FlowAnalyzer Documentation

Götz Ruprecht

FlowAnalyzer Documentation

Götz Ruprecht
For the Linux/Gtk FlowComposer and many other contributions: Lars Martin

Table of Contents

List of Figures

Chapter 1. Installation

Supported systems

There is a base package comprising a base flow kit, an AIDA (histogramming) kit, and the command line tool "flowshell" that runs a pflow file. Even though the histogramming system can be any AIDA conform dynamic library, there is currently only one that works on different platforms: OpenScientist. You need to install that package first if you want to do any histogramming. If you don't need to do any histogramming you are done by just downloading the base package.

We usually test the base package for *Ubuntu Linux*, *Scientific Linux*, and *Mac OS X*. For these systems we can provide binary distributions that are easy to install. For other systems the binary distribtion might work. If not, it is very likely that you can build it from the sources if your system is POSIX conform.

Prerequisites

You need at least version *16.10* of the *osc_vis* package of OpenScientist [http://openscientist.lal.in2p3.fr] if you want to do any kind of graphical (and non-graphical) histogramming. You can use the direct download link from the table below.

OpenScientist is a multi-GUI system. It can use Qt, GTK, XMotif, and more systems for the graphical output. The default is XMotif but there are 2 good reasons to use Qt:

- Qt is more advanced and has a better look and feel.
- On Mac OS X, Qt is not using the X windows layer, so it integrates and runs much better with Cocoa applications.

We strongly suggest the installation of Nokias Qt [http://www.qtsoftware.com] and provide only suppport for the combinations given in the table below. We recommend version 4.5.2 and higher. You can try older versions but qt4 is absolutely mandatory, so check your package version if Qt is already on your system and upgrade if necessary. For the binary package you can click on the direct download link below or use the package installer of your Linux distro.

Note

Problems with OSC using Qt have been reported for Gentoo Linux. This is no surprise because Gentoo relies on qt3 that was discontinued 5 years ago. You must compile qt4 on your own.

Installation of the binary

Note for Mac users: If you intend to use the graphical tool FlowComposer anyway, there is no need to download the base flow kit; it is already in there. Install Qt and OpenScientist and skip directly to the FlowComposer chapter.

Here is a list of the binary packages with direct download links (OSC = OpenScientist). Qt and OSC are foreign packages, PluginFlow is our package with the basic plugins, flowshell, and some OSC conform custom plotters.

System	Tested with	Qt	OSC	PluginFlow
Ubuntu Linux	9.04	libqt4-opengl [apt:libqt4-opengl]	x86 [http://openscientist.lal.in2p3.fr/go]	x86 [http://tactic.triumf.ca/]

System	Tested with	Qt	OSC	PluginFlow
Mandriva Linux	2009 Spring	libqtopengl4, use installer	download/16.10/osc_vis-16.10-Linux-i386-gcc_424.zip], x86-64(unoff.) [http://tactic.triumf.ca/software/download/Linux64/osc_vis-16.10-Linux-x86_64-gcc_413.zip]	software/download/Linux32/PluginFlow.zip], x86-64 [http://tactic.triumf.ca/software/download/Linux64/PluginFlow.zip]
Gentoo Linux		emerge x11-libs/qt-opengl		
Linux SL	xxxx	Better use the Qt SDK 32bit [http://qt.nokia.com/downloads/sdk-linux-x11-32bit-cpp]/64bit [http://qt.nokia.com/downloads/sdk-linux-x11-64bit-cpp]		
MacOS X	10.4, 10.5	32bit Universal [http://www.qtsoftware.com/downloads/mac-os-cpp]	x86 only [http://openscientist.lal.in2p3.fr/download/16.10/osc_vis-16.10-Darwin-i386-gcc_401.zip]	32bit Universal [http://tactic.triumf.ca/software/download/Mac/PluginFlow.zip]

Install Qt as the package manager or the installer suggests. OpenScientist comes as a ZIP package. Unpack it, move it to any desired directory but *do* not *install it yet*. *If you are a Mac user* and want to use the graphical tool *FlowComposer only*, the following steps are not necessary. Proceed with the FlowComposer instructions instead.

Now download our "PluginFlow" package and unzip it to your favourite packages folder (This will create a "PluginFlow" folder).

Note

In the following we assume that you are familiar with some basics of UNIX like "shell", "environment variables", etc. We also assume that you are running the bourne shell "bash". The procedure is slightly different for a different shell. For bash, there exist 2 initialization files in the users home directory, ".bashrc" if bash is just started as a command, and ".bash_profile" for a (local or remote) login shell. Make sure you add your settings to the right file. On Mac OS X ".bash_profile" is almost always the right file. On Linux, one usually (or at least often) loads ("sources") the other anyway. You could also set your terminal program to always start a login shell and use ".bash_profile". As a placeholder we will in the following always say ".bash_profile".

Qt configuration

On Linux systems, qt usually installs in the default library search path, so nothing is to do. If you installed it to a different location, add the path of the "qt/lib" directory in your installation to LD_LIBRARY_PATH. *On Mac OS X*, we need to extend the library search path. To your .bash_profile add the lines

```
# Qt settings
export Qt_home=/Library/Frameworks
export DYLD_LIBRARY_PATH=$Qt_home/QtNetwork.framework:$Qt_home/QtG
```

```
$Qt_home/QtOpenGL.framework:$Qt_home/QtSql.framework:\
$Qt_home/QtXml.framework:$Qt_home/QtCore.framework:$DYLD_LIBRARY_P
```

PluginFlow configuration

Add the following lines to your .bash_profile

```
# FlowAnalyzer settings
export PLF_BASE=<path to your PluginFlow folder, e.g. /home/lars/p
export LD_LIBRARY_PATH=$PLF_BASE/Controller:$PLF_BASE/Base:$PLF_BA
export DYLD_LIBRARY_PATH=$PLF_BASE/Controller:$PLF_BASE/Base:$PLF_
export PATH=$PLF_BASE/Controller:$PATH
```

Note: On Mac OS X, the line with "LD_LIBRARY_PATH" can be left out, on other systems the line with "DYLD_LIBRARY_PATH" can be dismissed.

