DFO “Team Whale” Beaked Whale Detector  
Reference Manual

Version 1.0

Wilfried Beslin, June 2019

Table of Contents

[1. INTRODUCTION 3](#_Toc10540781)

[2. BEAKED WHALE DETECTION ALGORITHM 3](#_Toc10540782)

[2.1 Definition of a Detection Event 3](#_Toc10540783)

[2.1.1 Data Segmentation 4](#_Toc10540784)

[2.1.2 Event Merging 4](#_Toc10540785)

[2.1.3 Continuous vs. Duty-Cycled Data 4](#_Toc10540786)

[2.2 Detection Criteria 5](#_Toc10540787)

[2.2.1 Click Discrimination Criteria 5](#_Toc10540788)

[2.2.2 Event Detection Criteria 5](#_Toc10540789)

[2.2.3 Event-Level and Validation-Level Assessments 6](#_Toc10540790)

[2.2.4 Defining Detection Criteria 6](#_Toc10540791)

[2.3 Detection Protocols 7](#_Toc10540792)

[2.3.1 Defining a Detection Protocol 7](#_Toc10540793)

[2.3.2 Using Multiple *ClickDiscrimParams* files 8](#_Toc10540794)

[2.3.3 Example Protocols 9](#_Toc10540795)

[3. MANUAL EVENT VALIDATION AND SPECIES IDENTIFICATION 12](#_Toc10540796)

[3.1 Summary of Validation Process 12](#_Toc10540797)

[3.2 Species ID Codes 15](#_Toc10540798)

[3.3 Interactive Click Discrimination 16](#_Toc10540799)

[3.4 Other features of the interactive Code 17](#_Toc10540800)

[3.4.1 Limit on Number of Clicks 17](#_Toc10540801)

[3.4.2 Switching Events 18](#_Toc10540802)

[3.4.3 Quitting and Resuming 18](#_Toc10540803)

[3.4.4 Saving, Crashes and Backups 18](#_Toc10540804)

[4. RUNNING THE CODE 19](#_Toc10540805)

[4.1 Code Directory Structure 19](#_Toc10540806)

[4.2 Input and Output 20](#_Toc10540807)

[4.2.1 WAV Files 20](#_Toc10540808)

[4.2.2 Triton cTg Files 21](#_Toc10540809)

[4.2.3 MAT Files 21](#_Toc10540810)

[4.2.4 Results Folder 22](#_Toc10540811)

[4.2.5 DetectionProtocol.txt File 22](#_Toc10540812)

[4.2.6 “RawEvents” Spreadsheet Files 22](#_Toc10540813)

[4.2.7 “Validated” Spreadsheet Files 23](#_Toc10540814)

[4.2.8 “Presence” Spreadsheet Files 24](#_Toc10540815)

[4.2.9 JPEG Files and Plot Folders 24](#_Toc10540816)

[4.3 “master” Interface Scripts 25](#_Toc10540817)

[4.3.1 BWD\_master.m 25](#_Toc10540818)

[4.3.2 identifySpecies\_master.m 25](#_Toc10540819)

[4.4 Additional Scripts 26](#_Toc10540820)

[4.4.1 Converting Species ID Codes 26](#_Toc10540821)

[4.4.2 Isolating and Revisiting Events 27](#_Toc10540822)

[APPENDIX A: IMPLEMENTING NEW DETECTION PARAMETERS 28](#_Toc10540823)

[Very Brief Introduction to Object-Oriented Programming in MATLAB 28](#_Toc10540824)

[Object-Oriented Handling of Detection Criteria 30](#_Toc10540825)

[How to Add New Parameters 33](#_Toc10540826)

[APPENDIX B: THE PACKAGE SYSTEM 34](#_Toc10540827)

[APPENDIX C: COMMENTS ON THE CODE AND IDEAS FOR THE FUTURE 35](#_Toc10540828)

[APPENDIX D: CAVEAT ON SAMPLING RATE 36](#_Toc10540829)

[APPENDIX E: GLOSSARY OF TERMS AND ACCRONYMS 37](#_Toc10540830)

# 1. INTRODUCTION

Our current system for detecting beaked whales from AMAR recordings depends on two MATLAB-based programs. The first is Triton, which is a comprehensive package developed by folks at the Scripps Whale Acoustic Lab for analyzing data from autonomous acoustic recorders (<http://cetus.ucsd.edu/technologies_Software.html>, though our version is not available for download here). The second is a custom beaked whale detection software, tentatively referred to as BWD (Beaked Whale Detector). This code is based on a series of scripts originally written by Simone Baumann-Pickering to detect beaked whales off California from HARP recordings, which were later modified by Joy Stanistreet for use with Atlantic AMAR data. These scripts have since been redeveloped by Wilfried Beslin into a more streamlined and versatile package.

The beaked whale detection process essentially consists of 3 steps:

1) General click detection using Triton

2) Automatic detection of beaked whale acoustic events using BWD

3) Manual validation of beaked whale events and species identification using BWD

The purpose of this document is to thoroughly describe how the new BWD works. For a step-by-step walkthrough of the full beaked whale detection procedure including Triton, refer to the “Walkthrough” document that complements this one.

