A Programmers Introduction to

Pamguard Developer Training Notes

Douglas Gillespie

Developer Training HWU 29 March, 2007

Introduction

Pamguard is an open source project, and as such, developers are welcome to look at any part of the code, comment on it, modify it, and hopefully improve it !

The purpose of this session is to concentrate on the development of new detector modules using the existing Pamguard infrastructure. We assume that participants in this session have some programming experience. If you're already a JAVA expert, you'll find it easy. If you've experience of C++, then JAVA will look pretty familiar, and you'll probably be pleased at how easy it is in comparison. If you're new to object orientated programming, then hang on in there, we hope you'll still get something out of this session !

This tutorial is laid out in two principal parts

1. An overview of the Pamguard program structure and how the different modules relate to one another and pass data between them.
2. A walk through example of a really simple detector, which has been written primarily to demonstrate and show off the Pamguard core structure and components rather than to be the ultimate in signal detection.

Documentation

The Pamguard source code is extensively documented using the Javadoc tool from Sun Microsystems. This documentation may be viewed as comments within the source code, may appear as pop-up help from within some development environments (such as Eclipse, which you will be using during the tutorial) and is also available on the web at http://www.pamguard.org/devdocs/index.html.

Eclipse

The Eclipse integrated development environment (IDE) (www.eclipse.org) and the Java software development kit (Sun's JDK 1.5) are already installed on the machines you'll be using. When you launch Eclipse, it will automatically take you to the Pamguard project. From the Window menu, select 'Open Perspective' and the 'Java Browsing' and your display should look something like this:

Working along the top of the top of the window, you'll see that the Pamsoft2006 'Project' is divided into a number of 'Packages', each of which contains a number of 'Types' – Java classes or interfaces, each of which contains 'Members' – functions of variables. Take some time to browse around, opening and closing the various 'Types' in the main window.

To run Pamguard from within the Eclipse IDE, press the run button (solid green triangle) in the toolbar. To debug Pamguard, press the debug button (icon with a picture of a little bug) just to the left of the green run button.