Another note:: If you installed FlowComposer (Mac), your PLF_BASE path is: PLF_BASE=<*path to your FlowComposer folder*>/FlowComposer.app/Contents/Resources/PLFBase

OpenScientist configuration

"Cd" to the OSC version directory (".../osc_vis/16.10") and type "./install". Note that once you called "./install" you can not move the directory to another location anymore without breaking the installation. Add the following lines to your .bash_profile:

```
# OpenScientist settings
source <path to your OSC version, e.g. /home/lars/packages/osc_vis
export ONXLAB_PLOTTER_ROOT=$PLF_BASE/GUIs
export ONXLAB_PLOTTER=$ONXLAB_PLOTTER_ROOT/BasicPlotter.onx
export ONX_ARGS=-qt
export LD_LIBRARY_PATH=$PLF_BASE/DLDs:$LD_LIBRARY_PATH
export DYLD_LIBRARY_PATH=$PLF_BASE/DLDs:$DYLD_LIBRARY_PATH
```

Note: On Mac OS X, the line with "LD_LIBRARY_PATH" can be left out, on other systems the line with "DYLD_LIBRARY_PATH" can be dismissed.

The installation instructions on the OpenScientist website [http://openscientist.lal.in2p3.fr/osc_web_16.8/16.8/html/download.html#download_install] are for version 16.8 and might fail for 16.10.

Run and test

Start a new shell to update to the new .bash_profile settings. Enter `osc-plot-qt`. This should start an OSC application with the default plotter where you can learn OSC and dig through all examples. If it works, you successfully installed OSC (and Qt). If not, your installation failed and you can go home.

Now start the flowshell by entering `flowshell`. Since no pflow filename is given, flowshell automatically looks for $PLF_BASE/Base/gaussian.pflow. If it works, you see numbers on the screen for a few seconds and the last line should be "Delete manager done". Next try to run the simple AIDA analyzer that comes with the package. Enter `flowshell -i -t -p OnXLabLoadAIDA -f`

`$PLF_BASE/Aida/simplehisto.pflow`. This should open a fully interactive OSC plotter with a histogram in it being filled with gaussian distributed numbers. Play with the plotter and finish it with the menu "File->Abandon Control". The last message in the terminal should be again "Delete manager done". *Congratulations, all parts of the FlowAnalyzer are running!*

You can now go ahead and download any desired FlowKits.

Source distributions / CVS

Compiling from the sources is a bit more demanding and not recommended unless you need to.

FlowKits

Use`make dist`to build package.

- The base package: *PluginFlow* (CVS [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/PluginFlow], tarball [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/PluginFlow/?view=tar])

  ### Note

  You need to set $PLF_BASE to the source PluginFlow folder. If you do `make dist` the package also expects $ONXFILES to be set to the OnX files folder (which can be again the binary or the compiled source [below]). We also suggest, if you also want to *use* PluginFlow from the source folder, that you link $ONXFILES/DLDs and $ONXFILES/GUIs into the PluginFlow source folder.

- For MIDAS [http://midas.triumf.ca]: *MidasFlow* (CVS [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/MidasFlow], tarball [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/MidasFlow/?view=tar]).

  MidasFile requires ROOTANA [http://ladd00.triumf.ca/~olchansk/rootana] (can be compiled without ROOT), MidasOnline requires MIDAS [http://midas.triumf.ca].

- For TACTIC: *TacticFlow* (CVS [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/TacticFlow], tarball [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/TacticFlow/?view=tar]).

- For UK MIDAS (used for TUDA [http://tuda.triumf.ca]): *UKMidasFlow* (CVS [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/UKMidasFlow], tarball [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/UKMidasFlow/?view=tar]).

You will need headers and libraries from several different packages, such as Qt4, libXslt and libGLU. Others may be needed depending on your system.

OnX files

- The GUI in OpenScientist is defined by XML-based files (OnX files) that can be customized by the application developer, independent from the actual graphics system (Qt, GTK, Motif...). For the Flow Analyzer we have OnX files for different purposes; they are collected in a CVS together with callback functions.

  Here is the CVS [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/OnXFiles] and the tarball [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/OnXFiles/?view=tar]

Chapter 2. Introduction

General

When data from an experiment are produced they usually go through several steps of transportation, reformatting, grouping, filtering, and so on, until they end up in some kind of visualization, histograms, curves, tracks, etc. Many of these steps are common to many experiments, others are very specific to only one particular setup. For the pure transportation, highly developed frameworks like MIDAS [http://midas.triumf.ca], usually called the "data acquisition" (DAQ), can be used but for the higher stages, usually referred to as the "analysis", no common solution can be found.

The leitmotiv of the Flow Analyzer is that as much as possible binary code analyzing data from a physical experiment can be re-used and not more than the code absolutely necessary for a particular analysis must be implemented. This means that the main program provides only a basic functionality and the actual analyzer is composed during the run time. Techniques like dynamic loading and linking or symbol lookup are indispensible to achieve this flexibility.

The idea dates back a few years when Jonty Pearson was upgrading TRIUMFs DAQ for the DRAGON experiment and started to separate the part of the analyzer program that never changes (start/stop, skip events, etc...) from the specific part, the "Plugin", for a particular experiment. The analyzer dynamically loads and links the plugin into its own running code. The coding was done in C++, and one plugin in principle represented one C++ object class, though there was only one single instantiation of that class and only one type of class (the analyzer class).

Soonafter, we decided for a more rigorous concept that even parts of the analyzer plugin are plugins, and that the entire analyzer composes itself from that parts when it starts up. These components ("modules", or also "plugins") could have defined interfaces that can be found by dynamic symbol lookup, so there would be no loss in speed compared with a custom compiled ("old-fashioned") analyzer.

So far, none of those concept came to an application for DRAGON but I picked up the code for the "single plugin" analyzer for the TACTIC experiment and developed it. Late 2008 there have been quite a few people from the TACTIC group working on different parts of the analysis and the "single plugin" concept became very complicated to manage. I started to develop also the "multi plugin" concept for TACTIC as described above and January 2009 we used it first time for a real experiment with great success

How does it work?

