Kitware SOURCE

SOFTWARE DEVELOPER'S QUARTERLY

Issue 10 • July 2009

TABLE OF CONTENTS

EDITOR'S NOTE

The Kitware Software Developer's Quarterly Newsletter contains articles related to the development of Kitware projects in addition to a myriad of software updates, news and other content relevant to the open source community. In this issue, David Cole explains organize large software projects into subprojects which can be viewed on a CDash dashboard using CDash 1.4. Berk Geveci provides a basic overview for using ParaView's Python interface, a rich scripting support system which allows users and developers can gain access to the ParaView engine. Claudio Silva, Juliana Freire and John Schreiner from VisTrails, Inc. collaborated on an article about the VisTrails plugin for ParaView and how the plugin can be used for exploratory visualization; this plugin was incorporated into the recent ParaView 3.6 release. And Blogger and QPid contributor, Steve Huston, wrote an article discussing why and how Apache QPid converted to CMake.

The Kitware Source is just one of a suite of products and services that Kitware offers to assist developers in getting the most out of its open-source products. Each project's website contains links to free resources including mailing lists, documentation, FAQs and Wikis. In addition, Kitware supports its open-source projects with technical books, user's guides, consulting services, support contracts and training courses. For more information on Kitware's suite of products and services, please visit our website at www.kitware.com.

RECENT RELEASES

PARAVIEW 3.6

Kitware, Sandia National Laboratories and Los Alamos National Lab are proud to announce the release of ParaView 3.6. The binaries and sources are available for download from the ParaView website. This release includes several new features along with plenty of bug fixes addressing a multitude of usability and stability issues including those affecting parallel volume rendering.

Based on user feedback, ParaView's Python API has undergone a major overhaul. The new simplified scripting interface makes it easier to write procedural scripts mimicking the steps users would follow when using the GUI to perform tasks such as creating sources, applying filters, etc. Details on the new scripting API can be found on the Paraview Wiki.

We have been experimenting with adding support for additional file formats such as CGNS, Silo, Tecplot using VisIt plugins. Since this is an experimental feature, only the Linux binaries distributed from our website support these new file formats.

ParaView now natively supports tabular data-structures thus improving support for CSV files including importing CSV files as point-sets or structured grids.

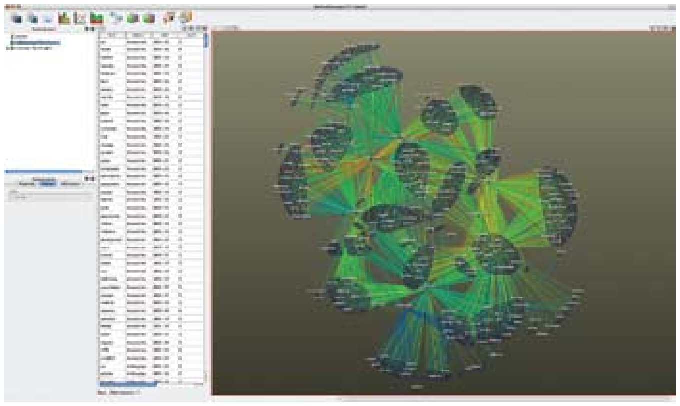
We have completely redesigned the charting/plotting components with several performance fixes as well as usability improvements. It is possible to plot arrays from arbitrary datasets directly using Plot Data filter. Upon hovering over the plots tooltips are shown which detail the plotted values.

In an effort to better support animations involving the camera, we have added support for specifying camera movements along splines or for orbiting around objects in space.

This version has many GUI usability improvements including, but definitely not limited to:

- Color palettes which make it easier to switch between color schemes that are suitable for printing and for screen.
- Improved support for temporal readers and filters.
- Axes annotations and scalar bar for 2D render view.
- Zooming to selected region in 3D view.
- Quick launch for creating sources and filters using Ctrl+Space or Alt+Space.

Apart from these enhancements, ParaView includes a pre-alpha release of OverView, an application developed using the ParaView application framework. OverView is a generalization of the ParaView scientific visualization application designed to support the ingestion, processing and display of informatics data. The ParaView client-server architecture provides a mature framework for performing scalable analysis on distributed memory platforms, and OverView uses these capabilities to analyze informatics problems that are too large for individual workstations. This application still contains many experimental features and is not yet documented, but feel free to try it out and report bugs and feature requests.



ParaView 3.6 also includes a pre-alpha release of StreamingParaView, another application developed using the ParaView application framework. StreamingParaView processes structured datasets in a piecewise fashion, on one or many processors. Because the entire dataset is never loaded into memory at once, StreamingParaView makes it possible to visualize large datasets on machines that have insufficient RAM to do so otherwise. Piece culling, reordering and caching preserve ParaView's normally high interactivity while streaming. This application still contains many experimental features and is not yet documented, but we encourage users to try it out and report bugs and feature requests. Note that both OverView as well as StreamingParaView are only available through source and not distributed with the binaries on the website.

Bugs, feature requests and any questions or issues can be posted to the ParaView Mailing List at paraview@paraview.org. Also check out the new feedback forum on paraview.org; this forum allows ParaView users to vote for features and ideas submitted by the user community which they'd like to see added or modified in ParaView.

VTK 5.4.2

The VTK 5.4.2 patch was released on June 4, 2009; this release is now available for download from the VTK website. A full list of changes is available through the VTK user's mailing list archives. This patch is backwards compatible and has worked on all datasets tested. Included in this release are the following:

- GUISupport/Qt/QVTKWidget.cxx: Instead of deleting timers immediately, mark them for deletion next time the event loop is entered
- Hybrid/vtkAxesActor.cxx: When a user transform is applied to vtkAxesActor, make sure the axis labels are translated with the axes
- Infovis/vtkCommunity2DLayoutStrategy.cxx: Fix a memory leak
- IO/vtkFLUENTReader.cxx: Bug fixes courtesy of Terry Jordan

The first set of changes was necessary because previously this reader didn't supply ParaView with its number of cells. The second set of changes corrected Fluent files that were either errant, had an error in their writer, or had possibly experienced some change to their file formatting causing them to not supply the index within the end of binary section entries.

Other fixes included in this release are as follows:

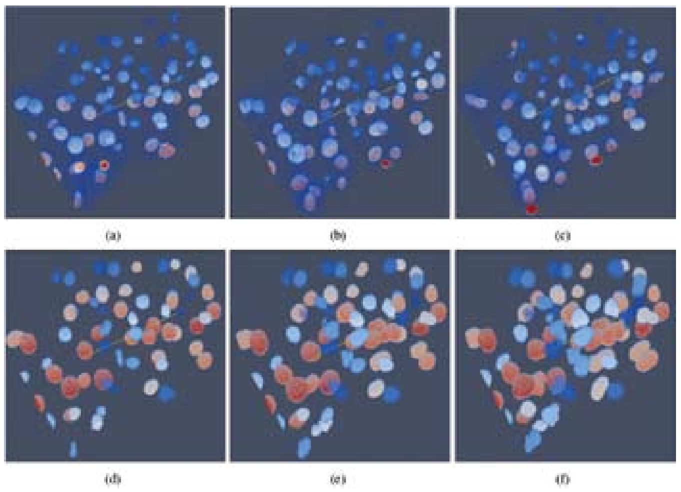
- Fixed uninitialized this->NumberOfCells. As this is used in PrintSelf. Detected by valgrind on otherPrint.
- Rendering/vtkCameraActor.cxx & Rendering/vtkLightActor.cxx: Changed vtkErrorMacro into vtkDebugMacro to avoid printing errors in the context of object instantiation. It happened because PrintSelf on the actor calls PrintSelf on superclass, which call GetBounds, which call UpdateViewProps() on the concrete classes vtkLightActor and vtkCameraActor.
- Rendering/vtkCarbonRenderWindowInteractor.cxx: Fix location for mouse events, bug #8261.
- Utilities/CMakeLists.txt: Use VTK_LIBRARY_PROPERTIES value for VERDICT_LIBRARY_PROPERTIES. Makes verdict library VERSION and SOVERSION consistent with other vtk utility libraries on Linux builds.
- Views/vtkGraphLayoutView.cxx: Check for a null pointer.
- Widgets/vtkLineRepresentation.h & Widgets/vtkLineRepresentation.cxx: Computation of parameteric coordinate, t and the closest_point, was done in display coordinates. The value t cannot directly be mapped to word coordinates, since it needs to be scaled according to the projection matrix. (The farther you go from the camera position, the larger the increments in t). To avoid this, keep everything in word coordinates by using a picker.

ITK 3.14

ITK 3.14 was released on May 28, 2009. The main changes in this release include the addition of classes contributed to the Insight Journal for performing:

- **Additional color map functors.** Based on an Insight Journal article by N. Tustison, H. Zhang, G. Lehmann, P. Yushkevich, and J. Gee, these classes provide a framework for converting intensity-valued images to user-defined RGB colormap images. Along with the framework, the classes also include several colormaps that can be readily applied for visualization of images, or adapted to generate user desired colormaps. Details are available from "Meeting Andy Warhol Somewhere Over the Rainbow: RGB Colormapping and ITK", which can be read in the January Source or on the Insight Journal (http://hdl.handle.net/1926/1452).
- **Efficient label map operations.** Based on an Insight Journal article by G. Lehmann, the added classes are the initial components of a 70+ class label map morphology module. These classes provide for the efficient representation of label maps and for conversion between current ITK label images and the efficient storage format. Details are available from "Label Object Representation and Manipulation with ITK", which can be read in the January Source or on the Insight Journal (http://hdl.handle.net/1926/584).
- **Direct FFT-based sinogram to image tomographic reconstructions.** Based on an Insight Journal article by D. Zosso, C. Bach, and J. Thiran, these classes provide the implementation of a direct Fourier method for tomographic reconstruction, applicable to parallel-beam x-ray images. Details are available from "Direct Fourier Tomographic Reconstruction Image-To-Image Filter" (http://hdl.handle.net/1926/585).

- **Fractal image dimension calculations.** Based on an Insight Journal article by N. Tustison and J. Gee, these classes provide an algorithm for converting a scalar image to a fractal dimension image. Details are available from "Stochastic Fractal Dimension Image" (http://hdl.handle.net/1926/1525).
- **Helper class for initializing BSpline Transforms.** Based on an Insight Journal Article by L. Ibáñez, M. Turek, S. Aylward, and M. Audette, this class allows a BSpline transform to be easily initialized to reasonable values prior to a registration operation. Details are available in "Helper class for initializing the grid parameters of a BSpline deformable transform by using an image as reference" (http://hdl.handle.net/1926/1338).
- **Additional level set algorithms including support for multi-component level sets.** Based on an Insight Journal article by K. Mosaliganti, B. Smith, A. Gelas, A. Gouaillard and S. Megason, these classes model an image as a constant intensity background with constant intensity foreground components. Segmentation results from a previous time point can be leveraged in processing a new time point, and real time tracking for fluorescence microscope applications can be attained. Details are available from "Cell Tracking using Coupled Active Surfaces for Nuclei and Membranes" (http://hdl.handle.net/10380/3055).