# 2. BEAKED WHALE DETECTION ALGORITHM

At its core, BWD’s approach to detecting beaked whales is very simple: features extracted from individual clicks are evaluated against a set of threshold values to determine if the clicks have characteristics of beaked whale clicks. Furthermore, while BWD was developed with north Atlantic beaked whales in mind as a whole, the algorithm is general enough that it could be applied to many subjects - including individual species, groups of species, and potentially even non-beaked whales. Therefore, subjects of interest will generally be referred to as “targets” in this manual.

This section describes in detail the concept behind automatic detection in BWD, how it works, and how to use it for detection of particular targets.

## 2.1 Definition of a Detection Event

*Events* are blocks of time within which there is enough evidence to suggest that a target was present at some point. What constitutes enough evidence is defined by the *detection criteria*. Detection criteria will be discussed further in subsection [2.2 Detection Criteria](#_2.2_Detection_Criteria), but fundamentally, they operate on clicks detected within the time series. If there occurred enough clicks anywhere within a given period of time that were likely produced by the target, then that period is considered to be an event.

### 2.1.1 Data Segmentation

BWD operates on the basis of time *segments*, which form the basic units that may be considered detection events. Users have the option to divide audio recordings into segments of their choosing, e.g. 1 minute. If a recording cannot be divided perfectly into equal-length segments, then its last segment will simply be shorter. If no segmentation is specified, then whole recordings will be considered segments.

When BWD runs automatic event detection, each segment is evaluated independently. Those segments that contain enough potential target clicks will be considered events, at least initially. The location of target clicks within each segment is not taken into account, except when computing inter-click intervals during the manual validation process.

### 2.1.2 Event Merging

After events have been detected on a per-segment basis, it is possible to merge them and create new, more useful definitions for the start/end time of a detection event. For example, revised events might constitute full calendar days (i.e. from hour 00:00:00 - 23:59:59) , or they might consist of consecutive segments that met the criteria for a detection event. When redefining event start/end times this way, the new events will be those periods of time that include at least one segment (or part of a segment) that was deemed an event. Thus, for example, if redefining events as calendar days from 1-minute segments, a full 24-hour period will be considered an event even if the target was confirmed present for just one minute within that day.

The available options for merging events are:

* **none** – no merging. Raw segment durations are used as the basis for event durations.
* **timegap** – segments with positive detections that occur within a certain time period of one another will be merged into single large events. This is most useful for continuous data. For example: if using a *timegap* of 3 minutes, and you have one detection that occurred from 02:01:00-02:02:00 and another detection from 02:04:00-02:05:00, then the two detections will be merged into one event lasting from 02:01:00-02:05:00.
* **calendar** - use fixed calendar durations (e.g. days, weeks, etc.) as the basis for event durations. A given calendar period will be considered a detection event if it contains at least one segment with beaked whale detections.

### 2.1.3 Continuous vs. Duty-Cycled Data

One advantage (and in fact the motivation for) using a data segmentation system is that it allows BWD to handle both continuous and duty-cycled data in the same way. With the segmentation framework, duty-cycled data essentially become something of a “special case” of the continuous scheme - the only difference is that duty-cycled data will necessarily have no event detections within the “off” periods.

In principle, users can divide both continuous and duty-cycled recordings into segments of their choosing, but it is generally useful to do this only for continuous data. If a segmentation period is not specified, then the routine will use whole recordings as segments. Thus, for duty-cycled data, this means that each recording will become its own segment, with the time periods in-between simply being ignored.

## 2.2 Detection Criteria

Detection criteria are a set of threshold-based rules used to ultimately determine the presence of a particular target. There are two types of detection criteria: *Click Discrimination Criteria* and *Event Detection Criteria*.

### 2.2.1 Click Discrimination Criteria

Click discrimination criteria evaluate the properties of individual clicks and decide if the clicks may correspond to the target or not. The list of criteria available in BWD at present is shown in [Table 2.1](#Table_2_1) (should you ever wish to implement new ones, see [Appendix A](#_APPENDIX_B:_IMPLEMENTING)).

|  |  |
| --- | --- |
| **Table 2.1**: list of features currently implemented in BWD for click discrimination. Lesser-used or legacy features (i.e. those maintained only for replication of SBP/JS’s version of the code) are in gray. All those that were used for discriminating beaked whales in the original code are denoted by an asterisk (\*). | |
| **Criterion** | **Description** |
| Fpeak\* | Peak frequency (kHz) |
| F0\* | Centroid frequency (kHz) |
| bw10db | Magnitude of 10 dB bandwidth (kHz) |
| bw10dbLower | Lower frequency of 10 dB bandwidth (kHz) |
| bw10dbUpper | Upper frequency of 10 dB bandwidth (kHz) |
| ZCR | Zero crossing rate (1/ms) |
| dur\* | Click duration, based on Teager-Kaiser estimate (ms) |
| slope\* | Slope of line fitted through peak frequencies over time (-8 dB bandwidth, kHz/ms) |
| slopeDur | Duration within which slope was measured; corresponds to -8 dB about the time period where peak frequency was most intense (ms) |
| durE50\* | Duration within which relative amplitude of the click Hilbert envelope was greater than 50% (ms) |
| bw3db | Magnitude of 3 dB bandwidth (kHz). Seems less useful than 10 dB. |
| bw3dbLower | Lower frequency of 3 dB bandwidth (kHz). Seems less useful than 10 dB. |
| bw3dbUppper | Upper frequency of 3 dB bandwidth (kHz). Seems less useful than 10 dB. |
| deltaE\* | Legacy feature. Determines whether max envelope energy is increasing or decreasing between the time intervals [0,0.1] and [0.125,0.35] (milliseconds); decrease supposedly suggests dolphin clicks. This feature does not support variable threshold values – leave them blank. |
| nSamples\* | Legacy feature. This is the duration within which slope was measured (-8 dB about peak frequency), in number of samples. *slopeDur* is the sampling rate-invariant version of this, and therefore is preferred over *nSamples*. |

