5.8

Released on 2011-08-29.

Release notes for version 5.8 can be found at https://www.kitware.com/vtk-5-8-0.

13.16 5.6

Released on 2010-05-28.

Release notes for version 5.6 can be found at https://itk.org/Wiki/VTK_5.6_Release_Planning.

13.17 5.4

Released on 2009-03-20.

Release notes for version 5.4 can be found at https://www.kitware.com/vtk-5-4-released.

13.18 5.2

Released on 2008-08-27.

Release notes for version 5.2 can be found at https://www.kitware.com/vtk-5-2-released.

13.19 5.0

Released on 2005-12-15.

Release notes for version 5.0 can be found at https://www.kitware.com/vtk-5-0-released.

PYTHON MODULE INDEX

INDEX