3D confocal images of a developing zebrafish embryo. (a-c) Raw images and (d-f) Tracking results at 1,5 and 10 time-points.

These classes can be found in the Code/Review Directory and can be enabled by setting the CMake variable ITK_USE_REVIEW to ON during the configuration process.

This release contains a re-factored Statistics module, found in Code/Review/Statistics, that makes the Statistics module more consistent with the rest of ITK. The use of the new module is still experimental and is disabled by default. To enable and test the new Statistics, turn ITK_USE_REVIEW_STATISTICS to ON during the configuration process.

Thanks to contributions from a number of developers, particularly Tom Vercauteren, this release offers a fix to platform rounding inconsistencies. To enable and test the new rounding functionality, turn ITK_USE_PORTABLE_ROUND to ON during the configuration process.

For more details about this release, please visit the ITK Wiki and search for "Release 3.14".

WHY AND HOW APACHE QPID CONVERTED TO CMAKE

Apache Qpid is an open source enterprise messaging system based on AMQP. I've been working on a Microsoft-funded project to convert the Apache Qpid C++ build system from the infamous GNU autotools (automake, autoconf, libtool) to CMake. In this article I'll discuss why Apache Qpid's C++ project decided to switch to CMake and how we completed the conversion. Hopefully there are some tips and techniques other projects can use in their conversions, and possibly tips or tricks you can share with us to help us improve our process.

BACKGROUND

For those who aren't familiar with how autotools work, there are two basic things a developer needs to write:

1. configure.ac: An M4-based script that gathers configuration options and examines the build system for availability and location of various capabilities, tools, files and libraries.
2. Makefile.am: A description of the build inputs and outputs. Essentially a higher-level Makefile.

These two basic areas (configuration and build items) are combined in CMake's CMakeLists.txt file, but in autotools they are separate, though very related, items.

To build in the autotools scheme one must follow these 3 steps:

1. Bootstrap the configure script using autoconf. This is done by the development team (or release engineer) and involves processing the configure.ac script into a shell script, generally named configure.
2. Configure the build for the target environment. This is done by the person building the source code, whether in development or at a user site. Configuration is carried out by executing the configure script. The script often offers command-line options for including/excluding optional parts of the product, setting the installation root, setting needed compile options, etc. The configure script also examines the build environment for the presence or absence of features and capabilities that the product can use. This step produces a config.h file that the build process uses as well as a set of Makefiles generated by automake's processing of the Makefile.am files.
3. Build, generally using the make utility.

WHY QPID SWITCHED TO CMAKE

The autotools work well, even if writing the configure scripts is a black art. People who regularly download open source programs are accustomed to the process of downloading a .tar.gz file, unpacking the source and then running configure and make. So why would Qpid want to switch? Two reasons, primarily:

1. Windows. The autotools don't work natively on Windows since there's no UNIX-like shell, and no POSIX-style make utility. Getting these capabilities involves installing MinGW. Many Windows developers, sysadmins, etc. simply won't do that.
2. Maintaining multiple build inputs (such as Makefile.am and Visual Studio projects) is an unnecessary time sink. At least one of them is always out of step. Keeping Visual Studio projects and autotools-based Makefile.am files

updated is very error-prone. Even the subset of developers that have ready access to both can get it wrong.

3. (Ok, this one is a bonus) Once you've spent enough nights trying to debug autoconf scripting, you'll do anything to get away from autotools.

We looked at a number of alternatives and settled on CMake. CMake is picking up in popular usage (KDE recently switched, and there's an effort to make Boost build with CMake as well). CMake works from its own specification of the product's source inputs and outputs, similar to autoconf, but has the following advantages:

- It also performs the "configure" step in the autotools process.
- It can generate make inputs (Visual Studio projects, Makefiles, etc.) for numerous build systems.
- It can be used to execute the test suite in addition to the build.
- It can optionally facilitate packaging as well as building, which is important for keeping the packaging information consistent and correct throughout the development process.

In the CMake world, the autotools "bootstrap" (step 1, above) is not needed. This is because rather than produce a neutral shell script for the configure step, CMake itself must be installed on each build system. This seems a bit onerous at first, but I think is better for two main reasons:

1. The configuration step in CMake lets the user view a nice graphical layout of all the options the developers offer to configure and build optional areas of the product. As the configuration happens, the display is updated to show what was learned about the environment and its capabilities. Only when all settings look correct does the user generate the build files and proceed to build the software. It takes the guesswork out of knowing if you've specified the correct configure options, or even knowing what options you have to pick from.
2. It will probably both motivate and facilitate more projects to offer pre-built binary install packages, such as RPMs and Windows installers, to help users get going quicker. One of CMake's components, CPack, helps to ease this process as well.

HOW QPID SWITCHED TO CMAKE

We started the conversion in February 2009. As of this writing, the builds have been running well for a while; the test executions are not quite done. So, it took about 3 months to get the builds running on both Linux and Windows. We're working on the testing aspects now. We have not really addressed the installation steps yet. There were only two aspects of the Qpid build conversion that weren't completely straightforward:

1. The build processes XML versions of the AMQP specification and the Qpid Management Framework specification to generate a lot of the code. The names of the generated files are not known a priori. The generator scripts produce a list of the generated files in addition to the files themselves. This list of files obviously needs to be plugged into the appropriate places when generating the makefiles.
2. There are a number of optional features (such as SSL support and clustering) which can be built into Qpid. In addition to explicitly enabling or disabling the features, the autoconf scheme checked for the requisite capabilities and enabled as many features as possible building as much as it could if the user didn't specify what to build (or not to build).

To start, one person on the team (Cliff Jansen of Interop Systems) ran the existing automake through the KDE conversion steps to get a base set of CMakeLists.txt files and did some initial prototyping for the code generation step. The original autoconf build ran the code generator at make time if the source XML specifications were available at configure time (in a release kit, the generated sources are already there, and the specs are not in the kit). The Makefile.am file then included the generated lists of sources to generate the Makefile from which the product was built. One big question we came across was where to place the code generating step in the CMake scheme. We considered two options:

- Execute the code generation in the generated Makefile (or Visual Studio project). This had the advantage of being able to leverage the build system's dependency evaluation and regenerate the code as needed. However, once generated, the Makefile (or Visual Studio project) would need to be recreated by CMake (recall that the code generation produces a list of source files that must be in the Makefile). We couldn't get this sequence to be as seamless as we had hoped.
- Execute the code generation in the CMake configuration step. This puts the dependency evaluation in the CMakeLists.txt file. Here the code regeneration had to be done by hand since we wouldn't have the build system's dependency evaluation available. However, once the code was generated, the list of generated source files was readily available for inclusion in the Makefile (and Visual Studio project) and the build could proceed smoothly.

We elected the second approach for ease of use. The CMake code for generating the AMQP specification-based code looks like this (note this code is covered by the Apache license):

```
# rubygen subdir is excluded from stable
# distributions. If the main AMQP spec is present,
# then check if ruby and python are present, and if
# any sources have changed, forcing a re-gen of
# source code.
set(AMQP_SPEC_DIR ${qpidc_SOURCE_DIR}/../specs)
set(AMQP_SPEC
  ${AMQP_SPEC_DIR}/amqp.0-10-qpid-errata.xml)
if (EXISTS ${AMQP_SPEC})
  include(FindRuby)
  include(FindPythonInterp)
  if (NOT RUBY_EXECUTABLE)
    message(FATAL_ERROR "Can't locate ruby, "
     "needed to generate source files.")
  endif (NOT RUBY_EXECUTABLE)
  if (NOT PYTHON_EXECUTABLE)
    message(FATAL_ERROR "Can't locate python, "
      "needed to generate source files.")
  endif (NOT PYTHON_EXECUTABLE)

  set(specs ${AMQP_SPEC}
    ${qpidc_SOURCE_DIR}/xml/cluster.xml)
  set(regen_amqp OFF)
  set(rgen_dir ${qpidc_SOURCE_DIR}/rubygen)
  file(GLOB_RECURSE rgen_progs ${rgen_dir}/*.rb)
# If any of the specs, or any of the sources used to
# generate code, change then regenerate the sources.
  foreach (spec_file ${specs} ${rgen_progs})
    if (${spec_file} IS_NEWER_THAN
        ${CMAKE_CURRENT_SOURCE_DIR}/rubygen.cmake)
      set(regen_amqp ON)
    endif (${spec_file} IS_NEWER_THAN
         ${CMAKE_CURRENT_SOURCE_DIR}/rubygen.cmake)
  endforeach (spec_file ${specs})
  if (regen_amqp)
    message(STATUS
      "Regenerating AMQP protocol sources")
    execute_process(
      COMMAND
        ${RUBY_EXECUTABLE} -I ${rgen_dir}
        ${rgen_dir}/generate gen ${specs} all
        ${CMAKE_CURRENT_SOURCE_DIR}/rubygen.cmake
      WORKING_DIRECTORY
        ${CMAKE_CURRENT_SOURCE_DIR})
  else (regen_amqp)
    message(STATUS "No need to generate AMQP "
                   "protocol sources")
  endif (regen_amqp)
else (EXISTS ${AMQP_SPEC})
  message(STATUS "No AMQP spec... won't "
                 "generate sources")
endif (EXISTS ${AMQP_SPEC})

# Pull in the names of the generated files,
# i.e. ${rgen_framing_srcs}
include (rubygen.cmake)
```

With the code generation issue resolved, I was able to get the rest of the project building on both Linux and Windows without much trouble. The CMake mailing list was very helpful when questions came up.

The remaining not-real-clear-for-a-newbie area was how to best handle building optional features. Where the original autoconf script tried to build as much as possible without the user specifying, I put in simpler CMake language to allow the user to select options, try the configure, and adjust settings if a feature (such as SSL libraries) was not available. This took away a convenient feature for building as much as possible without user intervention, though with CMake's ability to very easily adjust the settings and re-run the configuration step, I didn't think this was much of a loss.

