| | |
|---|---|
| **Doc type** | **PAMGuard Technical Note** |
| **Project Name:** | **PAMGuard Modification to Group Identification** |
| **Client:** | NOAA/NMFS PIFSC |
| **Client Contact** | Erin Oleson |
| **Lead Scientists:** | Douglas Gillespie |
| **Project Manager:** | Jason Wood |

| | |
|---|---|
| **Written by:** | Douglas Gillespie |
| **Date:** | August 23, 2016 |

DRAFT For public consultation

Sea Mammal Research Unit

PAMGUARD

# Table of Contents

# 1 Summary

This document describes the current data labeling and cross referencing systems used within PAMGuard and available for use in other programming languages such as Matlab or R. The current system is functional, but has a number of deficiencies making it hard to cross reference between data visible on the GUI, records in the PAMGuard database and data in the PAMGuard binary files.

Three alternatives are discussed in section 6 which might improve on the current systems. These are

1. Maintaining, but improving on the status quo through better documentation and a library of data access functions in Matlab or R
2. Adding additional numeric identifiers to all PAMGuard data. This would involve modifications to file and database formats which would not be backwards compatible.
3. The use of time as a unique identifier

Our current thinking is that option 2 (section 6.2) combined with minor improvements to the PAMGuard GUI (section 7) will provide users with a much improved data identification and cross referencing system however, this will involve the inconvenience of having to change the PAMGuard binary file structures in a way that will not be backwards compatible.

Comments on this document are welcome both at multiple levels:

1. Suggestions as to which of the discussed options would be most beneficial (or difficult) for users
2. Suggested minor changes to any of the options
3. Entirely new options which we may have overlooked

All correspondence should be sent to Douglas Gillespie dg50@st-andrews.ac.uk The consultation period will end on 9 September 2016 in order that work on code modifications can proceed in September / October 2016.

# 2 Glossary

| | |
|---|---|
| Binary Data | Files written to a hard drive in a proprietary data format which cannot be directly opened and read by humans and requires advanced knowledge of the exact file format before the files can be decoded. |
| Database | a collection of information organized to be easily accessed, managed, and updated |
| Data unit | A single unit of data (e.g. a click, a whistle, a GPS position, etc.) |
| Data Stream | Data output from a specific detector or process within PAMGuard (e.g. clicks, whistles, GPS positions, etc.) |
| GUI | Graphical User Interface |
| SQL | Structured Query Language (used to read and write from a database) |
| Timestamp | A number representing both date and time |
| UID | Unique Identification code |
| UTC | Universal Time Code (GMT, 'Zulu', etc.) |

# 3 Introduction

Detectors and other modules in PAMGuard are capable of generating many thousands of data objects (data units) per hour. These are stored in either a relational database or data files written in a proprietary binary format. During data analysis, the analyst often has to combine data from both the database and the binary data store, as well as data from raw audio sound files. Some analysis takes place using the PAMGuard Viewer software but many people also develop their own analysis routines using Matlab, 'R' or other analysis packages. With the current data labelling system, people find it difficult to relate data in the various data stores which directly impedes analysis.

This document describes how data are currently labelled and identified and discusses options for an improved data labelling system. This document will be circulated to PAMGuard developers and users for comment prior to work taking place to implement an improved data labelling system in PAMGuard.

# 4 Notes on the Java Language, Dates and Data Indexing

The PAMGuard software is written in Java. There are a few things worth knowing about Java which will help to understand why certain things are the way they are in PAMGuard data.

Any programmer will understand the concept of arrays of data. For numeric data, they are simply a way of storing a row or column of numbers instead of single numbers. In Matlab and R, many calculations can be applied to entire arrays at a time though in most programming languages it is necessary to write a loop to process array elements one by one. What is important is how data within an array are *referenced*. In Matlab, the *index* of the first element in an array is 1. For example, if you have an array called x and you want to add 2 to the first element you'd type x(1) + 2. In Java (and also C / C++) indexing starts at 0, so the above line in Java would be written as x(0) + 2. In Matlab, if you try to use the array element x(0) you will get an error. In Java, if you have n elements in the array and try to reference element x(n) you will get an error – the highest possible reference being n-1. This is important, since some database references to data unit positions in files are written using the Java index, i.e. the indexing starts at 0. When programming in a mixture of languages it is essential to keep a close eye on these reference indexes since it may be necessary to add or subtract one depending on which language you are using.