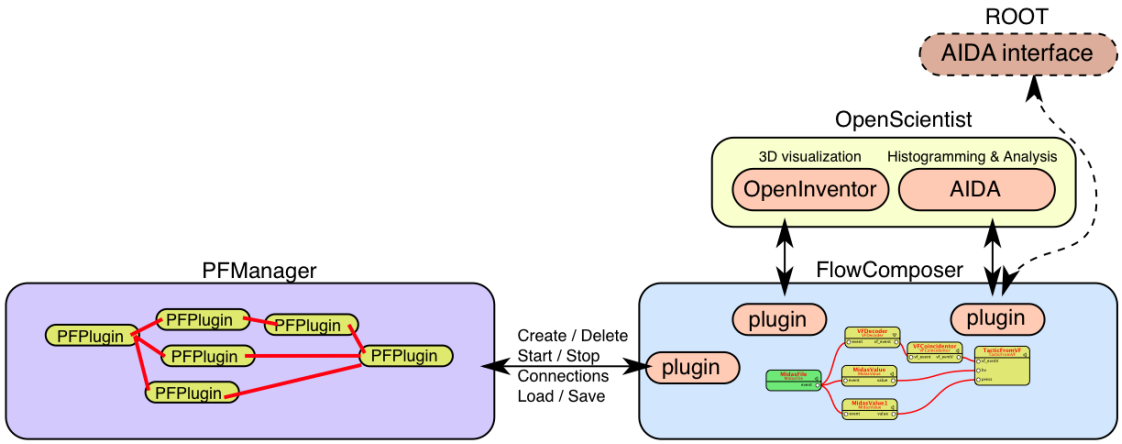
Soon after using the first versions of a multi-plugin analyzer there turned out to be a problem with the graphical display. Several instances of the same class behave the same way, so if a histogram plugin opens a window with a graphical representation of that histogram, another one does the same, and soon the screen is full with flying windows that belong to any plugins. In the same context there was also the problem that one histogram class is usually connected to one type of histogram, so you need different classes for ROOT [http://root.cern.ch] histograms or other systems (like custom made histograms or non-graphical histograms). The solution was AIDA [http://aida.freehep.org], an abstract histogramming (and analysis) interface.

AIDA [http://aida.freehep.org] solves both problems and has even more advantages. AIDA [http://aida.freehep.org] defines a set of classes and functions that can be used for histogramming data, fitting, etc. but there can be different implementations. This means that without changing the histogram plugin in the analyzer, the complete histogramming system can be exchanged, so the analyzer is not tied to any particular system like ROOT. And the implementations usually group the histograms in a single window with a tree

browser, so no flying windows anymore. And the AIDA implementation can also come as dynamically loadable module, so it can be loaded and linked with the running code using the same technique as for the analyzer plugins. At the moment, there is only one concrete AIDA implementation that satisfies our needs, OpenScientist [http://openscientist.lal.in2p3.fr]. This system comes also with another interface, OpenInventor [http://oss.sgi.com/projects/inventor] for 3D visualization.

So we have 2 types of loadable code, one big AIDA plugin for the histogramming, hereafter called "AIDA" or "AIDA system", and many small plugins for the different tasks on the data flow which we will call just "plugins" or "modules". (There is also another "big" plugin for OpenInventor and, internally used, for the PFManager.) Each of the plugins can in principle generate and fill histograms using AIDA function calls, though it is recommended to use the ready-made "Histo" plugin for that. The schematics of the Flow Analyzer is shown in Figure 2.1, “Basic structure of the Flow Analyzer.”

Figure 2.1. Basic structure of the Flow Analyzer.



The PFManager object can load and store document files, so-called "pflow" files (they have the suffix ".pflow"), maintains the plugins and their connections, starts and stops the data cycle loop, and more. In the drawing, "FlowComposer" stands for any main program. The main program loads and instantiates a PFManager object for each "pflow" file and loads the AIDA system (and also the OpenInventor system if more complex graphics is needed). Therefore, the main program is mainly a wrapper for PFManager to interface with the user. It can be a simple command line tool or an interactive graphical program. Several main programs are available in the distribution.

How is it implemented?

The "core" of the implementation uses POSIX [http://standards.ieee.org/regauth/posix] functions, in particular to load the plugins and the AIDA system. Therefore, it runs on most UNIX systems. We tested it with Linux (Ubuntu, Mandriva, and Scientific Linux) and Mac OS X / Darwin. A simple command-line program, "flowshell", provides the basic functionality, i.e. load the PFManager and the AIDA system, loads a pflow file, starts the run, and gives the GUI control to the AIDA plotter, if desired. The AIDA system OpenScientist in version 16.10 or higher also runs on these systems, and it successfully dynamically links with flowshell. But we also have advanced applications that allow to edit pflow files graphically. For Mac OS X this is FlowComposer, and for all UNIX systems we have a Qt [http://www.qtsoftware.com] based application.

The implementation was not quite trivial because the dynamic loading can be complicated if there are nested dependencies of the libraries. But most difficulties came from the threaded programming: To keep the interactivity with the user (updating the histograms, reacting on mouse clicks), several threads must be maintained, in particular the AIDA system must be able to process GUI events and data events in parallel.

On our request, the OpenScientist people put a lot of work into this issue to get the AIDA system running this way, so many thanks to this group here, in particular to Guy Barrand.

One plugin represents one C++ class that is a subclass of "PFBase". Each class and subclass is stored in a binary dynamically loadable object file with the extension ".pfp". Each plugin can have inputs and outputs. The inputs are implemented as object member functions whose entry points are published in a list and can be called from other modules. The outputs are just names associated with a list of other plugins and member functions that are called from the (sender) plugin. An input member function takes one argument, a memory pointer, pointing to the data to process. Inputs and outputs are "typed", i.e. they know what kind of data the transfered pointer is pointing to. When the plugins are being connected this type is checked and redirected to a converter function, if necessary.

The plugins are usually grouped in so-called flow kits. The base flow kit, that comes with every distribution, provides just functionality to "play" with the Flow Analyzer, generate some gaussian distributed histograms, add and multiply numbers, fill histograms, and so on. But there exist also flow kits for MIDAS [http://midas.triumf.ca] to capture data from experiments running online and offline, and flow kits for particular experiments. Flow kits are actually the "idea" of the Flow Analyzer because they can be *extended*. The C++ source code has a simple structure that can easily used by a beginner. Work can be distributed and different people can improve different plugins.

Chapter 3. The Flow Event

Why FlowEvents?

A FlowEvent is an object that contains a table (dictionary, or C++ map) with key/value pairs. The value of each entry is a FlowProperty and the key is a string that uniquely identifies that property. A FlowProperty in turn is a structure that contains a pointer to a value and a type identifier for these data. It also contains a description, a label, and more.

FlowEvents are supported by the FlowAnalyzer from scratch. The advantage compared with standard types (float, int, vector, ...) is that the grouping of these data is preserved even if a particular plugin does not need all data. It provides also a clear "trigger" for the input of a plugin. For example, a 2D histogram takes the x and the y value as input but since these numbers come in sequence x0, y0, x1, y1, ... the 2D histogram can easily get out of sync and take (x1, y0) ... as wrong number pairs. Even worse, if one of the numbers is rejected by one of the previous plugins there is no chance at all for the histogram to figure out which pairs are the correct ones. If it receives a FlowEvent, the matching properties for x and y can be identified, the corresponding data be retrieved and, if something is missing, the event be rejected.

Even though several "event streams" are in principle possible the usual case is that a FlowEvent is created by one dedicated plugin. This plugin usually creates the FlowEvent only once at the beginning and changes only the data (given by the pointers) during an event cycle. Other plugins can add properties, also at the beginning, and contribute their own data. The idea is that a FlowEvent starts with a minimum of raw data, e.g. ADC values from a decoder, and the connected plugins add calculated data until the plugin chain ends in a histogram where several of these "high-level" properties are picked out to fill the histograms.