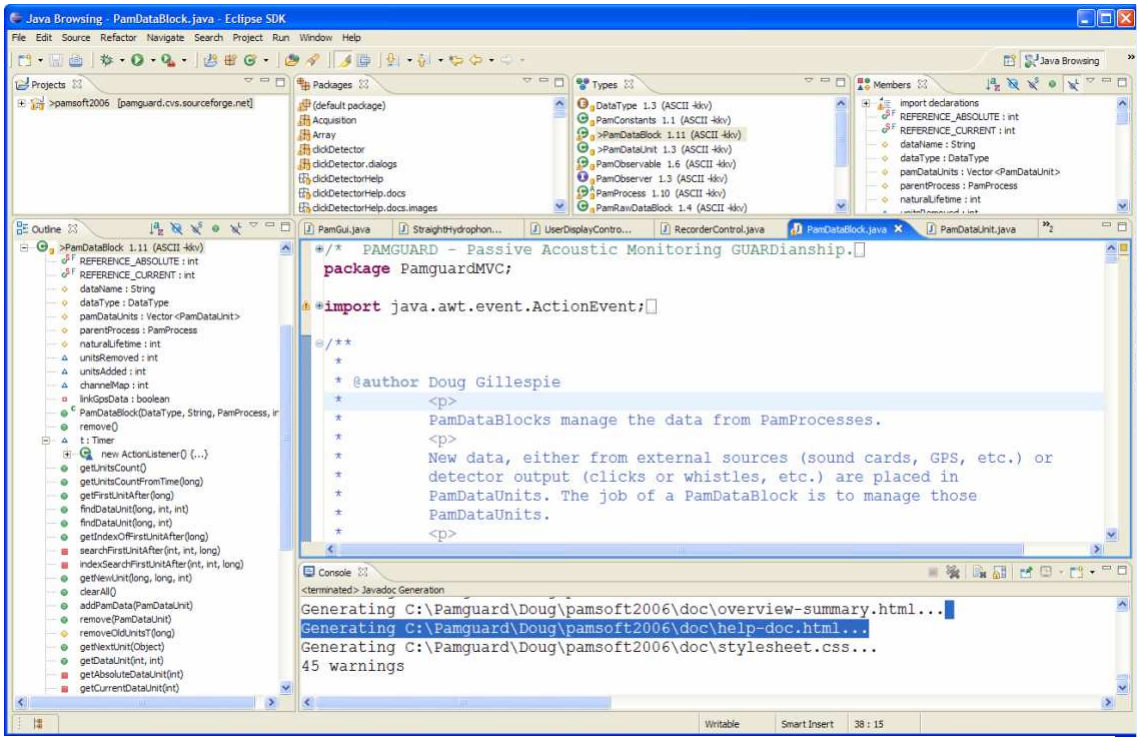

Figure 1. The Eclipse IDE.

When writing code within Eclipse, there is an excellent 'auto complete' function. Start typing a command and press 'Ctrl' and the space bar together and a list of possible commands / classes should appear. To find out what's available in the class you're working on, try typing the word `this`. (to refer to the current object) and holding pressing 'Ctrl-space' and see what's there ! Using auto complete will not only tell you what commands are available, but will spell them correctly for you (Java is a case sensitive language) and it will also insert any 'import's' you need in order to access classes from other packages (Java imports are similar to C header files).

Hover the mouse over almost anything and you'll probably get a pop-up telling you all about it.

Eclipse is very good at telling you if something is wrong. Any code that will not run will be underlined and the type / package / project will be highlighted with a little cross and a little red marker near the scroll bar so you can sort out errors in your code before you even try to compile and run it.

A function you may find useful as you try to understand the code is System.out.println(any object) which will print the output (or more specifically, the objects toString() member function) to the Eclipse console.

To add breakpoints for the debugger, right click in the margin to the left of the line you wish to pause at.

If you implement a Java interface, don't type the names of all the functions yourself ! Just right click on the java file, select 'Source, then 'override/implement methods…'

and you'll get a pop up box of possible functions with the ones you **must** implement as part of the interface already selected.

That's all you need to know about Eclipse for the time being.

Pamguard Structure (a brief Introduction)

Pamguard Data flow.
Most data within Pamguard are handled with PamDataBlocks and PamDataUnits. Check out the Javadoc help for these classes, starting at http://www.pamguard.org/devdocs/PamguardMVC/PamDataBlock.html or by looking at the source code, which you'll find in the PamguardMVC package.

Pamguard plug-in packages
Pamguard consists of a number of plug in packages each of which may or may not process data, may or may not have a display and may or may not have some user controls or configurable options. Every plug in inherits it's basic behaviour from a class PamControlledUnit. Check out the java doc for this class at http://www.pamguard.org/devdocs/PamController/PamControlledUnit.html and follow the link the 'how to make Pamguard plug-ins' overview.

A Pamguard plug in should be able to do absolutely anything you want, including drawing on the map, writing to the database, drawing over spectrogram displays, etc. without having to modify the map package, without having to modify the database package, without having to modify the spectrogram, etc. All the programmer has to do is to make a class based on PamControlledUnit and add one or two lines to the PamModel class to let Pamguard know that's it's available and everything else will be automatic.

The Example Package

An example plug in package 'WorkshopDemo' has been written specifically to demonstrate how to use PamDataBlocks, PamDataUnits and main Pamguard display features. It consists of a relatively simple detector that runs and 'in band energy detector' on spectrogram data. A measure of mean background noise in a given energy band is made as new data arrive. The instantaneous energy in that band is compared to the background and a detection is made whenever the energy / background ratio goes over threshold and then falls back down again.

The signal to noise ratio of the detector may be displayed as a plug in window on the bottom of spectrogram displays. If a detection is made, it may be displayed as a coloured rectangle overlaid on top of the spectrogram displays and as a symbol on the map display.