### 2.2.2 Event Detection Criteria

Event detection criteria assess the number of candidate target clicks in a segment (as determined by click discrimination criteria), and decide if a detection event has occurred or not. The list of event detection criteria presently available in BWD is shown in [Table 2.2](#Table_2_2).

|  |  |
| --- | --- |
| **Table 2.2**: list of criteria currently implemented in BWD for event detection (all of these were effectively implemented in the original SBP/JS code too) | |
| **Criterion** | **Description** |
| MinNumBeaked | Minimum number of candidate target clicks in a segment |
| MinPercentBeaked | Minimum percentage of all clicks detected in a segment that are candidate target clicks |

### 2.2.3 Event-Level and Validation-Level Assessments

There are two levels of assessment where detection criteria are applied, called event-level and validation-level. Event-level assessment describes the automatic detection of beaked whale events, and involves both click discrimination criteria and event detection criteria. Validation-level assessment decides which clicks to present to a user when validating events. Only click discrimination criteria are relevant here.

Click discrimination criteria need not be the same between event-level and validation-level assessments. At the event-level, precision is typically more important, since only a minimum number of target clicks are required to trigger an event (dependent on event detection criteria). For the validation-level however, it is generally acceptable to tolerate lower precision in favour of recall, since the user can recognize and ignore false positives (though there are exceptions, e.g. Cuvier’s beaked whales in the Gully). Furthermore, it is possible to use different sets of click discrimination criteria between the event-level and validation-levels, to allow e.g. general event detection followed by species-specific validation. This will be discussed further in subsection [2.3 Detection Protocols](#_2.3_Detection_Protocols).

### 2.2.4 Defining Detection Criteria

Detection criteria for a particular target are specified using Microsoft Excel spreadsheet files (.*xlsx* only, not *.xls*). Event detection criteria and each set of click discrimination criteria all use their own spreadsheet. For click discrimination, the first sheet must contain the following headings (order and capitalization are important):

*Criterion*, *Threshold1*, *Threshold2*, and *UseCategory*

For event detection, the headings are:

*Criterion*, *Threshold*, and *UseCategory*

In both cases, the *Criterion* column must include the names of those criteria listed in the tables above, as appropriate. The *Threshold* columns contain threshold values for each criterion. In the case of click discrimination, thresholds 1 and 2 correspond to minimum and maximum thresholds, respectively. For event detection, only a minimum threshold is used. If you want to effectively ignore a particular threshold (for example, if maximum is not important), set *Inf* for maxima, or -*Inf* for minima.

The *UseCategory* column can be used to decide if a criterion should be assessed or not: simply set to 1 to use a criterion, or 0 to ignore it. However, *UseCategory* also allows integers greater than 1. This involves an advanced feature called *MissTol* (miss tolerance), which allows greater flexibility when confirming a detection event or target click. *MissTol* is controlled using the second sheet in the spreadsheet file.

Each criteria spreadsheet must have a second sheet, containing the columns *Category* and *N*. The *Category* column must list all integers present in the *UseCategory* column of the first sheet (except zero). For each category, *N* determines how many criteria of that category may be ignored. For example, assume you define a click discrimination spreadsheet with the following properties:

* 7 criteria have *UseCategory* = 1
* 3 criteria have *UseCategory* = 2
* *N* = 0 for category 1
* *N* = 2 for category 2

In this case, a click is considered to “pass” only if it meets all seven category 1 criteria, and at least one of the three category 2 criteria.

Event detection and click discrimination spreadsheets must follow a very specific file naming convention and placement to be read properly by BWD. This will be discussed in the next section.

## 2.3 Detection Protocols

The collection of detection criteria used in the process of assessing presence/absence of one or more targets is referred to as a *detection protocol*. Since goals and conditions vary between analyses, it may be desirable to implement unique protocols for different situations. For example, in most regions around the Scotian shelf edge where various beaked whale species are occasional visitors, it is often appropriate to follow a simple detection protocol, where click discrimination at both levels is concerned only with separating beaked whales from non-beaked whales. However, this is much less effective in situations like the central Gully, where northern bottlenose whales are omnipresent and confound the detection of rarer species. In this case, it is more effective to use a protocol that separates northern bottlenose whale detections from those of other species.