Shortly after I got the first set of CMakeLists.txt files checked into the Qpid subversion repository, other team members started iterating on the initial CMake-based build. Andrew Stitcher from Red Hat quickly zeroed in on the removed capability to build as much as possible without user intervention. He developed a creative approach to setting the CMake defaults in the cache-based on some initial system checks. For example, this is the code that sets up the SSL-enabling default based on whether or not the required capability is available on the build system (note this code is covered by the Apache license):

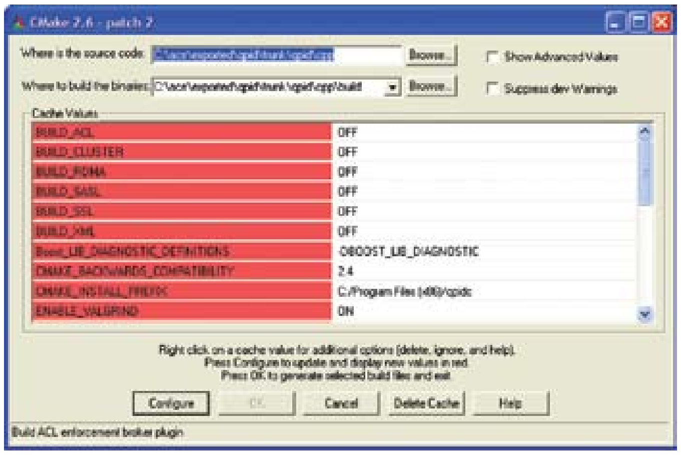
```
# Optional SSL/TLS support.
# Requires Netscape Portable Runtime.
include(FindPkgConfig)

# According to some cmake docs this is not a
# reliable way to detect pkg-configed libraries,
# but it's no worse than what we did under autotools
pkg_check_modules(NSS nss)

set (ssl_default ${ssl_force})
if (CMAKE_SYSTEM_NAME STREQUAL Windows)
else (CMAKE_SYSTEM_NAME STREQUAL Windows)
  if (NSS_FOUND)
    set (ssl_default ON)
  endif (NSS_FOUND)
endif (CMAKE_SYSTEM_NAME STREQUAL Windows)

option(BUILD_SSL "Build with support for SSL"
       ${ssl_default})
if (BUILD_SSL)

  if (NOT NSS_FOUND)
    message(FATAL_ERROR "nss/nspr not found, "
            "required for ssl support")
  endif (NOT NSS_FOUND)

  foreach(f ${NSS_CFLAGS})
    set (NSS_COMPILE_FLAGS
      "${NSS_COMPILE_FLAGS} ${f}")
  endforeach(f)

  foreach(f ${NSS_LDFLAGS})
    set (NSS_LINK_FLAGS "${NSS_LINK_FLAGS} ${f}")
  endforeach(f)

  # ... continue to set up the sources
  #     and targets to build.
endif (BUILD_SSL)
```

With that, the Apache Qpid build is going strong with CMake.

During the process I developed a pattern for naming CMake variables that play a part in user configuration and, later, in the code. There are two basic prefixes for cache variables:

- BUILD_* variables control optional features that the user can build. For example, the SSL section shown above uses BUILD_SSL. Using a common prefix, especially one that collates near the front of the alphabet, puts options that users change most often right at the top of the list, and together.
- QPID_HAS_* variables note variances about the build system that affect code but not users. For example, whether or not a particular header file or system call is available. These are passed through to compile time using the CMake configure_file statement.

As you can see from Figure 1, the settings that users would most often want to change are at the top of the list. This is a whole different experience from remembering help text (or guessing!) and typing in long command lines with the desired options.



Where is the source code: C:\acr\exported\qpid\trunk\qpid\cpp

Where to build the binaries: C:\acr\exported\qpid\trunk\qpid\cpp\build

Cache Values

BUILD_ACL	OFF
BUILD_CLUSTER	OFF
BUILD_RDMA	OFF
BUILD_SASL	OFF
BUILD_SSL	OFF
BUILD_XML	OFF
Boost_LIB_DIAGNOSTIC_DEFINITIONS	-DBOOST_LIB_DIAGNOSTIC
CMAKE_BACKWARDS_COMPATIBILITY	2.4
CMAKE_INSTALL_PREFIX	C:/Program Files (x86)/qpidc
ENABLE_VALGRIND	ON

Right click on a cache value for additional options (delete, ignore, and help).
Press Configure to update and display new values in red.
Press OK to generate selected build files and exit.

Build ACL enforcement broker plugin

Figure 1: Apache Qpid Configuration on Windows

Future efforts in the CMake area of the project will complete the transition of the test suite to CMake/CTest, which will have the side effect of making it much easier to script the regression test on Windows. The last area to be addressed will be how downstream packagers make use of the new CMake/CPack system for building RPMs, Windows installers, etc. The recently released Apache Qpid version 0.5 is the last one based on autotools and hand-maintained Visual Studio projects. The next version will be completely CMake-based. I believe this will help to improve the consistency of release results across supported platforms from build and test through to packaging.

REFERENCES

- http://qpid.apache.org
- Advanced Message Queueing Protocol (www.amqp.org)
- http://qpid.apache.org/license.html

__Steve Huston__ is a leading UNIX/Linux and Windows programming expert specializing in C++ network programming. Steve is President of Riverace Corporation and is a regular contributor to the ACE and Apache Qpid open source projects. He is co-author of C++ Network Programming (2 volumes) and The ACE Programmer's Guide. You can read Steve's blog at http://stevehuston.wordpress.com, follow him on Twitter at http://twitter.com/stevehuston, or send email to shuston@riverace.com.

PARAVIEW AND PYTHON

In this document, we will cover the basics of using ParaView's Python interface. Please note that this text is based on ParaView 3.6 and higher and is not directly applicable to earlier versions. This document only touches on the basic concepts of ParaView's Python interface. For more details and examples, see the ParaView Wiki and search for "Python Scripting" or "Python Recipes".

ParaView offers rich scripting support through Python. This support is available as part of the ParaView client (paraview), an MPI-enabled batch application (pvbatch), the ParaView python client (pvpython) or any other Python-enabled application. Using Python, users and developers can gain access to the ParaView engine called Server Manager.

GETTING STARTED

To start interacting with ParaView through Python, you have to load the paraview.simple module. This module can be loaded from any Python interpreter as long as the necessary files are in PYTHONPATH. These files are the shared libraries located in the paraview binary directory and as python modules in the paraview directory: paraview/simple.py, paraview/vtk.py, etc. You can also use either pvpython (for stand-alone or client/server execution), pvbatch (for non-interactive, distributed batch processing) or the Python shell invoked from `Tools->Python Shell` using the ParaView client to execute Python scripts. You do not have to set PYTHONPATH when using these.

In this document, I will be using the Python integrated development environment, IDLE. My PYTHONPATH is set to the following:

```
/Users/berk/work/paraview3-build/bin:/Users/berk/
work/paraview3-build/Utilities/VTKPythonWrapping
```

This is on my Mac and uses the build tree. In IDLE, let's start by loading the servermanager module.

```
>>> from paraview.simple import *
```

In this example, we will use ParaView in stand-alone mode. Connecting to a ParaView server running on a cluster is covered later.

CREATING A PIPELINE

The simple module contains many functions to instantiate readers, filters and other related objects. You can get a list of objects this module can create from www.paraview.org/OnlineHelpCurrent/.

Let's start by creating a reader object to read a VTK file.

```
>>> vtkreader = LegacyVTKReader()
```

You can get some documentation about the cone object using `help()`.

```
>>> help(vtkreader)
```

Help on LegacyVTKReader in module paraview.servermanager object:

```
class LegacyVTKReader(SourceProxy)

The Legacy VTK reader loads files stored in VTK's
legacy file format (before VTK 4.2). The expected
file extension is .vtk. The type of the dataset may
be structured grid, uniform rectilinear grid (im-
age/volume), non-uniform rectilinear grid, unstruc-
tured grid, or polygonal. This reader also supports
file series. The data descriptors are defined in
FileNames. FileNames is the list of files to be read
by the reader. If more than 1 file is specified, the
reader will switch to file series mode in which it
will pretend that it can support time and provide
1 file per time step. TimestepValues indicate the
available timestep values.
```

This gives you a full list of methods and properties (data descriptors). When using ParaView's Python interface, we interact with objects by reading and setting their properties. Let's start by setting the name of the file to be read.

```
>>> vtkreader.FileNames =
    [“.../VTKData/Data/office.binary.vtk”]
```

Note that we used a file from the VTKData repository. VTKData is a separate download from the download page on VTK.org. This file contains a uniform grid that has the numerical solution of the air flow in an office. Also note that the property is called `FileNames` rather than `FileName` and expects a Python list. This is because the VTK reader in ParaView can read a series of files that represent a time series.

Alternatively, we could have specified the file name when creating the object.

```
>>> vtkreader = = LegacyVTKReader(FileNames =
    [“.../VTKData/Data/office.binary.vtk”])
```

The constructor functions in paraview.simple all accept an arbitrary number of key,value pairs that represent property names and values.

We can read the property value back.

```
>>> vtkreader.FileNames
    ['/Users/berk/Work/VTKData/Data/office.binary.
vtk']
```

Now that we set the file name, we can access the meta-information about the data it contains. For example, we can get a list of point (node) centered data arrays by accessing the PointData property:

```
>>> vtkreader.PointData[:]
vtkFileSeriesReader : [ ........... ]
[Array: scalars, Array: vectors]
```

Note that before the function returned a value, "vtkFile-SeriesReader : [...........]" appeared on the output. This is because ParaView forced the reader to execute and read data from the file so that it can give us the metadata. The paraview.simple module forces readers and filters to execute when necessary to always produce up-to-date metadata. This file contains two arrays: scalars and vectors. We can get the range of the scalar array as follows.

```
>>> vtkreader.PointData["scalars"].GetRange()
(-3.8695590496063232, 0.71856027841567993)
```

Now that we know the range of the scalar's array, let's extract some isosurfaces.

```
>>> c = Contour(vtkreader)
>>> c.ListProperties()
['ComputeGradients', 'ComputeNormals',
 'ComputeScalars', 'Isosurfaces', 'Input',
 'ContourBy']
>>> c.ComputeScalars = True
>>> c.Isosurfaces = [-2, 0]
```

Here, we used the `ListProperties()` method to find the Isosurfaces method. We also set `ComputeScalars` on so that the output contains the contour value as a scalar.

RENDERING

Next, we will display the outline of the data as well as the isosurfaces. First, let's display the output of vtkreader as an outline.

```
>>> Show(vtkreader)
<paraview.servermanager.GeometryRepresentation
 object at 0xaf6de50>
>>> SetDisplayProperties(vtkreader,
    Representation='Outline')
```

Note that we used `SetDisplayProperties()` to set the representation type of vtkreader. We could have done the following instead.

```
>>> dp = GetDisplayProperties(vtkreader)
>>> dp.Representation = 'Outline'
```

Let's also turn on the visibility of the isosurfaces and render.