Different programming languages store timestamps in different formats. Timestamps in Java are given as a number of milliseconds from 01/01/1970 stored as a 64 bit signed integer number. Matlab stores them as a number of days since 00/01/0000 using a double precision floating point number. There are advantages and disadvantages to both systems which will not be entered into here. Conversion between the two is relatively straight forward in Matlab or any other programming language.

# 5 Current Data Labelling in PAMGuard

## 5.1 The Database

PAMGuard uses a relational database to store certain types of data. The current version of PAMGuard supports MySQL and SQLite database formats. Microsoft Access can also be used with 32 bit versions of PAMGuard so long as the PC is running Java 7[1]. The obvious advantage of storing data in the database is that the data is generally 'human readable' and most other software (Matlab, R, etc.) can read data directly from the database. However, many types of data, particularly those which don't have a fixed record length, or which have very large numbers of records, cannot be efficiently stored in the database. The database is therefore used if:

1. The data have a fixed record format
2. The data rate is not expected to be high (more than a few records per second, max.)

A typical data type which is ideal for database storage is GPS data, which require a date, time, latitude, longitude, heading and speed.

*Table 1.Database columns present in all tables*

| Column Name | Format | Function |
|---|---|---|
| Id | Automatic integer | Unique database identifier assigned automatically by the database system |
| UTC | TIMESTAMP | Data date and time in UTC. Some database systems will include milliseconds in this time format, others only to the nearest second. |
| UTCMilliseconds | Short Integer | Number of milliseconds. This is always written even if the database has included the milliseconds in the UTC time stamp. |
| PCLocalTime | TIMESTAMP | Data date and time using the local time format. As with UTC this may or may not include milliseconds. |
| PCTime | TIMESTAMP | The time the data were written at in UTC. When data are collected in real time, this will be the same as (or very slightly later than) UTC. When data are processed offline, then this will be the time at which data were processed, whereas UTC and PCLocalTime above, will still be the time of the data itself. |

All PAMGuard database tables have some columns in common, as listed in Table 1. These include a timestamp with millisecond accuracy, and a unique database identifier. The database identifier format is

[1] Java 7 will no longer be supported within PAMGuard beyond the end of 2016.

controlled by the database, but most will use a 32 bit integer number allowing for approximately 2 billion unique ID's. Generally, a database will be limited by other size constraints before the unique id limit is reached. For each data stream, the PAMGuard software then supplements the standard columns with additional columns containing data specific to each data type.

### 5.1.1 Structured Query Language

Data from databases are generally selected and read into software using Structured Query Language (SQL). The syntax of SQL commands is similar across programming languages and database types, although some details (such as timestamp formats) may vary slightly. Plenty of texts and online resources are available describing SQL syntax. Specific examples for Matlab are available in the sourceforge PAMGuard Matlab repository.

## 5.2 The Binary Store

Much of the data produced by PAMGuard fits poorly into a database format. For example, when a click is detected we store a short waveform clip, typically a few hundred samples long. The waveform clip is stored alongside appropriate metadata such as the start time of the click (both in milliseconds and in samples) and other assigned parameters such as its species id. Similarly, detected whistle contours store the time frequency contour and bearing information for each detected whistle. The length of each click or whistle is not predetermined, so the total amount of data stored for each unit of data varies. While it would be possible to create a database schema which could handle both the waveform or contour and the click or whistle metadata, it would likely require more than one table, with appropriate cross referencing between tables. This would be hopelessly inefficient both in terms of data write / read times and also storage space.

Most PAMGuard detector data is therefore stored in data files written in a binary data format. Although PAMGuard is written in Java, the binary data format does not use any Java specific formats (i.e. Java serialisation). This means that PAMGuard binary files may be read by any program capable of opening a file and reading data from it (e.g. C, C++, Matlab, R, etc.).

Binary files end with .pgdf for PAMGuard Data File. A common file name format is used for all PAMGuard module output: file names are made up from the module type, the module name and the data stream name plus the date in a YYYYMMDD_HHMMSS time format. For example, a click detector file name might read Click_Detector_Beaked_Whales_Clicks_20150825_032012.pgdf, i.e. a PAMGuard "Click Detector" module, called "Beaked Whales" with an output data stream "Clicks" which was created at 03:20:12 UTC on 25 August, 2015.

Each pgdf file contains the following blocks of data:

1. A general header which has the same format for all data streams.
2. A module specific header (optional) which may contain module specific configuration data.
3. Data objects. These can be of more than one type and there may be any number of them.
4. A module specific footer (perhaps giving summary data for that module over the duration of the file)
5. A general footer with information such as the data end time. This is the same for all modules.