### 2.3.1 Defining a Detection Protocol

Protocols are defined and interpreted by BWD using a hierarchical file structure and specific naming conventions. The root directory of each protocol should be located in the *DetectionCriteria* folder in the code directory.

The hierarchy for a protocol is as follows:

*Protocol* folder > *Target* folders > detection criteria spreadsheets

The *Protocol* and *Target* folders can have unique descriptive names, which will be used by BWD to control input/output. I/O will be described further in section [4.2 Input and Output](#_4.2_Input_and).

A *Target* corresponds to a species or group of species of interest. For example, a *Target* might encompass all beaked whales (e.g. “Beaked”), just bottlenose whales (e.g. “Ha”), or all beaked whales except bottlenose whales (e.g. “NonHa”). When running automatic detection in BWD, all *Targets* will be treated independently and will have their own output files.

Within each *Target* folder is a collection of detection criteria spreadsheet files. Each of these files must adhere to the following naming convention:

***CritType***\_***Level***\_***Name***.xlsx

Where:

* ***CritType*** designates the type of detection criteria. It **must be either**:
  + ClickDiscrimParams for click discrimination criteria
  + EventDetParams for event detection criteria
* ***Level*** designates the assessment level (event or validation). This is **only relevant for ClickDiscrimParams**. It **must be either**:
  + EventDet for event-level assessment
  + Validation for validation-level assessment
* ***Name*** is an identifier to help keep track of what the criteria are used for. It **can be anything,** and is **optional** unless you have multiple spreadsheets of the same type and level.

A *Target* folder must contain the following:

* **Exactly one** *EventDetParams* spreadsheet
* **At least one** *ClickDiscrimParams\_EventDet* spreadsheet
* **At least one** *ClickDiscrimParams\_Validation* spreadsheet

Finally, while not strictly necessary, it is good practice to include a *README.txt* file directly within the Protocol folder to explain what it does and what it should be used for.

### 2.3.2 Using Multiple *ClickDiscrimParams* files

With *ClickDiscrimParams*, it is possible to include more than one spreadsheet, for both the *EventDet* and *Validation* levels. This is useful, for example, if you want to discriminate clicks at a species-specific level, but still want to define events for all beaked whales as a whole. The effect of using multiple *ClickDiscrimParams* spreadsheets works as follows:

* If multiple *ClickDiscrimParams\_EventDet* files exist:
  + Each click must pass **at least one** set of criteria to be counted as a potential target click when detecting events
* If multiple *ClickDiscrimParams\_Validation* files exist:
  + The **user can choose** how each set of criteria should be used to filter out clicks during interactive event validation/species identification. Each set of criteria, or discriminator, can be used for **positive discrimination** (clicks are displayed if they meet criteria), **negative discrimination** (clicks are not displayed if they meet criteria), or they can be **disabled** (the criteria will not influence whether or not clicks are displayed). This will be revisited in section [3.3 Interactive Click Discrimination](#_3.3_Interactive_Click).

When multiple *ClickDiscrimParams\_Validation* files have been specified, an additional, optional spreadsheet named *ValidationDefaults.xlsx* can be included in the *Target* folder. This spreadsheet determines what the default discrimination scheme should be during event validation/species identification. This spreadsheet should have 2 columns: *File* and *DefaultSetting*.

* The *File* column should list the **full file names** of all *ClickDiscrimParams\_Validation* files, not including the *.xlsx* extension
* The *DefaultSetting* column specifies the default effect of each discriminator. The options are:
  + Positive
  + Negative
  + Disabled

If no *ValidationDefaults* file is specified, BWD will assume that the default scheme is for all discriminators to be Positive. In this scheme, clicks will be displayed only if they meet the criteria of at least one discriminator.

### 2.3.3 Example Protocols

This section illustrates what the file structure for particular detection protocols might look like. To enhance clarity, these examples will use the following colour and indentation code:

* *Protocol* directory
  + *Target* directory
    - *ClickDiscrimParams\_EventDet* spreadsheet
    - *ClickDiscrimParams\_Validation* spreadsheet
    - *EventDetParams* spreadsheet
    - *ValidationDefaults* spreadsheet

Note that these examples only serve to illustrate how protocols can be defined. Most of them have not actually been implemented. For information on actual protocols, refer to the *README.txt* files within each protocol folder.

*Example 1*

Assume you’re interested simply in detecting beaked whale events in general, and are not concerned about individual species. In this case, your protocol directory structure might look like this:

* General
  + Beaked
    - ClickDiscrimParams\_EventDet.xlsx
    - ClickDiscrimParams\_Validation.xlsx
    - EventDetParams.xlsx

Here, both *ClickDiscrimParams* files would define thresholds for discriminating all beaked whale clicks in general (this was effectively how SBP/JS’s version of the code worked).