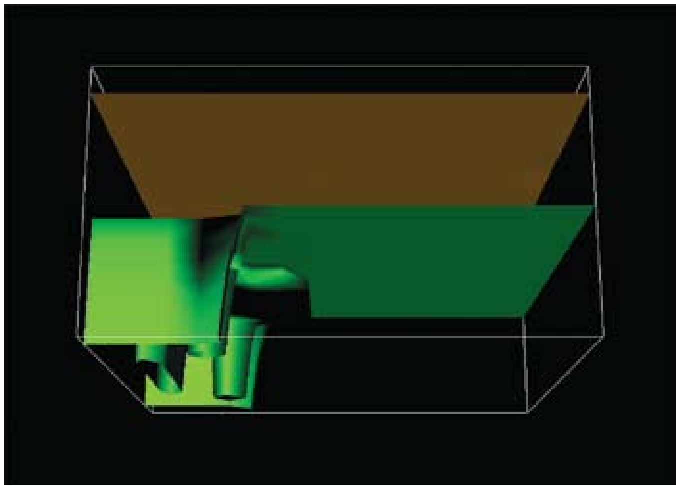
```
>>> Show(c)
vtkContourFilter : [ ........... ]
<paraview.servermanager.GeometryRepresentation
 object at 0x293110>
>>> dp.ColorArrayName = 'scalars'
>>> dp.LookupTable = MakeBlueToRedLT
(-3.8695590496063232, 0.71856027841567993)
>>> GetActiveCamera().Elevation(105)
>>> Render()
vtkPainterPolyDataMapper : [ ........... ]
<paraview.servermanager.RenderView
 object at 0x20e60030>
```

Note, we set the color of the isosurface to be mapped from the scalar's array. We also created a lookup table that is used for this mapping `(MakeBlueToRedLT)`. This lookup table interpolates color from blue to red in the given range. We also moved the camera by 105 degrees using the `Elevation()` method. Finally, `Render()` renders the scene.

We can capture this window as an image as follows.

```
>>> WriteImage("/Users/berk/test.png")
```

The resulting image is shown below.



Finally, we can save the state of ParaView to load later.

```
>>> servermanager.SaveState(".../state.pvsm")
```

USING THE PYTHON SHELL IN PARAVIEW

Let's continue our tutorial inside the ParaView user interface. Quit Python and start ParaView. First load the state file we just saved using `File->Load State`. Next, bring up the Python shell using `Tools->Python Shell`. You will notice that we no longer have access to variables we defined in our previous session (vtkreader, c and dp). So, how do we access the objects that were in the state? Paraview.simple has two convenience methods that makes this easy.

```
>>> GetSources()
{('Contour1', '1520'): <paraview.servermanager.
  Contour object at 0x24830110>,
('LegacyVTKReader1',
 '994'): <paraview.servermanager.LegacyVTKReader
 object at 0x248302b0>}
>>> c = FindSource("Contour1")
```

`GetSources()` returns a Python dictionary that has references to all pipeline objects as well as their names. `FindSource()` finds an object given its name. You can find the name of the object in the pipeline browser.

Next, let's delete the contour filter.

```
>>> Delete(c)
```

You will notice that the contour object is also removed from the pipeline browser. This is because both the user interface and the Python interpreter modify the same underlying engine and all changes made in one are immediately reflected by the other.

CLIENT/SERVER SCRIPTING

Let's now switch back to the Python shell and explore how to connect to a ParaView server. Feel free to skip this section if you are not running ParaView in client/server mode.

First, start a ParaView server (pvserver) either on the local machine or on a remote server. I ran pvserver on my desktop for this example.

You can use the `Connect()` function to connect to a server.

```
>>> from paraview.simple import *
>>> Connect("localhost", 11111)
Connection (localhost:11111)
```

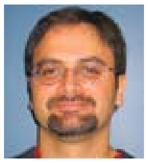
Here, the first argument is the hostname and the second argument is the port number. The easiest way to get started is to load the state we previously saved.

```
>>> servermanager.LoadState("/Users/berk/state.
pvsm")
>>> SetActiveView(GetRenderView())
```

Note that I had to set the active view manually as ParaView does not currently save the active objects in the state. We will fix this in the next release. Loading a state saved by a stand-alone ParaView application using a client/server ParaView works without problems as ParaView creates the appropriate objects under the cover. Everything discussed in this article should work without any modifications.

ACKNOWLEDGEMENTS

This work has been partially funded by Sandia National Laboratories through contract number DE-AC04-94AL85000.



__Berk Geveci__ is Kitware's Director of Scientific Computing where he leads the scientific visualization and informatics teams. He is one of the leading developers of the ParaView visualization application and the Visualization Toolkit (VTK). His research interests include large scale parallel computing, computational dynamics, finite elements and visualization algorithms.

INTRODUCING THE VISTRAILS PROVENANCE EXPLORER PLUGIN FOR PARAVIEW

In order to analyze and validate various hypotheses, it is necessary to create insightful visualizations of both the simulated processes and observed phenomena, using powerful data analysis and visualization tools like ParaView. But to explore data through visualization, scientists need to go through several steps. They need to select data products and specify series of operations that need to be applied to these data to create appropriate visual representations before they can finally view and analyze the results. Often, insight comes from comparing the results of multiple visualizations. Unfortunately, today this process is far from interactive and contains many error-prone and time-consuming tasks. As a result, the generation and maintenance of visualization data products has become a major bottleneck in the scientific process, hindering not only the ability to mine scientific data, but the actual use of scientific data in every day applications. In particular, scientists and engineers need to expend substantial effort managing data (e.g., scripts that encode computational tasks, raw data, data products, and notes) and record provenance (history) information so that basic questions can be answered, such as: Who created a data product and when? When was it modified and by whom? What was the process used to create the data product? Were two data products derived from the same raw data?

As a step toward addressing this problem, we have developed VisTrails (www.vistrails.org), an open-source, provenance-enabled scientific workflow system that can be combined with a wide range of tools, libraries, and visualization systems. VisTrails provides a comprehensive solution to the problem of managing data and processes used in data analysis and, in particular, data exploration through visualization. By capturing detailed provenance of the visualization and analysis pipelines—both of how they evolved over time and the data products they derive, VisTrails maintains documentation key to preserving data, determining the data's quality and authorship, and reproducing as well as validating results. The system also leverages provenance information to support collaboration, knowledge sharing and re-use, and it also provides intuitive interfaces that simplify common tasks in data exploration. These include: a visual interface for comparing pipelines and their results; a scalable mechanism for the exploration of large parameter spaces; and an analogy operation, which allows users to re-use pipeline refinements as a template for creating new pipelines.

The VisTrails Provenance Explorer plugin for ParaView brings provenance tracking and many of the benefits of provenance to ParaView users. The source code for the plugin can be downloaded from www.vistrails.org; it has been tested under Windows, Mac OS X, and Linux. The plugin is included with the ParaView 3.6 binaries.

USING THE VISTRAILS PROVENANCE EXPLORER PLUGIN FOR EXPLORATORY VISUALIZATION

The VisTrails Provenance Plugin for Paraview automatically and transparently tracks the steps a user followed to create a visualization. In contrast to the traditional undo/redo stack, which is cleared whenever new actions are performed, the plugin captures the complete exploration trail as a user explores different parameters and visualization techniques.

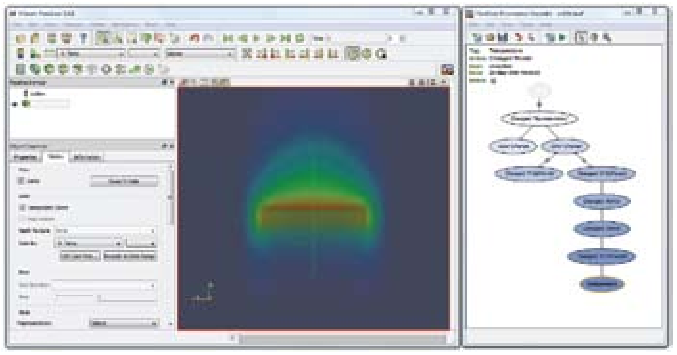


Figure 1. ParaView and the VisTrails Provenance Plugin. The Provenance Plugin (right) captures all the actions a user performs to create visualizations. These actions are stored as a tree that serves as a guide for future exploration, allowing users to return to a previous version in an intuitive way, to undo bad changes, to compare different visualizations, and to be reminded of the actions that led to a particular result.

Consider a user exploring the disk_out_ref.ex2 example dataset, derived by a simulation of the air around a heated rotating disk. The user may load the data and start by experimenting with a volume rendering of the temperature within the domain. After some time spent tweaking the visualization, he decides to examine what the temperature looks like, and labels the current version of the visualization in the Provenance Explorer as "Temperature" (see Figure 1). To look at just the air flow, he undoes all the volume actions, until the pipeline has only the data reader.

He then starts a new visualization by creating a streamline filter. After creating this new visualization and labeling it "Air Flow" (see Figure 2(b)), the user decides that a volume rendering of the pressure would help complete his view of the simulation. He would like to begin by simply reverting to the temperature visualization, and changing the scalar used for coloring to the pressure variable. Although this would not be possible using the standard undo/redo interface, because the VisTrails plugin never discards old states, the temperature visualization can easily be restored by clicking the corresponding version (node) in the Provenance Explorer window. The user can then continue modifying the volume rendering to explore the pressure data (see Figure 2(c)).

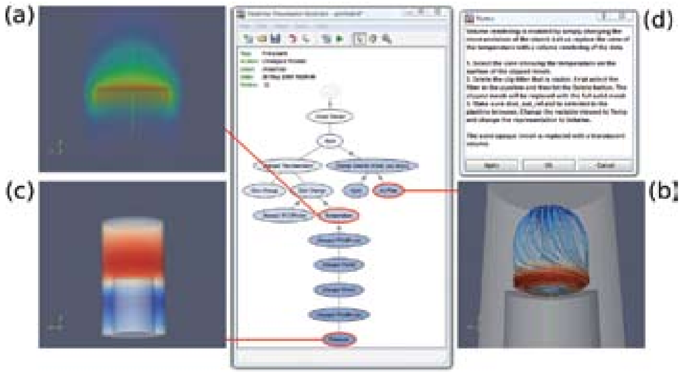


Figure 2. (a), (b) and (c) represent different visualizations generated by a user exploring the disk_out_ref.ex2 example dataset. Besides tracking all visualizations a user explored, how they evolved over time, and enabling them to be reproduced, the VisTrails Provenance Plugin captures additional metadata about the visualizations, including: a descriptive tag and notes (d), the date and time when the visualization was created, and the user who created it.

Once the different visualizations have been created by the user and recorded by the VisTrails plugin, switching between and replaying (re-generating) them to provide comparisons is fairly straightforward. This is different from the multiview visualizations supported by ParaView. The multiview mode in ParaView allows multiple instantiations of a common pipeline with different parameters. Switching between different versions in the Provenance Explorer creates the different pipelines for each version, which can reduce the clutter in the Pipeline Browser when they are significantly different. However, a disadvantage of this mode of comparison is that they cannot be viewed simultaneously side-by-side.

The VisTrails plugin has additional benefits when compared to the undo/redo framework. A tree-based view of the history of actions allows a user to return to a previous version in an intuitive way, undo bad changes, compare different visualizations, and be reminded of the actions that led to a particular result. Also, there is no limit on the number of operations that can be undone, no matter how far back in the history of the visualization they are. Last, but not least, the history is persistent across sessions. The VisTrails plugin can save all of the information needed to restore any state of the visualization in .vt files, which can be reloaded across ParaView sessions and shared among collaborators. This also allows multiple visualizations to be shared with a single file.