Running the detector

You should already be familiar with how to use Pamguard. Create an instance of the 'Workshop Demo' detector from the File / Add modules …' menu. If necessary, you will prompted to create an FFT data block to convert raw audio data into a spectrogram. You will also need at least one 'User Display Panel' on which you should create a spectrogram display.

To view the signal to noise ratio display, right click on the spectrogram and select 'Settings…' go to the 'Plug ins' tab and check that 'Workshop demo detector' is selected. The panel should then be visible at the bottom of the spectrogram display.

To view detections as overlays on the spectrogram, again right click on the spectrogram panel and select your detector from the pop-up menu.

Play a sound file back through the PC sound system, as per the user demonstration, and start Pamguard detection from the 'Detection / Start' menu.

Detection parameters may be adjusted from the 'Detection / Workshop demo detector parameters' menu.

A Walk through the code.

The example code consists of seven main Java classes and two extra lines in the PamModel class. These are listed in the order:

WorkshopController	is the main plug in class controlling the detector
WorkshopProcess	is the main detection process (does the work)
WorkshopProcessParameters	contains parameters controlling the detection process
WorkshopParametersDialog	is a dialog box used to set detection parameters
WorkshopOverlayGraphics	provides graphics functions for displaying detections on top of spectrogram displays
WorkshopPluinPanelProvider	provides plug in panels for the bottom of spectrogram displays.
WorkshopSQLLogging	provides functionality for logging detections to the Pamguard database.

The two lines added to PamModel are

```
mi = PamModuleInfo.registerControlledUnit("WorkshopDemo.WorkshopController",
        "Workshop Demo Detector");
```

and

```
mi.addDependency(new PamDependency(DataType.FFT, "fftManager.PamFFTControl"));
```

The first of which tells the Pamguard Model that this detector is available. The second line tells Pamguard that this detector requires FFT data and that the preferred source of FFT data would be an instance of fftManager.PamFFTControl.

WorkshopController

This is a subclass of PamControlledUnit (see Javadoc)

It creates instances of the WorkshopDetectionParameters, the WorkshopProcess and the WorkshopPluginPanelProvider.

It overrides the PamControlledUnit function NotifyModelChanged to make the detector aware when other modules are added or removed from Pamguard.

It overrides the PamControlledUnit functions createDisplayMenu() and createDetetionMenu() in order to set up menu commands for adjusting detection and display parameters. These menu items will automatically be incorporated into Pamguards main Detection and Display menus.

It implements the interface PamSettings and registers with the PamSettingsManager so that the WorkshopDetectionParameters are stored along with other Pamguard configuration settings when Pamguard is closed and restarted.

WorkshopProcess

WorkshopProcess does the actual work of making detections. Since the detector may be required to operate on several channels simultaneously, some of the functions are contained within an inner class ChannelDetector.

WorkshopProcess is a sub class of PamProcess. PamProcess already implements the PamObserver interface, enabling it to listen out for new PamDataUnits. WorkshopProcess only has to override the newData() function to get these notifications from the PamProcess superclass. When new data arrive, the channel number of the PamDataUnit is examined, and the data passed on to the appropriate ChannelDetector.

WorkshopProcess uses three different PamDataBlocks, one source data block and two output data blocks.

The source dataBlobk, fftDataSource, is the PamDataBlock the detector will subscribe to (become an observer of) in order to get notifications when new FFT data units are created.

Two output data blocks are created:

1. outputDataBlock will contain detections, (if and when they occur).
2. backgroundDataBlock will contain information required by the spectrogram plug in panels, if any are ever created.

Note that outputDataBlock is registered with the Pamguard system using the addOutputDataBock() function. This will enable other Pamguard modules, such as the map and spectrogram display, to find the data bock in the system and update their own options menus and dialogs accordingly. The backgroundDataBlock is only used internally within this module, so there is no need to tell the rest of Pamguard about it.