In order to calculate other data (properties) from the raw data there are often re-occuring simple operations like multiplying/adding (for calibraion), mapping, and so on.

Chapter 4. The Flow Composer

FlowComposer Mac

Götz Ruprecht <ruprecht@triumf.ca>

The Flow Composer is a full graphical analyzer for physical data. The data source is usually a MIDAS [http://midas.triumf.ca] file or an online MIDAS connection with a running experiment. All histogramming is done using the abstract AIDA [http://aida.freehep.org] and therefore can be done by any AIDA implementation. There is strong support for OpenScientist [http://openscientist.lal.in2p3.fr] but since version 0.6 beta FlowComposer now also comes with a built-in AIDA system supporting CERN's ROOT [http://root.cern.ch].

How to install

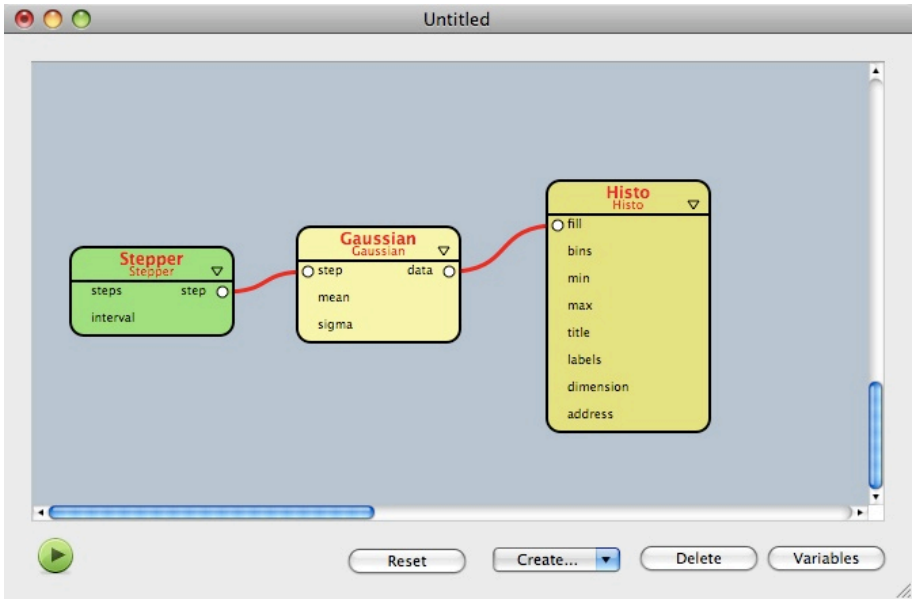
You need an Intel Mac with an operating system of version 10.5 or higher. Download the FlowComposer application (version 0.87 Beta) [http://tactic.triumf.ca/software/download/Mac/FlowComposerMac_0.87_beta.zip] (ZIP), unpack it, and move the application to your favourite folder (probably /Applications).

Building a simple analyzer

Well, this would not be a real analyzer but more a nonsense data producing tool using the FlowComposer/OSC histogramming features. The main document window shows you an empty area at the beginning. On the bottom of the window you see a menu "Create..." that, when opened, shows you the base plugin collection that comes with FlowComposer. This collection is intended to be extended by the advanced user but for now we have to live with these few modules.

From the "Create..." popup menu select "Stepper", "Gaussian", and "Histo", in this order. As soon as you select "Histo" a histogram viewer should pop up. All modules have inputs and outputs. With the mouse, you can connect outputs with inputs. Rearrange the modules in the window to be ordered from the left to the right. Connect "step" from "Stepper" with "step" from "Gaussian", and "data" from "Gaussian" with "fill" from "Histo". The result should look like in the picture below.

Figure 4.1. Document window for simpleanalyzer.pflow



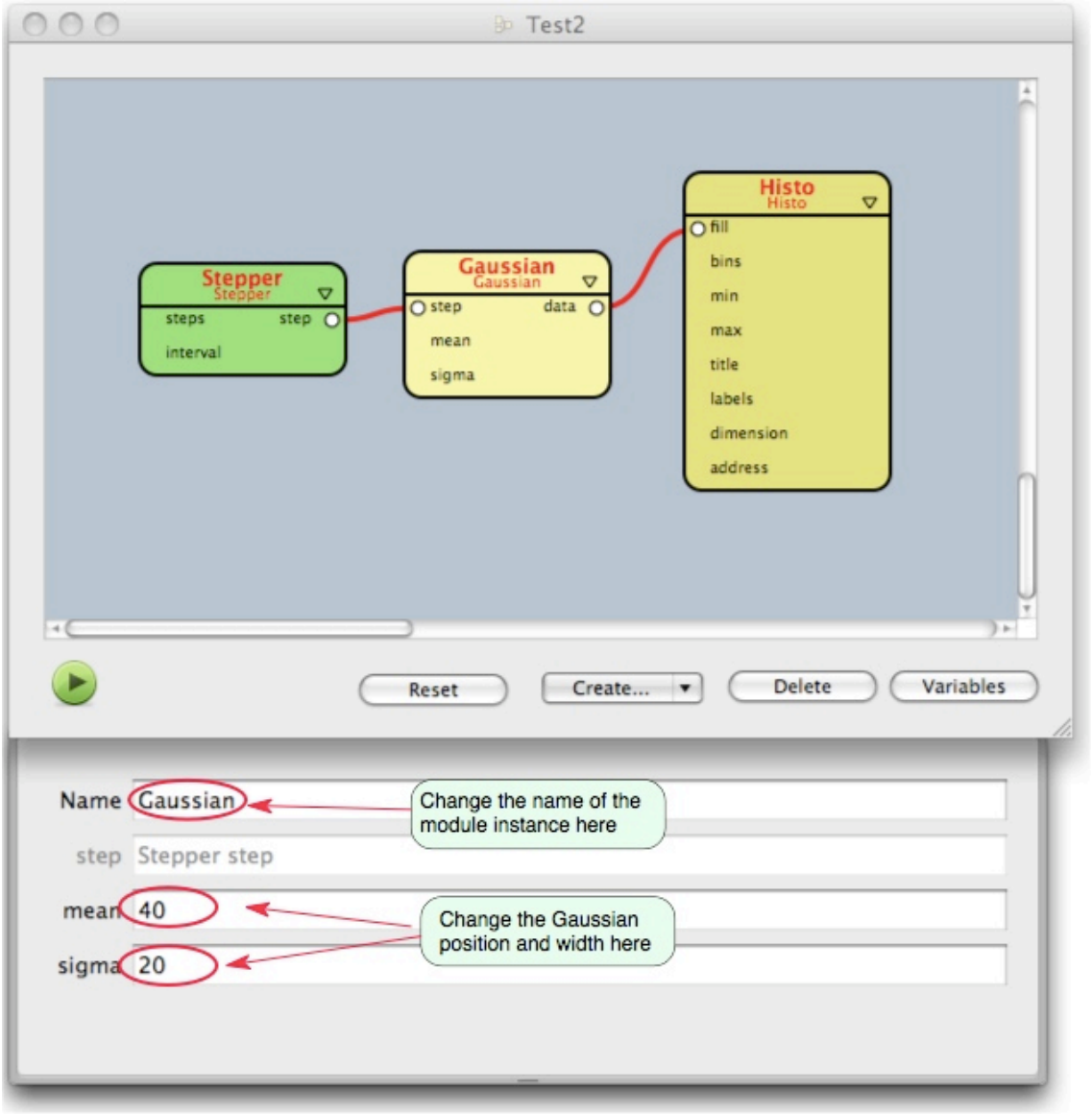
Now press the green "PLAY" button and the analyzer will start to run. In the histogram window you will observe the histogram being filled. While being filled you can work with the histogram window, change the scale, switch to log, etc.