In addition to the capturing and replaying capabilities, the VisTrails plugin has several other features that make the full visualization provenance more manageable. Over the course of creating a visualization, many versions may be recorded by the VisTrails plugin. If the user often tries something and then reverts back to an old version, the version tree in the Provenance Explorer will also have a large number of branches, many of which will be "dead ends." To remove clutter from the window, the user can select branches to be hidden. This does not actually delete them---it just removes them from the view. The full version tree can be restored at any time. This makes browsing the version tree easier, since only the most significant versions are visible.

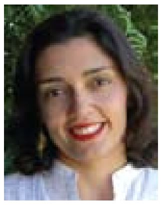
Each version recorded by the plugin has several annotations associated with it: the user that created it, the time and date that it was created, and the name of the action (see Figure 2). By default, each version is labeled in the Provenance Explorer window with the action name, as reported by ParaView. The user may additionally tag a version with a name to override this label, which makes key versions of the visualization easy to find among the others. There is also a field for notes where the user can leave comments, either for themselves, or for collaborators that they wish to share their visualizations with.

The display of the nodes of the version tree in the Provenance Explorer window also makes use of these annotations to give some immediate cues to their provenance. The versions created by the current user are colored blue to distinguish them from versions created by other users. Also, new versions use bolder coloring than old versions, making more recent changes stand out as well.

ACKNOWLEDGEMENTS

The VisTrails Provenance Explorer plugin for ParaView is the work of many talented and devoted individuals. The work is derived from the VisTrails scientific workflow middleware system. Huy Vo wrote a first version of the ParaView plugin, which was later refined and extended by Geoff Draper and John Schreiner, with substantial contributions from Tilo Ochotta and Steve Callahan. Funding has been provided by grants from the Department of Energy, the National Science Foundation, the State of Utah, and the University of Utah.

***Claudio Silva** is an Associate Professor of computer science and faculty member of the SCI Institute at the University of Utah, co-leader of the VisTrails project, and co-founder of VisTrails Inc. His research interests are in visualization and data analysis, geometry processing, and scientific data management.*



***Juliana Freire** is an Associate Professor of computer science and faculty member of the SCI Institute at the University of Utah, co-leader of the VisTrails project, and co-founder of VisTrails Inc. Her research interests are in databases, web technologies, and scientific data management.*

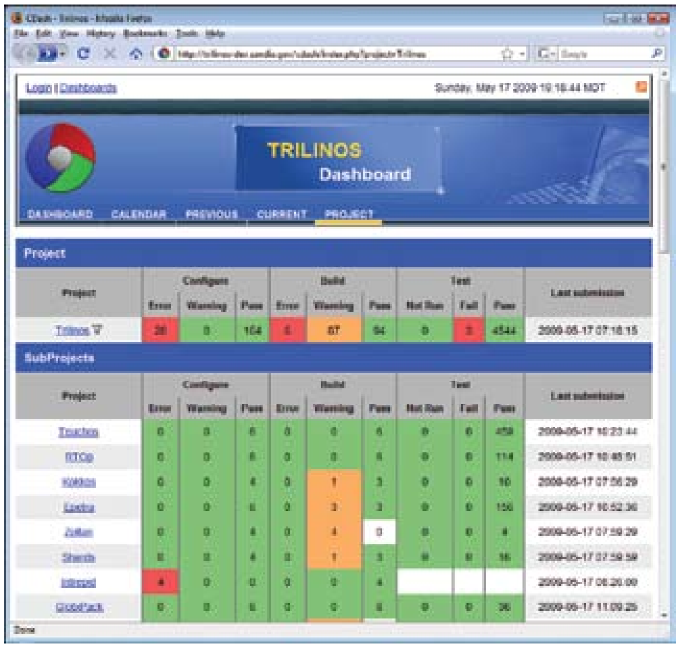
***John Schreiner** is a senior software engineer at VisTrails Inc. He is the technical lead on the VisTrails Provenance Explorer plugin for ParaView. He received a Ph.D. in computer science from the University of Utah for his thesis on geometric processing in 2008.*

CDASH SUBPROJECTS

A large project typically consists of several interdependent pieces such as libraries, executables, test suites, documentation, web pages and installers. Organizing your project into well-defined subprojects and presenting the results of nightly builds on a CDash dashboard can help you see where the problem spots are at whatever level of granularity you define. The new subproject feature of CDash (version 1.4 and later) saves you time, making it easy to identify and correct problems.

For the purposes of this article we assume you are using CDash 1.4 or later and CTest 2.7.20090520 or later (CVS HEAD of CMake from May 20, 2009, or later).

A project with subprojects has a different view for its top level CDash page than a project without any subprojects. It contains a summary row for the project as a whole and then one summary row for each subproject.



Project

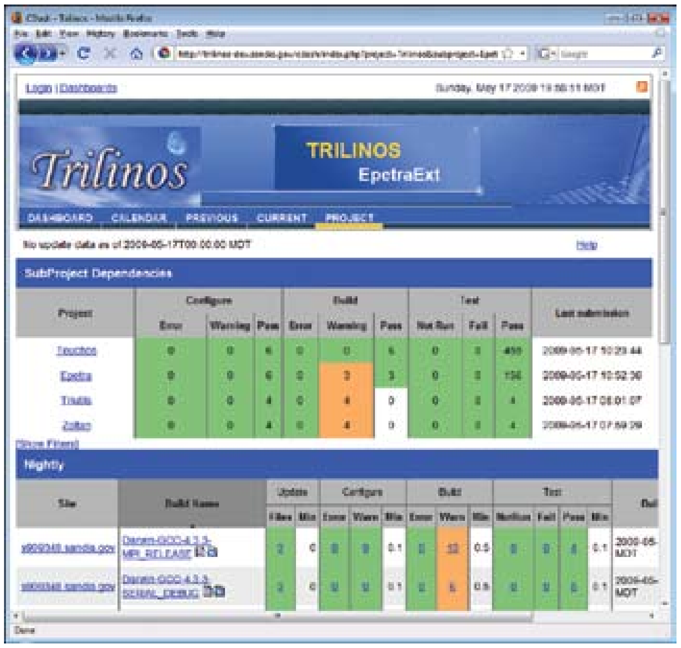
Project	Configure			Build			Test			Last submission
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass	
Trilinos	20	0	164	5	67	94	0	3	4544	2009-05-17 07:18:15

SubProjects

Project	Configure			Build			Test			Last submission
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass	
Teuchos	0	0	6	0	0	6	0	0	459	2009-05-17 10:23:44
RTOp	0	0	6	0	0	6	0	0	114	2009-05-17 10:45:51
Kokkos	0	0	4	0	1	3	0	0	10	2009-05-17 07:56:29
Epetra	0	0	6	0	3	3	0	0	156	2009-05-17 10:52:36
Zoltan	0	0	4	0	4	0	0	0	4	2009-05-17 07:59:29
Shards	0	0	4	0	1	3	0	0	16	2009-05-17 07:59:59
Intrepid	4	0	0	0	0	4				2009-05-17 08:20:00
GlobiPack	0	0	6	0	0	6	0	0	36	2009-05-17 11:09:25

The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages which are represented as subprojects on the Trilinos CDash dashboard (shown in the figure above). Explore the Trilinos dashboard to see how CDash displays projects and subprojects at trilinos-dev.sandia.gov/cdash.

The EpetraExt subproject page in the Trilinos project is an example of a subproject page with summary rows at the top (in the "SubProject Dependencies" section), as shown in the following figure.



No update data as of 2009-05-17T00:00:00 MDT

SubProject Dependencies

Project	Configure			Build			Test			Last submission
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass	
Teuchos	0	0	6	0	0	6	0	0	459	2009-05-17 10:23:44
Epetra	0	0	6	0	3	3	0	0	156	2009-05-17 10:52:36
Triutils	0	0	4	0	4	0	0	0	4	2009-05-17 08:01:07
Zoltan	0	0	4	0	4	0	0	0	4	2009-05-17 07:59:29

[Show Filters]

Nightly

Site	Build Name	Update		Configure			Build			Test				Bui
		Files	Min	Error	Warn	Min	Error	Warn	Min	NotRun	Fail	Pass	Min	
s909348.sandia.gov	Darwin-GCC-4.3.3-MPI_RELEASE	3	0	0	0	0.1	0	10	0.5	0	0	4	0.1	2009-05- MDT
s909348.sandia.gov	Darwin-GCC-4.3.3-SERIAL_DEBUG	3	0	0	0	0.1	0	6	0.5	0	0	6	0.1	2009-05- MDT

The Teuchos subproject page in the Trilinos project is an example of an independent subproject page. Since it has no subprojects that it depends on, there is no "SubProject Dependencies" section.

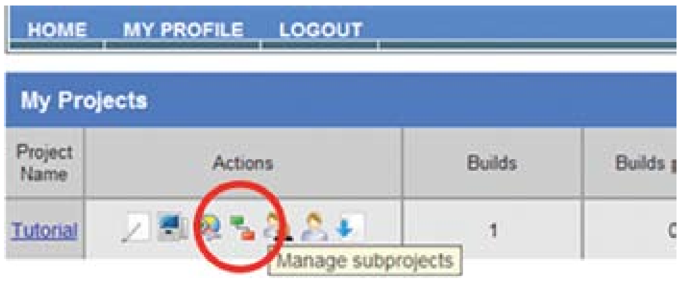
ORGANIZING AND DEFINING SUBPROJECTS

To add subproject organization into your project, you must do two things: (1) define the subprojects for CDash, so that it knows how to display them properly, and (2) use build scripts with CTest to submit subproject builds of your project. Some (re-)organizational work in your project's CMakeLists.txt files may also be necessary to allow building your project by subprojects.

There are two ways to define subprojects and their dependencies: interactively in the CDash GUI when logged in as a project administrator, or by submitting a Project.xml file describing the subprojects and dependencies.

ADDING SUBPROJECTS INTERACTIVELY

If you're a project administrator, you will have a "Manage subprojects" button for each of your projects on the My CDash page, as highlighted in the figure below. For the purpose of experimenting with the subprojects feature of CDash, create a "Tutorial" project on your CDash server and make yourself a project administrator.