Two additional functionalities are added to outputDataBlock: setOverlayDraw uses the WorkshopOverlayGraphics class to provide drawing functionality for both the map and the spectrogram displays. setLogging uses the WorkshopSQLLogging class to provide functionality for logging data to the Pamguard database.

prepareProcess is called whenever the Pamguard model changes (i.e. modules are added or removed) and whenever detection parameters are adjusted in WorkshopController. It's here that the WorkshopProcess subscribes to the correct data source and sets up the individual channel detectors.

PamStart is called by the Pamguard system just before detection starts. It doesn't do much apart from reset some of the detection flags in each ChannelDetector. Real detection will start when the first FFT datablock arrives.

ChannelDetector inner class
This does the actual work of making detections on an individual channel. Once a detection is made, it uses the outputDataBlock to create new PamDataUnits and send them off to the Pamguard system. *Note that unless the database is active or one of the Pamguard displays is using this data in some way absolutely nothing will happen to this data and it will very likely get deleted about a second later !*

WorkshopProcessParameters
Is a simple set of parameters controlling detection and display for this detector. They are all put into one class like this so that the PamguardSettingsManager can store them easily and efficiently in a serialised binary file (the ones you select when you start up Pamguard). Any class that will be used by the settings manager must implement the Serializable interface. This one also implements Cloneable so that it's easy to copy.

WorkshopParametersDialog
Is a dialog for setting workshop parameters. It's called from the WorkshopController menu ActionListener. It uses the PamDialog class to implement some standard functionality such as the SourcePanel class which provides a quick and easy way of generating a drop down list of available data sources and channel numbers. This is just an example of how dialogs are made – you can do it any other way you want depending how much you like the various Java LayoutManagers.

WorkshopOverlayGraphics
Implements PanelOverlayDraw and allows detections to be drawn on the map and on spectrogram displays. The Pamgaurd system can find all the different data blocks, by working through lists of ControlledUnits, PamProcesses and output data blocks. Our output data block has had an overlay graphics member set. Different displays within Pamguard will call WorkshopOverlayGraphics. CanDraw(GeneralProjector) to see if this particular implementation of PanelOverlayDraw knows how to draw on those particular displays. This depends on the axis types set in the GeneralProjector, which are Latitude and Longitude for the map and Time and Frequency for the spectrogram. The projector will turn Latitude and Longitude or Time and Frequency into screen coordinates without the WorkshopOverlayGraphics class having to know anything at all about spectrogram scales, map orientation or scales, etc.

If the map (or spectrogram) is set to draw Workshop Demos DataUnit's, then the map (or spectrogram) will subscribe to the data units. When new units arrive at the map (or spectrogram), or when the map (or spectrogram) redraws, it will call WorkshopOverlayDraw with it's projector as an argument.

WorkshopOverlayGraphics can work out what type of projector it is (map or spectrogram) and call the appropriate drawing function.

WorkshopPluginPanelProvider
As the name suggests, this class provides plug in display panels. These can currently only be 'plugged' into the bottom of the spectrogram display, it is possible however that other displays will support the same plug ins in the future.

The WorkshopPluginPanelProvider contains an inner class WorkshopPluginPanel, which extends DisplayPanel and implements PamObserver.

The WorkshopPluginPanels each show a line graph of signal to noise ratio (SNR) for each channel, they also have fixed lines at SNR = 0 and at the value of the detection threshold.

The various Pamgaurd displays will make as many individual WorkshopPluginPanels as necessary and each will run independently (for instance, they may end up with different x and y scales depending on what they are plugged into)

WorkshopPluginPanel implements PamObserver and each instance subscribes to the backgroundDataBlock from workshopProcess in order to receive updates from the process as new SNR values are calculated for each channel.