*Example 2, part a)*

Now assume you still want to detect events for beaked whales as a whole, but would like to view clicks at a species-specific level during event validation. In this case, the protocol file structure might look like this:

* General
  + Beaked
    - ClickDiscrimParams\_EventDet.xlsx
    - ClickDiscrimParams\_Validation\_Ha.xlsx
    - ClickDiscrimParams\_Validation\_Mb.xlsx
    - ClickDiscrimParams\_Validation\_Me.xlsx
    - ClickDiscrimParams\_Validation\_Zc.xlsx
    - EventDetParams.xlsx

Here, *ClickDiscrimParams\_EventDet* would define thresholds for discriminating all beaked whale clicks in general, whereas each *ClickDiscrimParams\_Validation* file would define thresholds for clicks of individual species. The Names appended at the ends correspond to the initials of each species’ scientific names.

*Example 2, part b)*

There is another way in which general event detection could effectively be achieved:

* General
  + Beaked
    - ClickDiscrimParams\_EventDet\_Ha.xlsx
    - ClickDiscrimParams\_EventDet\_Mb.xlsx
    - ClickDiscrimParams\_EventDet\_Me.xlsx
    - ClickDiscrimParams\_EventDet\_Zc.xlsx
    - ClickDiscrimParams\_Validation\_Ha.xlsx
    - ClickDiscrimParams\_Validation\_Mb.xlsx
    - ClickDiscrimParams\_Validation\_Me.xlsx
    - ClickDiscrimParams\_Validation\_Zc.xlsx
    - EventDetParams.xlsx

In this case, clicks would be discriminated for each species at the event-level, but all clicks that meet criteria for at least one species would contribute towards detection of the event.

*Example 3*

Assume you are analyzing data from the Gully and are interested in separating northern bottlenose whale events from Sowerby’s and Cuvier’s beaked whales. In this case, the protocol might look like this:

* Gully
  + Ha
    - ClickDiscrimParams\_EventDet\_Ha.xlsx
    - ClickDiscrimParams\_Validation\_Ha.xlsx
    - EventDetParams\_Ha.xlsx
  + NonHa
    - ClickDiscrimParams\_EventDet\_Mb.xlsx
    - ClickDiscrimParams\_EventDet\_Zc.xlsx
    - ClickDiscrimParams\_Validation\_Mb.xlsx
    - ClickDiscrimParams\_Validation\_Zc.xlsx
    - EventDetParams\_NonHa.xlsx

Here, there are two targets: “Ha” and “NonHa”. This means that northern bottlenose whale events will be detected independently from those of the other two species: the output will consist of spreadsheets for Ha events, and spreadsheets for NonHa events. The Names at the end of each *EventDetParams* file are optional, but help to keep track of which target they belong to.

*Example 4*

Assume you are concerned only about detecting Cuvier’s beaked whales in the Gully. This is especially tricky for two reasons: 1) northern bottlenose whale clicks are usually far more numerous and conspicuous than Cuvier’s clicks in this area , and 2) there is a fair amount of overlap in the range of parameter values between the clicks of the two species. This means there could be an overwhelming number of false positives, both in terms of events and clicks to examine during validation. While there may be better approaches, one way to mitigate this problem would be to use the following protocol:

* Gully\_Zc
  + Zc
    - ClickDiscrimParams\_EventDet\_Zc.xlsx
    - ClickDiscrimParams\_Validation\_Ha.xlsx
    - ClickDiscrimParams\_Validation\_Zc.xlsx
    - EventDetParams.xlsx
    - ValidationDefaults.xlsx

Within *ValidationDefaults.xlsx*, the *DefaultSetting* would be “Negative” for *ClickDiscrimParams\_Validation\_Ha*, and “Positive” for *ClickDiscrimParams\_Validation\_Zc*. With this scheme, clicks during the interactive process will be displayed if they meet criteria for Cuvier’s (Zc) only. In other words, clicks that look like both Cuvier’s and Bottlenose (Ha) will be hidden.

# 3. MANUAL EVENT VALIDATION AND SPECIES IDENTIFICATION

Automatic event detection is useful for processing many hours of recording in a short amount of time, but it is not perfect. There will always be some amount of missed and/or false detections, depending on detection criteria and segment/event duration (longer events are less likely to miss targets, but more likely to include false alarms). Furthermore, while BWD does support some level of species-specific discrimination, there are enough similarities among the clicks of different beaked whale species that it cannot separate them perfectly, at least not using the current set of features and threshold system. This is more of an issue for some species than others: Sowerby’s beaked whale for instance is fairly easy to identify automatically, but Cuvier’s beaked whale can be very difficult to differentiate from northern bottlenose whales.

To improve the reliability of presence/absence analyses, automatically-detected events should be validated by an observer. To help facilitate this, BWD includes an interactive part where users can examine clicks within each event, and confirm which species are present. This section describes some of the features of this interactive process.