Selecting modules and changing module parameters

Modules are usually in yellow color except the "driver" module (the "generator") which is marked green. The currently selected module is shown in a slightly brighter color tone. You select a module by just clicking on it. You make a module the generator by triple-clicking on it.

If a module is selected you can press the "Variables" button on the bottom right corner of the document window. This opens a drawer where you see the values of all unconnected inputs. Enter a value to assign a constant value to each input or leave it blank to use the default value.

Figure 4.2. Drawer for setting variables



Not all modules are guaranteed to be able to change variables while the analyzer is running but you can try it. However, it is always safer to stop the analyzer (the STOP button) before changing connections and variables or adding/deleting modules.

OpenScientist support

Instead of the built-in "MacRoodyAIDA" you can also use OpenScientist with the advantage that you can customize the plotter user interface. OpenScientist can also use different GUI systems with the most progressive being Qt. Qt for Mac OS X exists in 2 versions, 32bit/Carbon and 32bit/64bit Cocoa. The latter one is currently not working with OpenScientist (but might be in the near future), so we are restricted to the 32bit/Carbon version. Therefore, you must install Qt (32bit/Carbon, version >= 5.4.2) and, of course, OpenScientist (version >= 16.10). The direct download links are

- Qt (32bit/Carbon, version >= 5.4.2) [http://www.qtsoftware.com/downloads/mac-os-cpp]

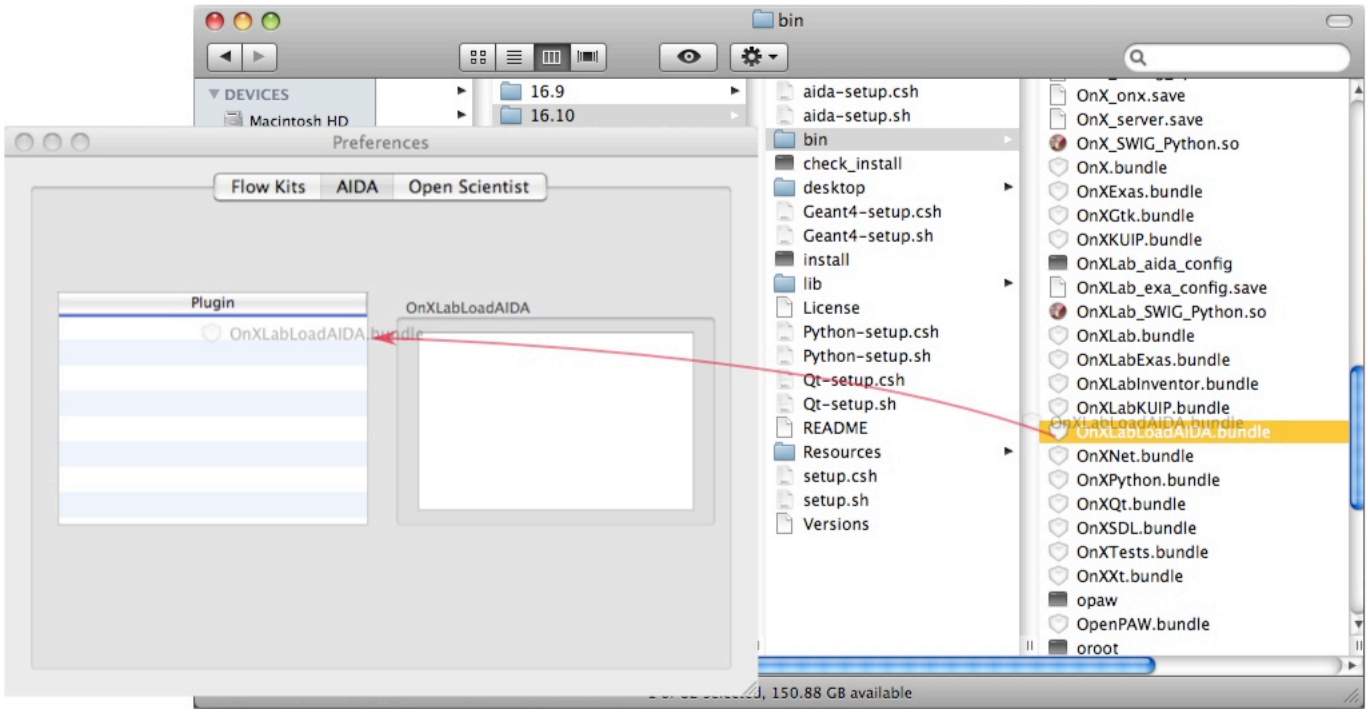
- OpenScientist (version >= 16.10) [http://openscientist.lal.in2p3.fr/download/16.10/osc_vis-16.10-Darwin-i386-gcc_401.zip]

Install Qt strictly following the Qt installer instructions.

OpenScientist comes in a ZIP package. Unpack the the package (if not done automatically) by doubleclicking and drag the osc_vis folder into any directory where you want to keep it. You do not need to follow the OpenScientist installation instructions unless you want to use it also from a terminal (see installation chapter).