Drawing on plug in panels takes place in two different places

1. Each time the update position on the spectrogram display changes, containerNotification() is called which informs the PluginPanel that scales or scale offsets have changed. In this example, all that happens is that the plot is cleared for a few pixels to the right of the current x coordinate on the spectrogram container. (Try commenting out these lines and you'll see that the line graphs are continually drawing over themselves).
2. Each time new data arrive from the backgroundDataBlock, those new data are drawn on the line graphs. Information required to calculate x coordinates from the time of the backgroundDataUnit and from the spectrograms current x coordinate and current time is used. Previous values are stored locally so that lines can be joined up.

WorkshopSQLLogging
This class is used to add functionality to the main detection outputDataBlock which will allow data to be written to the Pamguard database. Pamguard currently only supports MySQL databases and MySQL is not installed on the teaching machines making it difficult to demonstrate this code.

The abstract SQLLogging class has been created to hide all the unpleasantness of writing database Structured Query Language (SQL) commands from the Pamguard user. SQLLogging will automatically create the appropriate database table, and populate that table with data in the format chosen by the developer. If additional table columns are added to contain additional information at a later date in the development process, those columns will get added to an existing table. As we implement support for other databases (such as MS Access), the developers code will add tables to those databases and populate them in the exact same way.

SQLLogging has two main functions, the first provides the table definition, the second (setTableData(…)) gets called back by the database manager when a new PamDataUnit is created and requires the developer to populate the database fields.

When defining a database table format, it is possible to set cross reference information to other database tables. In this example, we store a reference to the most recent entry into the GPSData table.

Each database column is defined with a specific format (e.g. TIMESTAMP, FLOAT, LONG, etc.). The types of the data set in setTableData must match the column types or the write operation may fail.

Suggested Exercises

Improve the detector:

1. Set a maximum length for detections
2. Set a minimum gap within detections, so that if the gap is small, detections merge into one.
3. Include any parameters controlling these functions to the dialog and the WorkshopProcessParameters.

Use more Pamguard functionality

1. Add a counting module that appears as a side panel giving the number of detections in the last 10 minutes (Hint: to do this, copy the ClickDetector.ClickSidePanel class and override the getSidePanel() function in WorkshopController).
2. Automatically trigger recordings whenever the detector makes a detection. For this to work, you will need to create at least one instance of the Pamguard Sound Recorder from the Pamguard 'File/Add modules…' menu, then add the following code:
    a) Create an inner class in the WorkshopProcess which implements the RecorderTrigger interface.
    b) Create an instance of this class and register it with the class RecorderControl using RecorderControl.registerRecorderTrigger (…)
    c) Whenever you have in interesting event (i.e. at the time you create a new PamDataUnit) call RecorderControl.actionRecorderTrigger(…) which will tell all sound recorders in your system that you want to record.

What will happen ?

a) When you register your recorder trigger, on each sound recorder panel, you should get to see a check box allowing you to enable or disable the trigger for that particular recorder.
b) When you call actionRecorderTrigger, all sound recorders that have selected triggering from your detector will start a recording, taking historical data that's been stored in a buffer and adding it to the start of the recording and then continuing that recording for the amount of time defined in the member functions of your class that implemented RecorderTrigger.

See the Click Train Detector class ClickDetector.ClickTrainDetector for an example of this code.

Note that this is a rather naive example since this simple detector will probably trigger quite often, so you'll end up recording almost continually. Automatic recorder triggering is much better suited to events that are relatively rare, such as the detection of a sperm whale click train.

Add your own detector
We hope that Pamguard will be able to provide you with a framework within which you can develop your own detector modules and also hope that you will then share your detectors with the rest of the research community as we have shared ours.

The workshop demo detector took one of the Pamguard core development team a bit less than one day to write from scratch, so it's unlikely that you will have time to write your own detector in this training session. However, please take the opportunity to discuss what your detector does, what it needs as input and what it would provide as output with one of the core team and we will be able to advise you on how to proceed. (For info, the workshop demo detector took one of the core team one day to write from scratch).

Feedback

If you think the Pamguard code is total rubbish: Please tell us what you think, your comments will be useful to us and will help to make future improvements.
If you think the Pamguard code is not bad: Please tell us and we might even buy you a drink.