Project Name	Actions	Builds	Builds
Tutorial		1	0

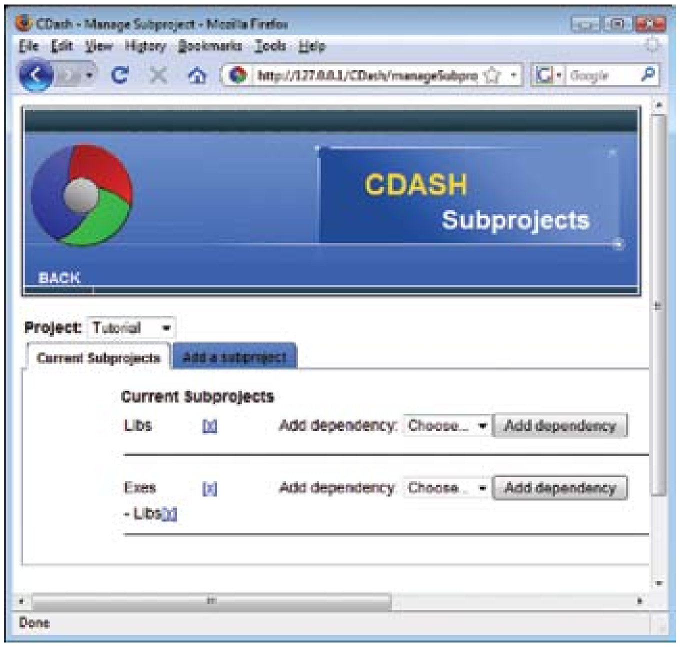
Clicking the Manage subprojects button takes you to the manage subproject page where you may add new subprojects or establish dependencies between existing subprojects. There are two tabs on this page. One for viewing the current subprojects and their dependencies and one for creating new subprojects.

When you first visit this page for a given project, it will be empty.

To add two subprojects, called Exes and Libs, and to make Exes depend on Libs:

- Click the “Add a subproject” tab
- Type “Exes” in the “Add a subproject” edit field
- Click the “Add subproject” button
- Click the “Add a subproject” tab
- Type “Libs” in the “Add a subproject” edit field
- Click the “Add Subproject” button
- In the “Exes” row of the “Current Subprojects” tab, choose “Libs” from the “Add dependency” drop down list and click the “Add dependency” button.

Now the “Current Subprojects” tab looks something like this:


CDash - Manage Subproject - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1/CDash/manageSubpro
Google
CDASH
Subprojects
BACK
Current Subprojects
Add a subproject
Done

Project: Tutorial

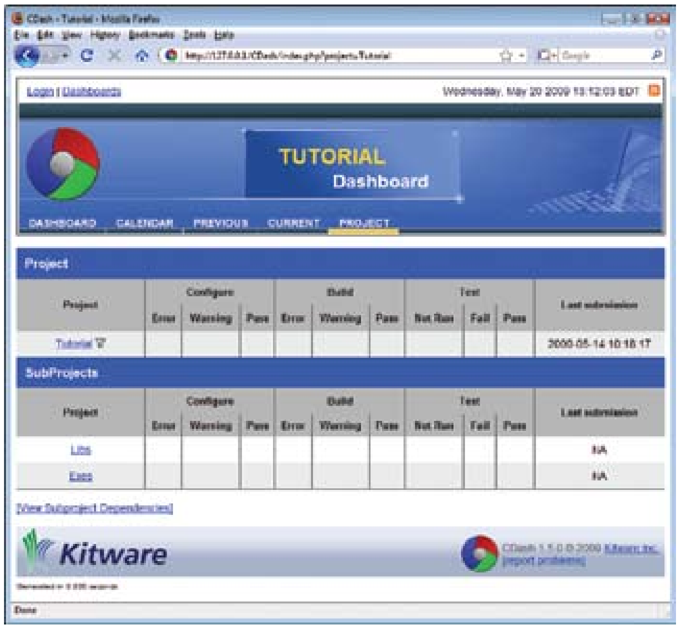
Current Subprojects

Libs [x] Add dependency: Choose... Add dependency

Exes [x] Add dependency: Choose... Add dependency
- Libs [x]

To remove a dependency or a subproject, click on the “X” next to the one you want to delete.

After adding subprojects to the Tutorial project, the main dashboard page for Tutorial looks like this:


CDash - Tutorial - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://127.0.0.1/CDash/index.php?project=Tutorial
Google
Login | Dashboards
TUTORIAL
Dashboard
DASHBOARD
CALENDAR
PREVIOUS
CURRENT
PROJECT
Kitware
CDash 1.5.0 © 2009 Kitware Inc.
[report problems]
Done

Wednesday, May 20 2009 13:12:03 EDT

Project

Project	Configure			Build			Test			Last submission
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass	
Tutorial										2009-05-14 10:18:17

SubProjects

Project	Configure			Build			Test			Last submission
	Error	Warning	Pass	Error	Warning	Pass	Not Run	Fail	Pass	
Libs										NA
Exes										NA

[View Subproject Dependencies]

ADDING SUBPROJECTS AUTOMATICALLY

The other way to define CDash subprojects and their dependencies is to submit a “Project.xml” file along with the usual submission files that CTest sends when it submits a build to CDash. To define the same two subprojects as in the interactive example above (Exes and Libs) with the same dependency (Exes depend on Libs) the Project.xml file would have contents as follows:

```
<Project name=”Tutorial”>
 <SubProject name=”Libs”>
 </SubProject>
 <SubProject name=”Exes”>
  <Dependency name=”Libs”/>
 </SubProject>
</Project>
```

Once you write or generate the Project.xml file, you can submit it to CDash from a CTest -S script using the new `FILES` argument to the `ctest_submit` command, or directly from the CTest command line in a build tree configured for dashboard submission.

From inside a CTest -S script:

```
ctest_submit(FILES ”${CTEST_BINARY_DIRECTORY}/
Project.xml”)
```

From the command line:

```
cd ../Project-build    (or wherever CTEST_BINARY_
DIRECTORY is for your build)
ctest --extra-submit Project.xml
```

CDash will automatically add subprojects and dependencies according to how you define them in the Project.xml file. It will also remove any subprojects or dependencies not defined in the Project.xml file. Or, if you submit the exact same file multiple times, the second and subsequent submissions will have no observable effect: the first submission adds/modifies the data; the second and later submissions send the same data, so no change is necessary. CDash tracks changes to the subproject definitions over time to allow for project evolution. If you view dashboards from a past date, CDash will present the project/subproject views according to the subproject definitions in effect as of that date.

Whenever a client submits a Trilinos dashboard via Trilinos/cmake/ctest/TrilinosCTestDriverCore.cmake, it automatically submits a (possibly updated) subproject definition file with these lines of the script:

```
CTEST_SUBMIT(FILES
 ”${TRILINOS_CMAKE_DIR}/python/data/
CDashSubprojectDependencies.xml”
 RETURN_VALUE SUBMIT_RETURN_VAL
 )
```

Other parts of the build process make sure that the CDashSubprojectDependencies.xml file is always up-to-date according to how the Trilinos packages are defined.

CTEST_SUBMIT PARTS AND FILES

CTest submits results to CDash with the `ctest_submit` command in CTest -S scripts. The `ctest_submit` command sends xml files describing the results of the stages of the dashboard (update, configure, build, test, …) and any note files attached by the script author.

In CTest 2.7.20090520 and later, the `ctest_submit` command supports new `PARTS` and `FILES` arguments.

With PARTS, you can send any subset of the xml files with each ctest_submit call. Previously, all PARTS would be sent with any call to ctest_submit. Typically, the script would wait until all dashboard stages were complete and then call ctest_submit once to send the results of all stages at the end of the run. Now, a script may call ctest_submit with PARTS to do partial submissions of subsets of the results. For example, you can submit configure results after ctest_configure, build results after ctest_build and test results after ctest_test. This allows for quicker posting of information as builds progress.

With FILES, you can send arbitrary xml files to CDash. In addition to the standard build result xml files that ctest sends, CDash also handles the new Project.xml file that describes subprojects and dependencies. (…described in the section "Adding Subprojects Automatically"…)

The valid PARTS values are listed in the "ctest --help-command ctest_submit" documentation:

```
ctest_submit
    Submit results to a dashboard server.

     ctest_submit([PARTS ...] [FILES ...] [RETURN_
VALUE res])

    By default all available parts are submitted if
no PARTS or FILES are specified. The PARTS option
lists a subset of parts to be submitted.
    Valid part names are:

     Start    = nothing
     Update    = ctest_update results, in Update.xml
     Configure = ctest_configure results, in
Configure.xml
     Build    = ctest_build results, in Build.xml
     Test     = ctest_test results, in Test.xml
     Coverage  = ctest_coverage results, in
Coverage.xml
     MemCheck  = ctest_memcheck results, in
DynamicAnalysis.xml
     Notes    = Files listed by CTEST_NOTES_FILES, in
Notes.xml
     ExtraFiles = Files listed by CTEST_EXTRA_
SUBMIT_FILES
     Submit    = nothing
```

The FILES option explicitly lists specific files to be submitted. Each individual file must exist at the time of the call.

Prior to the addition of the ctest_submit PARTS handling, a typical dashboard script would contain a single ctest_submit() call as its last line. Now, submissions may occur incrementally with each part of the information sent piecemeal as it becomes available:

```
ctest_start(Experimental)
ctest_update(SOURCE "${CTEST_SOURCE_DIRECTORY}")
ctest_configure(BUILD "${CTEST_BINARY_DIRECTORY}")
ctest_submit(PARTS Update Configure Notes)
ctest_build(BUILD "${CTEST_BINARY_DIRECTORY}")
ctest_submit(PARTS Build)
ctest_test(BUILD "${CTEST_BINARY_DIRECTORY}")
ctest_submit(PARTS Test)
```

Submitting incrementally by PARTS means you can inspect the results of the configure stage live on the CDash dashboard while the build is still in progress. Likewise, you can inspect the results of the build stage live while the tests are still running.

SPLITTING YOUR BUILD INTO MULTIPLE SUBPROJECT BUILDS

In a CTest -S dashboard driver script, a ctest_build call delegates to the native build tool to build the target named "${CTEST_BUILD_TARGET}". If you have not defined the CTEST_BUILD_TARGET variable in your driver script, then the build tool will build everything, using the "all" or "ALL_BUILD" target.

One ctest_build() invocation that builds everything followed by one ctest_test() invocation that tests everything is sufficient for a project that has no subprojects. In order to submit results on a per-subproject basis to CDash, you will have to make ctest_build and ctest_test calls for each subproject. To split your one large, monolithic build into many smaller subproject builds, you can use a FOREACH loop in your CTest driver script. To help you iterate over your subprojects, CDash provides a variable named CTEST_PROJECT_SUBPROJECTS in CTestConfig.cmake.

Given the above Tutorial example, CDash produces a variable like this:

```
set(CTEST_PROJECT_SUBPROJECTS
Libs
Exes
)
```

CDash orders the elements in this list such that the independent subprojects (that do not depend on any other subprojects) occur first. Then subprojects that depend only on the independent subprojects, and then subprojects that depend on those, and so on. That logic is continued until all subprojects are listed exactly once in an order that makes sense for building them incrementally.

If you organize your CMakeLists.txt files such that you have a target to build for each subproject, and you can derive the name of that target based on the subproject name, then revising your script to separate it into multiple smaller configure/build/test chunks should be relatively painless.

To facilitate building just the targets associated with a subproject, use the variable named CTEST_BUILD_TARGET to tell ctest_build what to build. To facilitate running just the tests associated with a subproject, assign the LABELS test property to your tests and use the new INCLUDE_LABEL argument to ctest_test.