Both the general and the module header contain version numbers which enable us to change the format over time. There will always be backwards compatibility with older data types, however new data created with new PAMGuard versions may not open with older versions. Beginning with PAMGuard version

1.15.04, warnings will be issued if you attempt to open binary files created with a later PAMGuard version. For example, if you collected data with version 2.00.00 (which does not exist yet, but may use a slightly different file format) and attempt to open those files with PAMGuard 1.15.03 uncontrolled errors will occur. If you attempt to open those files with version 1.15.04, PAMGuard will still not be able to read the files, but warnings will be issued telling you to upgrade your PAMGuard version.

Output (from PAMGuard) uses only sequential file access (rather than random access), although other programs could of course open the files in any way they wish. This means that the file headers contain the file start time, but not the file end time, length or number of data objects which are only encoded in the file footer. To speed up data indexing when dealing with large data sets, the headers and footers (items 1,2 4 and 5 in the above list) are also written into files ending with .pgdx which have the same name as the .pgdf files. These are used for mapping and finding data when using the PAMGuard viewer. During data analysis using the PAMGuard Viewer, a data structure known as a "datagram" may also be added to the pgdx index file. Datagrams are used in the PAMGuard viewer to provide a visual picture of data density over time.

Details of the binary file structure are available in Table 2. As with the database, each item starts with a common structure which is then followed by module specific information. The general format for each data item is shown in Table 2. Note that the "Object Identifier" is a constant for each type of data and is NOT a unique identifier to individual data units.

*Table 2. General format for binary data objects.*

| Version | Item | Format | Notes |
|---|---|---|---|
| 0 | Length in File | Int32 | Length of this object |
| 0 | Object Identifier | Int32 | Positive integer (can be 0) which must be unique within this data stream, not across PAMGuard |
| 0 | Time milliseconds | Int64 | Timestamp in milliseconds |
| 2 | Time nanoseconds | Int64 | Additional nanosecond resolution time stamp |
| 0 | Object binary length 'l' | Int32 | = Length in File – 20 (-24 for V2) |
| 0 | Object Data | Byte[l] | Length = Object binary Length. Need not be same as Object 1. |

It can be seen from Table 2 that data units in the binary store don't have a unique id that is in any way analogous to the unique data 'Id' field in all database tables. Instead data are generally referred to by the name of the file they are in and their position within that file.

## 5.3 Cross Referencing Data

Within PAMGuard, data are sometimes cross referenced between the database and the binary store. The current main example of this is the marking of click events using the PAMGuard Viewer (also available for online click event marking from version 1.15.00, autumn 2016). For each marked event, data are written to two tables: the **Event** table and the **Clicks** table. The **Event** table contains one record per event, giving basic information such as the start time, number of clicks, etc. as well as localization information if a click train has been localized using target motion analysis. The **Clicks** Clicks~ table contains a list of clicks which form part of that event. These are cross referenced back to the **Events** table by a link between the **Clicks** EventId column and the **Event** Id column. The ClickNo column indicates the order (or index number) in which that particular click was saved to the binary file. For example, Figure 1 shows tables describing two marked events in the click detector containing 6 and 10 clicks respectively. The EventId column in the **Clicks** table cross references to the Id column in the **Event** table. Clicks in the binary files can be located based on the information in the BinaryFile and the ClickNo columns of the event table.

What should also be apparent from Figure 1 is that not all clicks in the binary file are included in the database. The first 6 clicks (0 – 5) in the binary file are included in event Id 1, then there is a gap and clicks 12, 15, 16, etc. are included in event Id 2. Clicks 6-11 in the binary file must not have been associated with a click event, and therefore never saved to the database.

## 5.4 Reading Data into Matlab

Say you want to read all of the click waveforms for a set of events into Matlab, for example to perform additional click classification processing based on an analysis of their waveforms. You would have to perform the following tasks:

1. Query the events table using SQL to determine event IDs for events of a particular type (e.g. with > 10 clicks), or write a list of events of interest.
2. For each event ID, construct a new SQL query string to select clicks associated with that event, e.g. to select clicks for event Id=anEventId in Matlab you would write