Now tell FlowComposer where to find OpenScientist you downloaded previously. To do so, start FlowComposer, open the "FlowComposer->Preferences" menu, and select the "AIDA" tab. Now open a Finder window and go to the folder where you installed OpenScientist, then to the subfolder "bin" and find the file "OnXLabLoadAIDA.bundle". This is the AIDA plugin you want to use with FlowComposer. Make sure that both, the FlowComposer preferences panel and "OnXLabLoadAIDA.bundle" in the Finder are visible on the screen. Now drag "OnXLabLoadAIDA.bundle" with the mouse into the left table (with the title "Plugin") of the FlowComposer AIDA preferences panel as indicated in the picture below.

Figure 4.3. How to drag the AIDA dylib

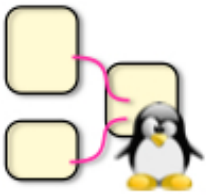


In the plugin list you see now a new row "New Plugin". You can click on it to give it a more meaningful name, like "OSC 16.10 32bit precompiled", for instance. When selecting the row, the right column shows you the complete search path for modules that might be loaded dynamically. The Qt pathes have been added automatically. You can change them if you have chosen a different location for Qt.

Restart FlowComposer and you are ready to go.

FlowComposer Gtk/Linux

Lars Martin <lmartin@triumf.ca>



A graphical frontend for the FlowAnalyzer system

This is a frontend only

This is not a replacement for the FlowAnalyzer system itself, as found in PluginFlow. Before you proceed with this, install the base system and check if your installation worked according to the installation instructions. If you plan to compile FlowComposerGtk from source, you need to do the same with PluginFlow.

How to install

Prerequisites

You need the libgtkmm and libgoocanvasmm libraries installed. On an Ubuntu or Debian system you can just click this link [apt:libgoocanvasmm-0.1-5]. If that doesn't work for you, try finding a package called libgoocanvasmmXXXX in your package manager. For some reason they use version numbers in the package name, so it's hard to tell what the exact name will be. This should take care of both dependencies, as libgoocanvasmm depends on libgtkmm.

Installation

Download the FlowComposer program (32bit [http://tactic.triumf.ca/software/download/Linux32/FlowComposerGtk.tgz]/64bit [http://tactic.triumf.ca/software/download/Linux64/FlowComposerGtk.tgz]), unpack it, and move the executable 'floco' to your favourite folder (probably ~/bin or something similar). If you want to start the program by mouse click, move the script 'FlowComposerGtk' to the same folder and set up a Launcher/Starter/Shortcut/Icon to open it in a new terminal. This way all environment variables should be the same for the program as in a normal shell.

From Source

Prerequisites

You need the sources for libgtkmm and libgoocanvasmm (try libgoocanvasmm-dev [apt:libgoocanvasmm-dev]), as well as the sources for PluginFlow.

Installation

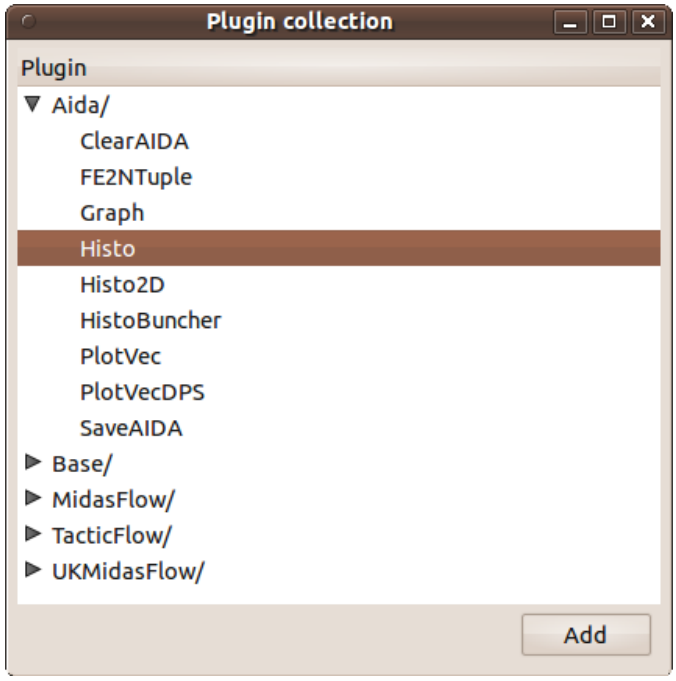
Download the FlowComposer source (CVS [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/FlowComposerGtk], tarball [http://ladd00.triumf.ca/viewvc/FlowAnalyzer/trunk/FlowComposerGtk/?view=tar]), put it in the location of your choice and call 'make'. In order for this to work, PluginFlow needs to be built and installed from source as well.

Building a simple analyzer

Well, this would not be a real analyzer but more a nonsense data producing tool using the FlowComposer/OSC histogramming features. The main document window shows you an empty area at the beginning.

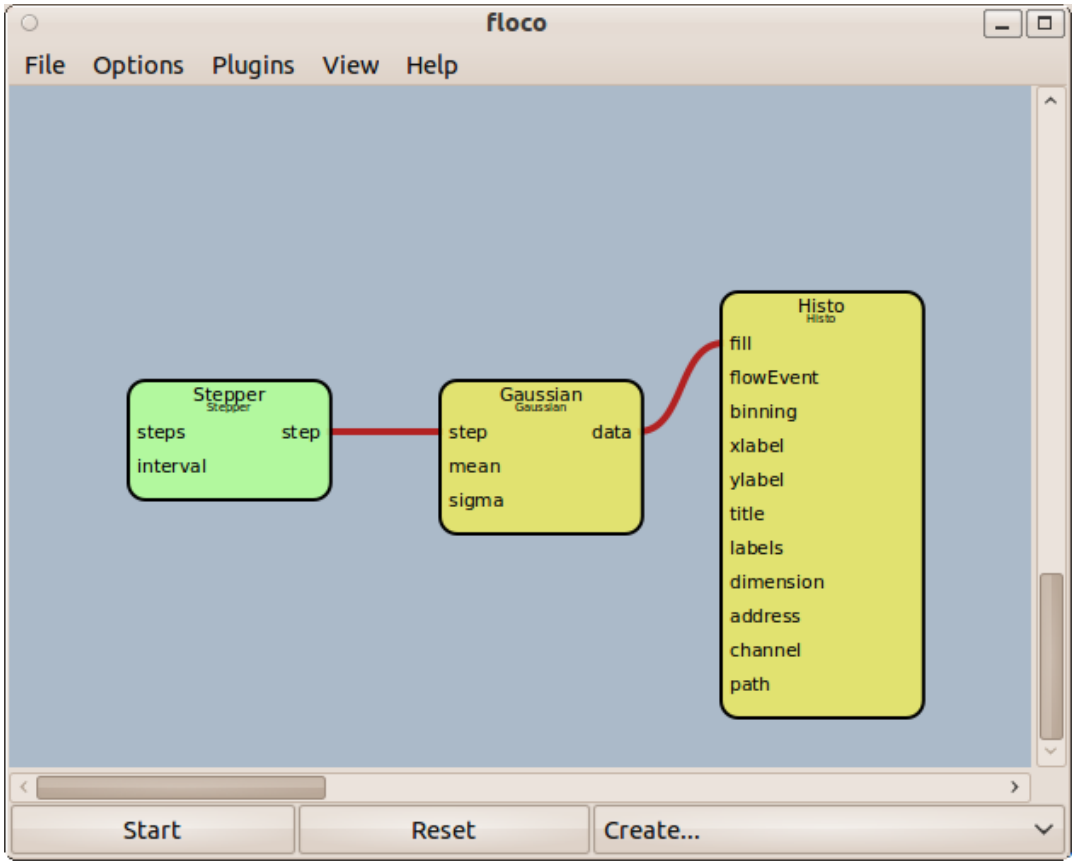
If you select "Plugins/From Collection" from the menu bar you will open a second window showing the base plugin collection that comes with FlowComposer. This collection is intended to be extended by the advanced user but for now we have to live with these few modules.