## 3.1 Summary of Validation Process

The interactive event-validation/species ID application presents events one at a time. For each event, four figures are presented which display the following:

* Plots summarizing the peak frequency, ICI, and mean spectrum of clicks within that event (see [Figure 3.1](#Figure_3_1))
* A plot comparing mean click spectrum with reference spectra derived from a library of clicks from various species (see [Figure 3.2](#Figure_3_2))
* The waveform and spectrogram of an individual click (see [Figure 3.3](#Figure_3_3))
* Information on parameter values and discrimination scores of the click (see [Figure 3.4](#Figure_3_4))

|  |
| --- |
| C:\Users\BeslinW\Documents\BeakedWhales\Documentation\Fig_Summary.png |
| **Figure 3.1**: Event summary plots.  Top left: distribution of peak frequencies. The frequency range at the top corresponds to the bin with the most occurrences (or the first of such bins, if there are more than one).  Top right: distribution of inter-click intervals. The ICI range at the top corresponds to the bin with the most occurrences (or the first of such bins, if there are more than one).  Bottom left: mean click spectrum, with peak frequency marked in red and -3 dB and -10 dB bandwidths marked in gray. The dotted line is the mean spectrum of noise samples.  Bottom right: concatenated spectrogram of click spectra, where clicks are sorted by lowest peak frequency. |

|  |
| --- |
| C:\Users\BeslinW\Documents\BeakedWhales\Documentation\Fig_Overlay.png |
| **Figure 3.2**: Spectral overlay plot. Mean click spectrum is shown in solid black, and reference spectra in other colours as noted in the legend. |

|  |
| --- |
| C:\Users\BeslinW\Documents\BeakedWhales\Documentation\Fig_Click.png |
| **Figure 3.3**: plot of the waveform (top) and spectrogram (bottom) of an individual click. The part of the title in brackets denotes the sorting scheme. |

|  |
| --- |
| C:\Users\BeslinW\Documents\BeakedWhales\Documentation\Fig_ClickParams.png |
| **Figure 3.4**: Display of parameter values and discrimination status for an individual click. Precise time of occurrence of the click is printed at the top. The table at the bottom indicates whether or not the click has passed each discriminator in the active detection protocol (validation-level). Those entries with a darker shade correspond to “disabled” discriminators. The coloured columns at the right in the middle table show how individual parameters meet discrimination criteria; note that this information is hidden by default to help maximize processing speed. |

Users have the option to change click discrimination settings to display different sets of clicks, cycle between plots of individual clicks, and confirm species presence with a numeric code (discussed in the following subsection, [3.2 Species ID Codes](#_3.2_Species_ID)). When beginning manual validation on a series of automatically-detected events, the application will create an Excel spreadsheet file with the tag *Validated* in the filename (to be discussed further in section [4.2.7 “Validated” Spreadsheet Files](file:///C:\Users\BeslinW\Documents\BeakedWhales\Documentation\Validated#_4.2.7_)). As each event is validated, the application will automatically fill out the spreadsheet with the species ID codes marked by the user.

Commands are entered via the MATLAB command prompt. The complete list of commands available is described in [Table 3.1](#Table_3_1).

|  |  |
| --- | --- |
| **Table 3.1**: list of commands available during interactive event validation/species identification. Press “Enter” after inputting a command code to execute it. | |
| **Command** | **Action** |
| *(None)* | Plot the next click. If you reach the last, it will loop back to the first. |
| d | Change direction of click plotting. Use this to go back to clicks you want to view again. |
| o | Change sorting order. This will switch between plotting clicks in order of highest peak-to-peak signal level, and in order of highest peak frequency. |
| r | Return to the first click in the sequence |
| s | Save a JPEG image of the click currently being displayed |
| f | Toggle display of parameter-specific discrimination scores in click parameter values window on/off |
| p | Edit click discrimination scheme |
| c | Add comment about the current event to the output spreadsheet. Entering this command will open a popup window where you can type in the comment. |
| u | Update the spreadsheet file on disk manually. While there is some level of auto-saving, It’s still good to do this often. |
| q | Close the application. Doing this will also update the spreadsheet automatically. |
| *(Numeric code)* | Confirm species ID for this event and move to next |

## 3.2 Species ID Codes

Species are assigned using a set of numeric codes, which are specified in an input spreadsheet file called *SpeciesCodes.xlsx*. The codes implemented at time of writing are listed in [Table 3.2](#Table_3_2). Note that these codes and species could be changed by editing *SpeciesCodes.xlsx*, however there must be codes for “Other” and “Uncertain”. Furthermore, only single-digit codes are currently supported.

|  |  |  |
| --- | --- | --- |
| **Table 3.2**: Current list of species ID codes, as defined in the file *SpeciesCodes.xlsx.* Species known to frequent the Scotian Shelf are bolded. | | |
| **Classification code** | **Species** | **Common name** |
| **1\*** | **Me - *Mesoplodon europaeus* Mm *- Mesoplodon mirus*** | **Gervais’ beaked whale True’s beaked whale** |
| **2** | **Zc - *Ziphius cavirostris*** | **Cuvier’s beaked whale** |
| 3 | Md - *Mesoplodon densirostris* | Blainville’s beaked whale |
| 4 | BW31 | probably Blainville’s |
| 5 | BW38 | probably Blainville’s (variant click type) |
| **6** | **Mb - *Mesoplodon bidens*** | **Sowerby’s beaked whale** |
| **7** | **Ha - *Hyperoodon ampullatus*** | **Northern bottlenose whale** |
| 8 | UBW - unknown beaked whale |  |
| 9 | Other | Not a beaked whale |
| 0 | Uncertain |  |
| \* Note that at present we cannot acoustically distinguish True’s beaked whale clicks from Gervais’ beaked whale clicks with any certainty. For now, code 1 should be used for this “Mm/Me” click type. | | |