A) Event table

| | Id | UTC | UTCMilliseconds | PCLocalTime | PCTime | EventEnd | eventType | nClicks |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2016-08-11 14:35:46.989 | 989 | 2016-08-11 15:35:46.989 | 2016-08-11 14:35:52.916 | 2016-08-11 14:35:51.136 | NULL | 6 |
| 2 | 2 | 2016-08-11 14:35:54.907 | 907 | 2016-08-11 15:35:54.907 | 2016-08-11 14:36:03.212 | 2016-08-11 14:36:02.756 | NULL | 10 |

B) Clicks table

| | Id | UTC | UTCMilliseconds | PCLocalTime | PCTime | EventId | BinaryFile | ClickNo |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2016-08-11 14:35:46.989 | 989 | 2016-08-11 15:35:46.989 | 2016-08-11 14:35:50.754 | 1 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 0 |
| 2 | 2 | 2016-08-11 14:35:47.807 | 807 | 2016-08-11 15:35:47.807 | 2016-08-11 14:35:50.755 | 1 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 1 |
| 3 | 3 | 2016-08-11 14:35:48.634 | 634 | 2016-08-11 15:35:48.634 | 2016-08-11 14:35:50.756 | 1 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 2 |
| 4 | 4 | 2016-08-11 14:35:49.457 | 457 | 2016-08-11 15:35:49.457 | 2016-08-11 14:35:50.756 | 1 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 3 |
| 5 | 5 | 2016-08-11 14:35:50.282 | 282 | 2016-08-11 15:35:50.282 | 2016-08-11 14:35:50.766 | 1 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 4 |
| 6 | 6 | 2016-08-11 14:35:51.136 | 136 | 2016-08-11 15:35:51.136 | 2016-08-11 14:35:50.768 | 1 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 5 |
| 7 | 7 | 2016-08-11 14:35:54.907 | 907 | 2016-08-11 15:35:54.907 | 2016-08-11 14:35:59.126 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 12 |
| 8 | 8 | 2016-08-11 14:35:56.249 | 249 | 2016-08-11 15:35:56.249 | 2016-08-11 14:35:59.127 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 15 |
| 9 | 9 | 2016-08-11 14:35:57.062 | 62 | 2016-08-11 15:35:57.062 | 2016-08-11 14:35:59.127 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 16 |
| 10 | 10 | 2016-08-11 14:35:57.884 | 884 | 2016-08-11 15:35:57.884 | 2016-08-11 14:35:59.127 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 18 |
| 11 | 11 | 2016-08-11 14:35:58.700 | 700 | 2016-08-11 15:35:58.700 | 2016-08-11 14:35:59.127 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 19 |
| 12 | 12 | 2016-08-11 14:35:59.510 | 510 | 2016-08-11 15:35:59.510 | 2016-08-11 14:35:59.137 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 20 |
| 13 | 13 | 2016-08-11 14:36:00.326 | 326 | 2016-08-11 15:36:00.326 | 2016-08-11 14:35:59.888 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 21 |
| 14 | 14 | 2016-08-11 14:36:01.139 | 139 | 2016-08-11 15:36:01.139 | 2016-08-11 14:36:00.647 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 23 |
| 15 | 15 | 2016-08-11 14:36:01.951 | 951 | 2016-08-11 15:36:01.951 | 2016-08-11 14:36:01.571 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 27 |
| 16 | 16 | 2016-08-11 14:36:02.756 | 756 | 2016-08-11 15:36:02.756 | 2016-08-11 14:36:02.454 | 2 | Click_Detector_Click_Detector_1_Clicks_20160811_143546.pgdf | 30 |

*Figure 1. Click Event Database Tables. Two events are shown, containing 6 and 10 clicks respectively. The second table shows detailed information identifying each click.*

```
sqlString = sprintf('select * from ClicksTable WHERE EventId = %d
ORDER BY UTC, UTCMilliseconds', anEventId)
```

3. Once that query had been executed you would have a list of data containing the file names for each data and the click numbers in each file. Since a single event may straddle multiple binary files, the next task would be to identify how many unique files are involved, then for each of those files:
    a. Identify which clicks are associated with that file
    b. Read data from that binary file (there are pre-written Matlab functions to do this)
    c. Select the clicks from that file that you want for this event (remembering that the indexes from the database will start at 0 and that Matlab indexing starts at 1)
    d. Concatenate all the data from all the files together to make an array of PAMGuard click data for that event
4. Process those data however you want
5. Move on to the next event.

## 5.5 What's good about the current system

The current system is simple and it works.

The database Id's use the guaranteed uniqueness of the database indexing system to label data and data from the database are easily identified and updated.