Figure 4.4. Collection of Plugins after additional Kits have been added



Add "Base/Stepper", "Base/Gaussian", and "Aida/Histo", in this order (You can add by selecting and clicking 'Add' or just by double-click.) As soon as you select "Histo" a histogram viewer should pop up. All modules have inputs and outputs. With the mouse, you can connect outputs with inputs. Rearrange the modules in the window to be ordered from the left to the right. Connect "step" from "Stepper" with "step" of "Gaussian", and "data" from "Gaussian" with "fill" of "Histo". The result should look like in the picture below.

Figure 4.5. Document window for simpleanalyzer.pflow



Now press the 'Start' button and the analyzer will start to run. In the histogram window you will observe the histogram being filled. While being filled you can work with the histogram window, change the scale, switch to log, etc.

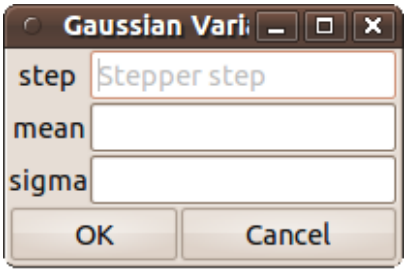
Selecting modules and changing module parameters

Modules are usually yellow in color except the "driver" module (the "generator") which is marked green. The currently selected module is shown in a slightly brighter tone. You select a module by just clicking on it. You make a module the generator by selecting 'Set Start' from the right-click context-menu.

Other things you can do from the context menu are:

- Delete the module
- Switch between collapsed and uncollapsed display mode (in collapsed mode, only connected in- and outputs are shown, reducing the size of the plugin box)
- Open the modules variables window. This opens a window where you see the values of all inputs. Enter a value to assign a constant value to each input or leave it blank to use the default value.

Figure 4.6. Window for setting variables



Not all modules are guaranteed to be able to change variables while the analyzer is running but you can try it. However, it is always safer to stop the analyzer (the STOP button) before changing connections and variables or adding/deleting modules.

Chapter 5. Flow Kits

Adding FlowKits

In addition to the included basic plugins, it is possible (and necessary to do real work) to add further modules. They are grouped in Kits, so-called FlowKits. To add a FlowKit, unzip it where you like and

- *FlowComposer (Mac):* drag the folder into the FlowKits list in FlowComposers preferences. If two folders with the same name are added, the plugins are both shown in the same category. If these folders contain plugins with the same name, position in the FlowKit list decides which one is used.
- *no FlowComposer (Unix):* add the path of the folder to the environment variable LD_LIBRARY_PATH (for Mac, use DYLD_LIBRARY_PATH), analog to the instructions in installation.

Note

Each FlowKit folder contains a short description of the included Plugins in "plugin_summary.txt"

The basic PluginFlow kit

Display and WriteToFile

Any "normal" data produced by modules (normal means plain C data types like float, double, int, etc, or C++ vectors [http://en.wikipedia.org/wiki/Vector_(C%2B%2B)] of plain types) can be sent to standard output (if you don't see a terminal in FlowComposer (Mac), make sure the "redirect standard output..." and "Terminal window..." boxes are checked in Preferences/Misc and restart the application) by using the Display module.

Simply connect the desired output to Display's "data" input. You can specify optional pre- and suffixes (i.e. "Pulseheight:\t") to distinguish the output from several Displays. The default prefix is empty, the default suffix is a linebreak ("\n").

The WriteToFile module works exactly the same way, except that you specify a filename to write to instead of standard output.

Histo, Histo2D and PlotVec

These are all OpenScientist based AIDA-histograms. PlotVec simply takes a C++ vector of numbers at the "signal" input and plots it in the OpenScientist window. If you're using the BasicPlotter setup (Preferences/ OpenScientist or environment variable ONXLAB_PLOTTER) it should show up in the main memory tree on the left, in other setups it may be on a different page.

If you want to plot several channels, fill in the "dimension" entry (i.e. "3 4" for three folders containing four histograms). You can label these dimensions with the "label" input, i.e. "Module Channel". Then use the address related outputs (i.e. "module", "channel" on VFDecoder) of the module producing the data to generate an address vector with Vectorize and connect the vector output to PlotVec's "address" input.

Additional information for the histograms Info box can be fed in via the "fw_values" input and selected and labeled via "infos".

The Histo modules are similar but different. They take numbers at their "fill" input(s) and fill a histogram. Range and binning have to be set via the appropriate inputs. Creation and addressing of several histograms works exactly as for PlotVec. There are several example pflow files included in the kit. For the more advanced features of histograms, please refer to the file PlotVecTest.pflow in the MidasFlow kit for a typical use of some of the modules.

All three of these modules also have inputs for a FlowEvent and accept the names of properties contained in the FlowEvent in some of the inputs (the "fill" and the "channel" inputs in particular). Binning and axis labels are taken from the FlowEvent automatically unless they are set manually by the user.

MidasFlow

MacOSX 10.4/10.5	Ubuntu 9.04	Scientific Linux	Mandriva
x86 [http://tactic.triumf.ca/software/download/Mac/MidasFlow.zip]	-------- x86 [http://tactic.triumf.ca/software/download/Linux32/MidasFlow.zip], x86-64 [http://tactic.triumf.ca/software/download/Linux64/MidasFlow.zip] ----------		

Remember to add this to your LD_LIBRARY_PATH / the FlowKits section in the FlowComposer preferences.

MidasFlow provides modules for interfacing with MIDAS [http://midas.triumf.ca]. This also includes the handling of decoded events for several different sets of electronics.