To organize your monolithic build into a sequence of smaller subproject builds, make the following changes:

CMakeLists.txt modifications

- Name targets same as subprojects, base target names on subproject names, or provide a look up mechanism to map from subproject name to target name
- Possibly add custom targets to aggregate existing targets into subprojects, using add_dependencies to say which existing targets the custom target depends on
- Add LABELS target property to targets with a value of subproject name
- Add LABELS test property to tests with a value of subproject name

CTest driver script modifications

- Iterate over the subprojects in dependency order (from independent to most dependent)

- Set the "SubProject" and "Label" global properties – CTest uses these properties to submit the results to the correct subproject on the CDash server
- Build the target(s) for this subproject: compute name of target to build from subproject name, set `CTEST_BUILD_TARGET`, call `ctest_build()`
- Run the tests for this subproject using `INCLUDE` or `INCLUDE_LABEL` arguments to ctest_test`()`
- Use `ctest_submit()` with `PARTS` argument to submit partial results as they are done

To simplify the following example which demonstrates the changes required to split a build into smaller pieces, assume the subproject name is exactly the same as the target name required to build the subproject's components.

Here is a snippet from CMakeLists.txt, in the hypothetical Tutorial project. The only additions necessary (since the target and subproject names are the same) are the calls to `set_property` for each target and each test.

```
# "Libs" is the library name (therefore a target
name) *and* the subproject name
add_library(Libs …)
set_property(TARGET Libs PROPERTY LABELS Libs)
add_test(LibsTest1 …)
add_test(LibsTest2 …)
set_property(TEST LibsTest1 LibsTest2 PROPERTY
LABELS Libs)

# "Exes" is the executable name (therefore a target
name) *and* the subproject name
add_executable(Exes …)
target_link_libraries(Exes Libs)
set_property(TARGET Exes PROPERTY LABELS Exes)
add_test(ExesTest1 …)
add_test(ExesTest2 …)
set_property(TEST ExesTest1 ExesTest2 PROPERTY
LABELS Exes)
```

Here's what the CTest driver script might look like before and after organizing this project into subprojects.

Before:

```
ctest_start(Experimental)
ctest_update(SOURCE "${CTEST_SOURCE_DIRECTORY}")
ctest_configure(BUILD "${CTEST_BINARY_DIRECTORY}")
ctest_build(BUILD "${CTEST_BINARY_DIRECTORY}") #
builds *all* targets: Libs and Exes
ctest_test(BUILD "${CTEST_BINARY_DIRECTORY}") # runs
*all* tests
ctest_submit() # submits everything all at once at
the end
```

After (new chunks emphasized):

```
ctest_start(Experimental)
ctest_update(SOURCE "${CTEST_SOURCE_DIRECTORY}")
ctest_submit(PARTS Update Notes)

# to get CTEST_PROJECT_SUBPROJECTS definition:
include("${CTEST_SOURCE_DIRECTORY}/CTestConfig.
cmake")

foreach(subproject ${CTEST_PROJECT_SUBPROJECTS})
 set_property(GLOBAL PROPERTY SubProject ${subproj-
ect})
 set_property (GLOBAL PROPERTY Label ${subproject})

 ctest_configure(BUILD "${CTEST_BINARY_DIRECTORY}")
 ctest_submit(PARTS Configure)

 set(CTEST_BUILD_TARGET "${subproject}")
 ctest_build(BUILD "${CTEST_BINARY_DIRECTORY}")
  # builds target ${CTEST_BUILD_TARGET}
 ctest_submit(PARTS Build)

 ctest_test(BUILD "${CTEST_BINARY_DIRECTORY}"
  INCLUDE_LABEL "${subproject}"
 )
  # runs only tests that have a LABELS property
matching "${subproject}"
 ctest_submit(PARTS Test)
endforeach()
```

In some projects, more than one `ctest_build` step may be required to build all the pieces of the subproject. For example, in Trilinos, each subproject builds the "${subproject}_libs" target and then builds the "all" target to build all the configured executables in the test suite. They also configure things such that only the executables that need to build for the currently configured packages build when the "all" target is built.

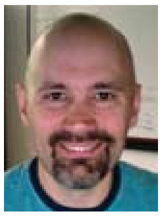
Normally, if you submit multiple Build.xml files to CDash with the same exact build stamp, it will delete the existing entry and add the new entry in its place. In the case where multiple `ctest_build` steps are required, each with their own `ctest_submit(PARTS` Build) call, use the "APPEND" keyword argument in all of the `ctest_build` calls that belong together. That APPEND flag tells CDash to accumulate the results from multiple Build.xml submissions and display the aggregation of all of them in one row on the dashboard. From CDash's perspective, multiple `ctest_build` calls (with the same build stamp and subproject and APPEND turned on) result in a single CDash buildid.

GO FORTH AND CODE!

Go ahead and adopt some of these tips and techniques in your favorite CMake-based project.

- LABELS is a new CMake/CTest property that applies to source files, targets and tests. Labels are sent to CDash inside the result xml files.
- Use ctest_submit(PARTS …) to do incremental submissions. Results are available for viewing on the dashboards sooner.
- Use INCLUDE_LABEL with ctest_test to run only the tests with labels that match the regular expression.
- Use CTEST_BUILD_TARGET to build your subprojects one at a time, submitting subproject dashboards along the way.

If you follow the guidelines in this article and split your favorite CMake-based project into subprojects, let us know about it on the CMake and CDash mailing lists. If you run into problems along the way, post questions to the mailing lists which are the principal means of communication among developers and users.

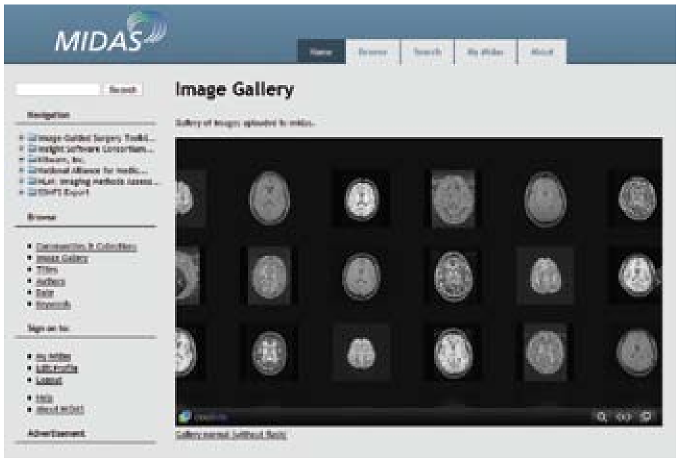


David Cole *is an R&D Engineer in Kitware's Clifton Park, NY office. David has contributed code to the VTK, CMake, ITK, ParaView, KWWidgets and gccxml open-source projects. He has also contributed to Kitware's proprietary products, including ActiViz and VolView.*

KITWARE NEWS

MIDAS 2.2

MIDAS, Kitware's digital archiving and distributed processing system, has seen its second major release in a year. MIDAS collects, manages and process digital media.



Among the new features added are:

- Better policies management
- Improved navigation
- Faster database access and rendering
- New DICOM metadata extraction and search
- Improved security for private collections
- Faster search
- LDAP support
- Improved image gallery
- Initial support for custom workflow
- New web services API
- Redesigned grid computing management

Kitware's public instance of MIDAS is available at http://insight-journal.org/midas, and is host to hundreds of freely available scientific and medical datasets.

For more information about MIDAS and to download a free trial version, visit www.kitware.com/midas.

2ND ANNUAL BEST BIOMEDICAL VISUALIZATION INVOLVING ITK OR VTK AT MICCAI 2009

Kitware is hosting its 2nd Annual Best Biomedical Visualization Contest in conjunction with MICCAI 2009, the 12th International Conference on Medical Image Computing and Computer Assisted Intervention, which will be held in London from September 20-24.

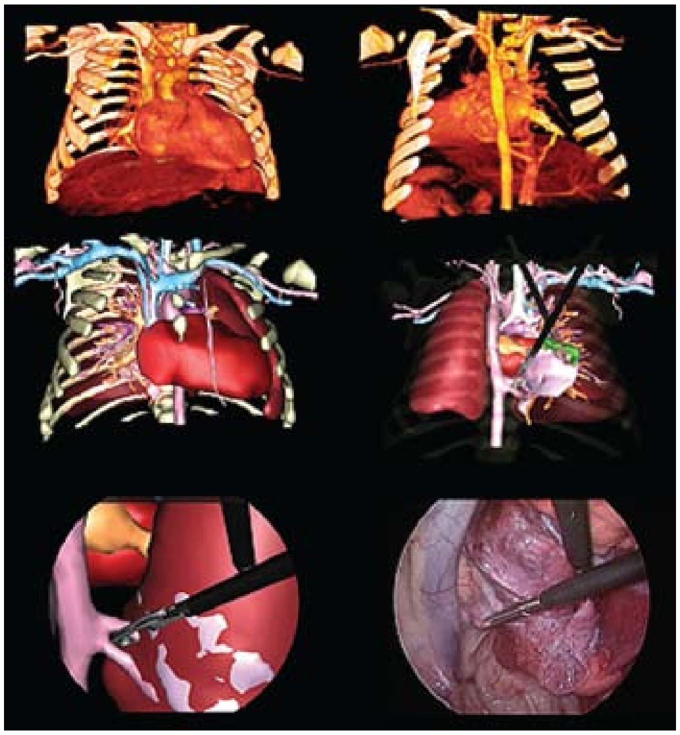
The contest, which is open to the public, is being hosted at: public.kitware.com/ImageVote. Users may submit visualizations and cast votes after registering, for free, at the contest website. Visualizations may be charts, graphs, photographs, or renderings from the biomedical field.

All images must have been produced by programs that use ITK and/or VTK. To that effect we've generated a list of applications on the contest website and kitware.com which use ITK and VTK and are therefore eligible.

We have enlisted a panel of expert judges to help determine the winners. Entries will be judged based on scientific significance, aesthetics, and use of ITK or VTK. The winners will be announced during the conference's awards session.

- First-place prize is $750 and a Kitware shirt or hat.
- Second-place prize is $250 and a Kitware shirt or hat.

Submissions must be received by 5:00 PM EDT on September 1st, 2009. *Please note that images generated by Kitware employees are not eligible for this contest.*



2008 winner, Luc Soler from IRCAD

LATE SUMMER/FALL CONFERENCES AND EVENTS

If you're interested in meeting with a Kitware representative at one of these events, email us at kitware@kitware.com.