For the binary files, including the file name in any cross reference information makes it easy to immediately know which file needs to be opened and by having the index of a particular data unit, earlier units in the file can be quickly and efficiently skipped during file reading.

## 5.6 What's bad about the current system

Data units within the binary files need to be identified both by a String (the file name) and by a number (the index within the file). For programmers this is inconvenient and many string comparisons can be time consuming when processing lots of data. For an analyst making notes about data they are working on, writing down both a file name and an index number can be inconvenient.

Another issue is that data cannot be added or removed from binary files, or the order of data within files altered in any way. Data units are only identified by their position in the file, so change that and everything breaks.

There is no direct correspondence between data ID's in the binary files and the database ID's, so relating data in the database to data in a binary file can require querying multiple database columns.

While some database tables contain cross reference information into binary files, this is not included for all database tables. Furthermore, the text field required for the file name uses considerable storage space for every record when compared to a single unique identifier number. Conversely, database indexes are not stored in the binary store, so it is currently almost impossible to find a database line for a given binary file object (though the opposite is possible if the database table contains the binary file information).

# 6 Alternatives and Options for Improving Data Referencing in PAMGuard

## 6.1 Improve the Status Quo

While complex to understand (partly no doubt through lack of documentation) the current system of data identification works. There is also a considerable body of Matlab software for reading data both from the databases and from the binary files. This software however is poorly maintained, even more poorly documented and it's possible that the software available on the PAMGuard sourceforge site is not up to date.

There are therefore a number of improvements which could be made which would make referencing data easier for the user without making changes to the core PAMGuard code:

1. Provide better documentation on both the binary file and database structures. This would take the form of documents on the PAMGuard website.
2. Improve and better document the existing Matlab code for reading from the database and binary files, along with improved example files and a tutorial available on the PAMGuard website.
3. Develop a similar library for R or any other programming language of interest to users (e.g. C/C++).
4. Where appropriate improve PAMGuard displays to show the existing cross referencing information, either on displays themselves or in "Tool Tip Texts" which appear when the mouse is hovered over data on the PAMGuard displays.

### 6.1.1 Pros

Clearly there are advantages to not modifying the current PAMGuard storage systems, particularly since any changes would have to be applied retrospectively to existing data and any converted data would possibly no longer be compatible with earlier PAMGuard versions.

### 6.1.2 Cons

Maintaining the bad points outlined in section 5.6.

### 6.1.3 Programming Tasks

Maintaining the status quo involves no changes to the current PAMGuard Java code except for the small changes outlined above and in section 7. Programming effort would however be required to improve on current Matlab and R code libraries for other analysis.

## 6.2 Unique numeric ID's in the binary files

The second options discussed here is the addition of a new system of unique identifiers (UID's) which would be added to both the binary files and to the database.

There are several ways in which additional UID's could be added to binary files. One option might be to add the database ID's to the binary data structures, exploiting the ability of the database to generate unique ID's. This approach would however be flawed for several reasons. Firstly, not all binary data are written to the database either because a data stream is not associated with the database at all, or only a subset of data are included in the database. Secondly PAMGuard is often run without the database even present.

A better alternative would be to generate separate UID'sfor every data unit. These would probably take the form of a 32 bit integer value, allowing for up to 2 billion UID's for each data stream. The standard header structure (Table 2) for each binary data unit would be modified to include the new UID and an

additional column would be added to each database table. This database column would NOT use automatic numbering in order to ensure that PAMGuard can write the number it has used for the binary files.

### 6.2.1 Pros

Adding additional integer identifiers to data would make it easy to identify and cross reference data in both the binary files and in the database. Database queries would be fast and efficient, numeric comparisons in code or in an analyst's head would be straight forward and simple. Additional storage needs within files would be very modest, not requiring significant storage space or slowing the reading and writing of data from disc.

### 6.2.2 Cons

#### *6.2.2.1 Ease of use*

Datasets with millions of data points would inevitably have large UID's. Noting 7 or 8 digit numbers may not be easier than noting a file name and a smaller index number within that file.

Even knowing a unique id, it would not be directly possible to know which binary file the data had come from, or the position of the data within that file. Within PAMGuard, the existing file name / position within the file system would have to be retained, although the addition of a UID check would make this system more robust. First and last UID's for each stream would also have to be held in the binary file headers which are mapped within PAMGuard to enable quick searching. A similar datamap to enable rapid data retrieval could be developed for binary files. The system of cross referencing and extracting data from multiple tables described in section 5.3 would remain pretty much the same as it is at the moment.