If you plan to use Midas files, your first step should be to set the environment variable MIDASDATA (also in "Preferences/Misc" in Mac FlowComposer) to the path of the directory containing your Midas files (drag&drop works).

To open one or more Midas files, use MidasFile and enter the filename, or compile a list of filenames in a textfile and use that as "listfile". The file will be looked for in the path specified by MIDASDATA.

The "event" output will provide MidasEvents, the "bor"/"eor" outputs provide the copy of the ODB sent at begin-of-run/end-of-run.

MidasEvents can be sent to standard output (if you are running the Mac FlowComposer and don't see a terminal, make sure the "redirect standard output..." and "Terminal window..." boxes are checked in Preferences/Misc and restart the application) with the MidasEventOut module or decoded with a hardware-specific module like VFDecoder or StruckDecoder (look at the respective tooltips for help on these) or something for reading out standard Midas outputs like MidasValue. They can also be split by MidasSplitter according to their EventID before further processing (i.e. ID 1 to VFDecoder, ID 3 to MidasValue).

Since TACTIC [http://tactic.triumf.ca] uses VF48s, most hardware specific modules are VF48-related (they also work for VF64 and TIG10):

- VFEventOut sends VFEvent information to standard output
- VFCoincidentor sorts VFEvents in groups according to timestamp.
- VFTimes compares timestamps/signal times of VFEvents to find coincidence times
- VFSplitter provides the data from events at separate outputs

The kit includes pflow examples to explain the usage of some modules.

PlotVecTest.pflow reads a datafile and produces a hitpattern, a signal view and several addressed histograms. This is a good example for the creation of multiple histograms and the implemented addressing system.

The Midas file used in this example (a real TACTIC run) is to large to be included in the package but can be found here (140 MB) [http://tactic.triumf.ca/software/download/testdata/run06116.mid.gz]. The file does not need to be unzipped to be read (but can be).

TigressTest.pflow reads data from the TIGRESS [http://tigress.triumf.ca] experiment. The example datafile is here [http://tactic.triumf.ca/software/download/testdata/run00473.mid.gz].

To connect to a running Midas experiment use the MidasOnline module and set the parameters: host to the name of the computer running Midas (mserver), the default is localhost, experiment to the name of the experiment if more than one are defined, and IDs to the EventID you want (default: all).

It should be obvious that you should NEVER try to connect to somebody else's experiment without asking permission.

TacticFlow

MacOSX 10.4/10.5	Ubuntu 9.04	Scientific Linux	Mandriva
x86 [http://tactic.triumf.ca/software/download/Mac/TacticFlow.zip]	-------- x86 [http://tactic.triumf.ca/software/download/Linux32/TacticFlow.zip], x86-64 [http://tactic.triumf.ca/software/download/Linux64/TacticFlow.zip] ----------		

Remember to add this to your LD_LIBRARY_PATH / the FlowKits section in the FlowComposer preferences.

Specific to the TACTIC [http://tactic.triumf.ca] experiment.

The interface between the actual analysis of data from a TACTIC run and the more or less abstract MidasFlow is the module TacticFromVF. It takes a vector of VFEvents (from VFCoincidentor) and several parameters or slowly changing values (i.e. HV and pressure from MidasValue) to construct a TacticEvent, the basic structure for all following analysis.

These Tactic events can then be fed to TacticPlotter, a module that plots the ion tracks in the OpenScientist window (You need to be using TacticPlotter.onx for this.), or to TacticTracker, the module meant for doing the bulk of the track analysis (i.e. vertex reconstruction, angle calculation, energy calculation etc. -THIS IS VERY MUCH A WORK IN PROGRESS-) and filling in the additional data in the same TacticEvent object.

Toaster produces a string representation of the TacticEvent, ready to be written to a file for later visualisation with TOAST [http://trshare.triumf.ca/~ulrike/toast.jar] or to be presented on a server for online visualization with TOAST or taclet. (We don't have a working server at the moment.)

Finally, specific data can be extracted from a TacticEvent with TacticSplitter for histogramming and signal plots.

The included example TacticTest.pflow shows a typical setup (without the dozens of histograms attached to the TacticSplitter, this is left as an exercise for the reader.) It uses the same Midas file (140 MB) [http://tactic.triumf.ca/software/download/testdata/run06116.mid.gz] as the MidasFlow example PlotVecTest.pflow.

UKMidasFlow

MacOSX 10.4/10.5	Ubuntu 9.04	Scientific Linux	Mandriva
x86 [http://tactic.triumf.ca/software/download/Mac/UKMidasFlow.zip]	-------- x86 [http://tactic.triumf.ca/software/download/Linux32/UKMidasFlow.zip], x86-64 [http://tactic.triumf.ca/software/download/Linux64/UKMidasFlow.zip] ----------		

Remember to add this to your LD_LIBRARY_PATH / the FlowKits section in the FlowComposer preferences.

Specific to data produced by the TUDA [http://tuda.triumf.ca] DAQ. UK MIDAS is based on the EUROGAM data packet structure, see also here [http://npg.dl.ac.uk/MIDAS].

In order to analyse a UKMidas run file, use the UKMidasFile module, select it and click the variables button. Put the filename in the appropriate box (drag&drop works if the textbox is selected). *The file will be searched in the directory specified in the environment variable UKMIDASDATA (also in "Preferences/Misc" in Mac FlowComposer) or by absolute path.*

The "step" parameter decides how many events you want to read from the file (0 means all of them), "interval" is a pause between event reads.

To check if it works, attach a TudaEventOut to the output of UKMidasFile. If you click the green "Play" button, information should start to appear at standard output (if you don't see a terminal, make sure the "redirect standard output..." and "Terminal window..." boxes are checked in Preferences/Misc and restart the application)

When this succeeds, you can add TudaAdcFilter (you can remove or disconnect TudaEventOut if you don't want it anymore). Write the desired channel number into the approriate box and start if not still running. (If the file was read completely you will have to click Reset to restart from the beginning.)

The output "value" now provides the values of that channel which can be connected to a histogram, Display or other modules as described in PluginFlow.

The included example TudaTest.pflow is a simple test setup, reading a datafile and producing a hitpattern and an ADC histogram.

FlowComposer should open an OpenScientist window with a tree containing two Histograms on the left. If you select the ADC histogram after running a while, a click on the "Peakfind" button should produce a Gauss-fit. If not, open Analysis/Parameters and check "Gauss-fit". The fit output should show up in the terminal.

The UKMidas file used in this example (a TUDA run) is to large to be included in the package but can be found here (31 MB) [http://tactic.triumf.ca/software/download/testdata/run29_0.tuda].