IEEE Cluster 2009

August 31-September 4, 2009, in New Orleans, LA. Kitware, along with Sandia National Labs, will be presenting a half day tutorial on Parallel Distributed-Memory Visualization with ParaView". The tutorial will take place on Monday, August 31. http://www.cluster2009.org

Euromech Solid Mechanics Conference 2009

September 7-11, 2009, in Lisbon Portugal. Michel Audette of Kitware is cochairing the symposium "Image Processing and Visualization in Solid Mechanics Processes" along with João Manuel R. S. Tavares of the University of Porto, Portugal. http://www.dem.ist.utl.pt/esmc2009

MICCAI 2009

September 20-24, 2009, at the Imperial College, London, UK. Kitware is co-organizing a workshop on "Systems and Architectures for Computer Assisted Interventions". In addition, Kitware is providing the MIDAS Journal to all workshop organizers to assist with publication management and we are hosting the 2nd Annual Best Biomedical Visualization Contest in conjunction with the conference. http://www.miccai2009.org

ICCV 2009
September 27-October 4, 2009, in Kyoto, Japan. Arslan Basharat of Kitware and Dr. Mubarak Shah from the University of Central Florida will be presenting a paper on "Time Series Prediction by Chaotic Modeling of Nonlinear Dynamical Systems". http://www.iccv2009.org

IEEE Vis 2009
October 11-16, 2009, in Atlantic City, NJ. Kitware is a Gold Level Sponsor for this conference. We have also submitted multiple tutorial proposals and are awaiting acceptance. Kitware will also be holding a Birds of a Feather Session at the conference. http://vis.computer.org/VisWeek2009

Supercomputing 2009
November 14-20, 2009, in Portland, OR. Kitware, along with Sandia National Labs, will be presenting a half-day tutorial on "Large Scale Visualization with ParaView". The tutorial will take place on Sunday, November 15. http://sc09.supercomputing.org

IGSTK EUROPEAN USER'S MEETING
The IGSTK development team hosted a European User's meeting at the Computer Assisted Radiology and Surgery conference in Berlin, Germany on June 23, 2009. The invitation-only meeting drew 35 participants from 11 countries. The meeting started with a short introduction and update on new developments in the 4.2 release.

This was followed by presentations on IGSTK-based applications from the user community. The following members of the user community presented:

- Silesian University of Technology, Biomedical Division, Poland
- Medizinische Universität Innsbruck, Austria
- ICCAS, University of Leipzig, Germany
- Princess Margaret Hospital, Toronto, Canada
- University Hospital of Geneva, Switzerland (OsiriX)
- SINTEF, Department of Medical Technology, Trondheim, Norway

At the end of the meeting, an application demonstration was presented by tracking device vendors NDI and Claron Technology. The demonstration exhibited an IGSTK-based needle biopsy application using their tracking device.

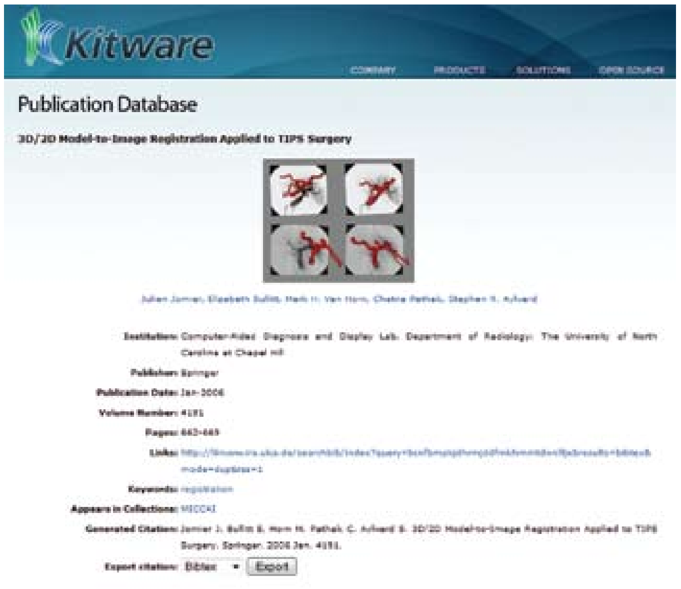
For abstracts and presentation slides for this event, please visit the IGSTK Wiki and search for "European Users Meeting".

For more details on this and other User's Group Meetings please visit: http://igstk.org/IGSTK/help/meetings.html.

KITWARE WINS DOE PHASE I SBIR
Kitware has been awarded a Phase I SBIR from the Department of Energy entitled "Multi-Resolution Streaming for Remote Scalable Visualization." This project expands upon Kitware's expertise in Distance Visualization, which is becoming a critical area of research now that scientists and the massive amounts of data processed on supercomputers are often in two or more geographically separate locations. The goal of this Phase I award is to demonstrate a system that allows computational results to remain on the supercomputing system, while using a client-server architecture to process, analyze, and visualize the data. In order to achieve this goal, we will employ multi-resolution visualization methods that only load and display blocks of data, at the appropriate resolution, that are necessary for a particular visualization task.

PUBLICATION DATABASE 1.2
In conjunction with the overhauling of the MIDAS 2.2 framework, the Publication Database version 1.2 was also released this month. The Publication Database collects, manages and disseminates publications. Thanks to automatic import tools established in this release, importing papers from and searching for publications submitted by your institution has never been easier.

The Publication Database is based on open standards and ensures that submitted publications are referenced by major search engines, thus greatly improving the dissemination of current research and the publishing activities completed by your institution. The system also allows dynamically exporting, via web services, the content of the database to external websites, for instance listing the most recent publications for a user.



Publication Database

3D/2D Model-to-Image Registration Applied to TIPS Surgery

Julien Jomier, Elizabeth Bullitt, Mark H. Van Horn, Chetna Pathak, Stephen R. Aylward

Institution: Computer-Aided Diagnosis and Display Lab. Department of Radiology. The University of North Carolina at Chapel Hill

Publisher: Springer

Publication Date: Jan-2006

Volume Number: 4191

Pages: 662-669

Links: [illegible]

Keywords: registration

Appears in Collections: MICCAI

Generated Citation: Jomier J, Bullitt E, Horn M, Pathak C, Aylward S. 3D/2D Model-to-Image Registration Applied to TIPS Surgery. Springer. 2006 Jan. 4191.

Export citation: Bibtex Export

The Publication Database is currently used by the Surgical Planning Laboratory at Harvard: http://www.spl.harvard.edu/publications and a new instance referencing publications completed by Kitware employees is available at http://www.kitware.com/publications.

To install an instance of the Publication Database at your site, please visit http://www.kitware.com/midas.

NEW HIRES

Ann D'Alessio
Ann joined Kitware in April 2009 as a Contract Specialist where she works with Kitware's Contracts Administrator processing proposals and contracts. Prior to joining Kitware Ann worked as a Paralegal for litigation and estate planning law firms. She received an AAS in Paralegal Studies from Schenectady County Community College.

Christopher Greco
Christopher joined Kitware in April 2009 where he is currently involved as an R&D Engineer for the Computer Vision team. Prior to joining Kitware, Christopher worked in the Robotic Vision Lab at BYU while completing his BS and MS in Electrical Engineering, and then as a contractor with the Visualization and Computer Vision lab at the GE Global Research Center.

NEW INTERNS

Sophie Chen
Sophie joined Kitware in May 2009 as an intern for the Insight Toolkit (ITK). Sophie attends Rensselaer Polytechnic Institute where she's working toward completing her BS in Information Technology with a concentration in Pre-Law and a minor in Management and Technology.

Chris Gardiner
Chris joined Kitware in June 2009 as a graphic design intern. Chris attends the College of Saint Rose where he's working toward completing his BFA in Graphic Design.

John Jalbert
John joined Kitware as a Computer Vision intern in May 2009. John attends Rensselaer Polytechnic Institute, where he's working toward completing his dual degree in Electrical and Computer Systems Engineering.

Adrien Bailly
Adrien joined Kitware in July of 2009 as an intern at the Chapel Hill, NC office where he is working on the MIDAS framework. Adrien will be interning with Kitware for a year as part of his academic program the ESCPE-Lyon in France where he studied Computer Science and Signal Processing.

Mikael Le Gall
Mikael joined Kitware in July of 2009 as an intern at the Chapel Hill, NC office where he's working on the MIDAS framework. Mikael will be interning with Kitware for a year as part of his academic program the ESCPE-Lyon in France where he studied Computer Science and Signal Processing.

Nikhil Shetty
Nikhil joined Kitware in May 2009 where he's working as an intern on the ParaView toolkit. Nikhil is a PhD candidate from the University of Louisiana at Lafayette, where he completed his MS in Computer Science. He has a BTech degree in Computer Science and IT from Jawaharlal Nehru Technological University in India.

INTERNSHIP OPPORTUNITIES

Kitware Internships provide current college students with the opportunity to gain hands-on experience working with leaders in their fields on cutting edge problems. Our business model is based on open source software—an exciting, rewarding work environment.

At Kitware you will assist in the development of foundational research and leading-edge technology across our five business areas: supercomputing visualization, computer vision, medical imaging, data publishing and quality software process. We offer our interns a challenging work environment and the opportunity to acquire advanced software training. Apply by sending your resume to internships@kitware.com.

EMPLOYMENT OPPORTUNITIES

Kitware is seeking qualified applicants to work with leaders in the fields of computer vision, medical imaging, visualization, 3D data publishing and technical software development.

We offer comprehensive benefits including: flex hours; six weeks paid time off; a computer hardware budget; 401(k); health and life insurance; short- and long-term disability, visa processing; a generous compensation plan; profit sharing; and free drinks and snacks. Interested applicants should forward a cover letter and resume to jobs@kitware.com.

In addition to providing readers with updates on Kitware product development and news pertinent to the open source community, the *Kitware Source* delivers basic information on recent releases, upcoming changes and detailed technical articles related to Kitware's open-source projects. These include:

- The Visualization Toolkit (www.vtk.org)
- The Insight Segmentation and Registration Toolkit (www.itk.org)
- ParaView (www.paraview.org)
- The Image Guided Surgery Toolkit (www.igstk.org)
- CMake (www.cmake.org)
- CDash (www.cdash.org)
- KWWidgets (www.kwwidgets.org)
- BatchMake (www.batchmake.org)
- VTKEdge (www.vtkedge.org)

Kitware would like to encourage our active developer community to contribute to the *Source*. Contributions may include a technical article describing an enhancement you've made to a Kitware open-source project or successes/lessons learned via developing a product built upon one or more of Kitware's open-source projects. Authors of any accepted article will receive a free, five volume set of Kitware books.

Kitware's Software Developer's Quarterly is published by Kitware, Inc., Clifton Park, New York.

Contributors: Lisa Avila, Utkarsh Ayachit, Stephen Aylward, Jeff Baumes, David Cole, Katie Cronen, David DeMarle, Andinet Enquobahrie, Juliana Freire, Zack Galbreath, Chris Gardiner, Berk Geveci, Bill Hoffman, Steve Huston, Luis Ibàñez, Julien Jomier, Niki Russell, John Schreiner, Will Schroeder, Claudio Silva and Wes Turner.

Design: Melissa Kingman, www.elevationda.com

Editor: Niki Russell

To contribute to Kitware's open source dialogue in future editions, or for more information on contributing to specific projects, please contact the editor at kitware@kitware.com.