The BWD package includes a script to interpret numeric species codes and convert them into binary presence/absence tables (see section [4.4.1 Converting Species ID Codes](#_4.4.1_Converting_Species)). However, for this to work, the codes must follow a certain set of rules. These rules are not enforced when IDs are entered interactively, so it’s important to get them right the first time. Using the current numbers described in [Table 3.2](#Table_3_2), the rules are as follows:

* For multiple species, enter codes consecutively (the order does not matter, though personally I’ve always kept it consistent)
  + Example: “67” = both Mb and Ha
* If you’re unsure about a particular species, enter “0” after its classification code. 0’s that follow a particular code apply only to that species.
  + Example 1: “20” = uncertain Cuvier’s
  + Example 2: “120” = certain Gervais’ and uncertain Cuvier’s
  + Example 3: “1020” = uncertain Gervais’ and uncertain Cuvier’s
* If you’re not sure about anything, just type “0”.
* “9” is only used to indicate false events. Do not include “9” with other codes.

## 3.3 Interactive Click Discrimination

During the validation/species ID process, only clicks that meet criteria for validation-level discrimination are displayed. While threshold values cannot be changed, it is possible to change the influence of one or more discriminators on-demand (where “discriminator” refers to a particular set of criteria).

As discussed in the “Detection Protocols” section, the options for discriminator influence are:

* Positive (clicks are displayed if they meet criteria)
* Negative (clicks are not displayed if they meet criteria)
* Disabled (the criteria will not influence whether or not clicks are displayed)

The presence of Positive or Negative discriminators will affect results as indicated in [Figure 3.5](#Figure_3_5).

|  |
| --- |
|  |
| **Figure 3.5**: Influence of “Positive” and “Negative” discriminator combinations on the set of clicks displayed for interactive event validation/species identification |

Whenever a new event is loaded, discrimination settings will be reset to the default scheme (which may be specified by the file *ValidationDefaults.xlsx* discussed in section [2.3.2 Using Multiple *ClickDiscrimParams* Files](#_2.3.2_Using_Multiple)).

## 3.4 Other features of the interactive Code

### 3.4.1 Limit on Number of Clicks

In some cases, the number of clicks in an event could be too overwhelming to examine efficiently. To avoid this, it is possible to set a limit on the number of clicks presented.

If click limitation is enforced, an algorithm will be run to decide which clicks to include. This will apply every time discriminator settings are changed, and of course whenever a new event is presented. The algorithm works as follows:

* Segments within the event are ranked by greatest proportion of clicks that meet discrimination criteria
* Clicks within the highest-ranking segments are added to the pool of clicks to be presented, until the maximum number of clicks allowed is reached
* If the last included segment has too many clicks, only the first couple of clicks (in order of occurrence within the segment) are retained, such that the total number of clicks in the pool is equal to the maximum number

One advantage of this algorithm is that ICI information is retained, since it is essentially sampling sequences of clicks rather than individual clicks from across the event.

### 3.4.2 Switching Events

Users may revisit completed events or jump forward to new ones by entering the “e” command. This command will bring up a table listing each event start time and the species ID code if there is one (see [Figure 3.6](#Figure_3_6)). If the event has been completed before, all data entered previously (i.e. species ID code and comments) will be retained unless the user overwrites them.

|  |
| --- |
| C:\Users\BeslinW\Documents\BeakedWhales\Documentation\Fig_EventSwitch.png |
| **Figure 3.6**: Event switcher window. Events yet to be validated are marked in red. Choose an event to jump to by clicking its checkbox in the *Active* column, then click *OK* to confirm (note that the table will automatically scroll back to the top every time a checkbox is clicked; this odd behavior is unfortunately a MATLAB limitation). Clicking the *Last* button will instantly change to the first event that has not been validated. |

### 3.4.3 Quitting and Resuming

If the application closes before all events have been completed, it will automatically pick up where it left off on the next run (provided it is run on the same dataset). When this happens, any options involved in the initial creation of the spreadsheet will be ignored.

### 3.4.4 Saving, Crashes and Backups

When the application is running, it keeps an internal version of the output table. However, this table is only saved to disk when the user specifies to do so with the “u” command, or when the application closes (with the “q” option or upon completing a dataset).

Note that the spreadsheet file cannot be saved if it is already open in Excel. However, if the application fails to save, it will always give you the option to try again, so there should be no worries about data loss. Furthermore, should the application run into an unexpected error, it will attempt to save a backup version of the table. This backup will have the same name as the original file, followed by a timestamp. Such crashes are not expected, but are always a possibility. The only case known to certainly trigger a crash is if you close any of the persistent figure windows (i.e. anything but pop-ups).

# 4. RUNNING THE CODE

The new BWD code was designed to separate the interface from the implementation. In other words, there is no longer a need to edit the core function files directly. All settings are applied either in spreadsheet files (as with detection criteria), or in dedicated .m files designated “master”. The “master” scripts are also used to run each process, which means they are the only code files you need to interact with. Furthermore, BWD attempts to automate as much of the I/O handling as possible, to minimize the number of input parameters that need to be entered. However, this feature means that it is very dependent on file naming conventions.