#### *6.2.2.2 File compatibility*

We would need to modify PAMGuard in such a way that if an old dataset were opened the dataset would be updated (i.e. all binary files rewritten and the database also updated) to include the new UID's. This process might take considerable time for large datasets and altering the PAMGuard binary file format would mean that files written with the latest PAMGuard version could no longer be opened using earlier versions. From Version 1.15.04 some protection has been included in PAMGuard so that it will not even attempt to read files with a later version number, earlier versions of PAMGuard (1.15.03 and earlier) might attempt to read the file and could possibly corrupt it.

#### *6.2.2.3 Storing and Resetting UID's*

A significant issue is the persistent storing and possible resetting of UID's. Clearly, when PAMGuard exits, it would be possible to write a file containing the latest UID for each data stream and this file could be re-read when PAMGuard is restarted in order to continue with the next available UID. What would happen however when PAMGuard crashes? In that case, the UID file would not have been rewritten, and so the UID reference list would not be up to date. Alternatively, rewriting the list every time data were created would be too time consuming and would impact PAMGuard performance. An alternative would be to have PAMGuard always read the last binary file in any dataset each time it starts and extract the latest UID information. This would be more robust, but would also create a delay whenever PAMGuard started (depending on the size of the last file). A combination of the above two methods might also be possible:

1. Write a file of UID's when PAMGuard exits properly
2. When PAMGuard starts, read the file if it exists, then delete it immediately. Deleting the file will ensure that it cannot be read in the event of PAMGuard crashing and the file being out of date.

3. If the UID file doesn't exist, then assume that PAMGuard crashed and read the last binary file to search for the UID (first and last UID's would be included in file headers and footers so that they could be read even if the file contained no actual data).
4. For a new dataset, no UID file would be found and no binary data files would exist, so UID's would be created at zero.

This might form a basis for a simple, fast and adequately robust UID system.

### 6.2.3 Programming Tasks

Considerable changes to the PAMGuard core software would be needed to implement these changes. In particular:

1. Implementation of the UID system for new PAMGuard data
    a. Persistent and safe storage and generation of UID's
    b. Addition of the UID to the standard database columns
    c. Addition of the UID to data headers in all binary data files
2. Updating of existing data sets
    a. Identifying that an existing dataset was created with an earlier PAMGuard version
    b. Reading and rewriting all binary data files with a new UID (overwrite or create new binary store options would be required)
    c. Adding additional columns to existing database tables and populating with new UID's
    d. Where data exist in both the database and binary files, ensure that both have the same UID's (this is possible where a database table contains cross references back into the binary files, but may not be possible for some datasets)
3. Update and make available existing Matlab binary file reading code to read the additional UID fields from files.

## 6.3 Cross Referencing based on timestamps

The third option discussed is to cross reference purely on the timestamp of each data unit. Java uses an integer number of milliseconds which can be compared relatively simply and can also be rapidly translated into a printable data in any programming language. We are also in the process of investigating a nanosecond timestamp for PAMGuard in addition to the Millisecond time. This would use the same reference time (1/1/1970) as the millisecond stamp, so numbers would be 1000000 time larger than the millisecond times. A 64 bit number of nanoseconds can stretch for 292 years either side of 1970 before reaching the limits of 64 bit integer numbers.

### 6.3.1 Pros

Simple and easy system to understand.

### 6.3.2 Cons

The most important disadvantage of this system is that there is no guarantee that the ID's would be unique. While it is unlikely that two calls of the same type would be received within this time interval, it is not impossible for high frequency data (e.g. porpoise clicks when multiple animals are present). Often a PAMGuard detector will detect on multiple channel groups, in which case the same call will be detected separately on different hydrophones and written to the same store. In this case there is the very clear possibility of detections having the same millisecond or nanosecond time and therefore the same ID.

Querying the database would also be difficult since many database systems have to store the second time and the millisecond time in separate columns and do not support storage of 64 bit numbers.

### 6.3.3 Programming Tasks

This option does not seem realistic for implementation in PAMGuard so detailed programming tasks have not been considered.

# 7 GUI Improvements

It is not uncommon for an analyst to be using the PAMGuard viewer to process data and for them to be working on additional analysis routines in proprietary software. In viewer mode, on the PAMGuard GUI some data will show additional information when the mouse is hovered over particular data. How this information is presented can be a bit haphazard. Immediate improvements could be made to this whatever the adopted cross referencing system by ensuring that the UID information, in whatever form it takes, is clearly visible in every one of these information boxes.