This section describes the contents of BWD’s root directory, input and output files, routine parameters, and ultimately how to run the code.

## 4.1 Code Directory Structure

Before running any part of BWD, it is good to make sure that the code directory contains all required files and folders, with the correct file names. The contents of the root directory are described in [Table 4.1](#Table_4_1).

|  |  |
| --- | --- |
| **Table 4.1:** List of files and folders in the BWD code directory. Folders are in italics. | |
| **File/Folder** | **Description** |
| *+BWD* | Folder containing all code files for the beaked whale detection routine. The plus sign is important as it is MATLAB syntax for “package” (see [Appendix B](#_APPENDIX_B:_THE)) |
| *DetectionCriteria* | Folder containing detection protocol definitions. The files within should follow the structure described in section [2.3 Detection Protocols](#_2.3_Detection_Protocols). |
| *ReferenceSpectra* | Folder containing MAT files with information on mean spectra of clicks from various beaked whale species. These are used when displaying spectral overlay plots during the interactive event validation/species ID process. |
| *Documentation* | Folder containing this and other reference documents |
| BWD\_master.m | Interface script for setting up and running automatic event detection |
| identifySpecies\_master.m | Interface script for setting up and running manual event validation |
| createPresenceTable.m | Function for converting species ID codes into binary presence/absence tables |
| subsetValidated.m | Script for creating subsets of “Validated” spreadsheets |
| SpeciesCodes.xlsx | Microsoft Excel spreadsheet listing species ID codes |
| Triton\_detParams\_8dB.spec.prm | File containing initial click detection parameters to be used in Triton |

Of all these files, the only ones that you need to edit regularly are *BWD\_master.m* and *identifySpecies\_master.m*, and also *createPresenceTable.m* and *subsetValidated.m* if you wish to use them. The code within *+BWD* should not normally be accessed, unless you plan to make changes to how BWD works. *DetectionCriteria* may be edited if you want to modify or create new beaked whale detection protocols.

## 4.2 Input and Output

There are several types of input and output files, most of which use specific naming conventions. BWD will store all of its output for a particular dataset within one analysis folder (to be specified by the user).

This section only discusses input and output files specific to a dataset, not global inputs such as detection criteria or reference spectra.

To explain naming conventions, variable filename parts will be colour-coded as follows:

* Deployment name (specified by user)
* Detection protocol target
* Timestamp
* Integer
* Other (explained in text)

Non-variable filename parts will remain in regular black font.

### 4.2.1 WAV Files

BWD uses WAV audio files to extract click parameters and save them in MAT files. They are required input for the automatic event detection part, unless all MAT files already exist (if MAT file creation was interrupted, BWD will pick up where it left off automatically).

WAV files can have any name, as long as the recording start time is encoded somewhere in the filename with the format:

*YYYYMMDD.hhmmss*

where “.” represents any single character. WAV files are also the only input files that may be stored anywhere (i.e. they do not need to be within the analysis folder).

### 4.2.2 Triton cTg Files

cTg files produced by Triton contain information on start/end times of clicks. Like WAV files, these are used to extract click parameters and save them in MAT files, and are required input for the automatic event detection part unless all MAT files already exist.

cTg files should have exactly the same filenames as their corresponding WAV files. They must also be stored in a folder specifically called *metadata* located within the analysis folder.

### 4.2.3 MAT Files

After click parameters have been extracted from WAV and cTg files, they are stored in MATLAB data files (MAT). Each recording will have at least one MAT file and possibly more, depending on how many clicks were detected. These MAT files are required for all subsequent parts of BWD, including event validation.

MAT files will have the following naming convention:

*DepName\_YYYYMMDD\_hhmmss\_i.wav*

where *i* is an integer used to distinguish multiple MAT files corresponding to the same recording.

To maintain compatibility with SBP’s version of the code, the contents of these MAT files has not changed. Each MAT file contains the variables listed in [Table 4.2](#Table_4_2).

|  |  |
| --- | --- |
| **Table 4.2:** List of variables in MAT data files for each recording | |
| **Variable** | **Description** |
| bw10db | 10 dB lower, upper, bandwidth |
| bw3db | 3 dB lower, upper, bandwidth |
| dur | click duration (ms) |
| F0 | center frequency (kHz) |
| Fs | sampling rate |
| ici | Inter-click-interval (ms) |
| nSamples | number of samples within -8 dB of peak (another measure of duration) |
| offset | # of samples before click in extracted timeseries |
| peakFr | peak frequency (kHz) |
| pos | position of click in wav file (in seconds from start of file) |
| ppSignal | peak-to-peak signal level |
| rawDur | duration of .wav file (s) |
| rawStart | start date/time of .wav file |
| rmsNoise | rms level of noise sample preceding click |
| rmsSignal | rms level of click |
| slope | slope of click (kHz/ms) |
| snr | signal to noise ratio |
| specClick | power spectrum of click |
| specNoise | power spectrum of noise sample |
| yFilt | timeseries of click |
| yNFilt | timeseries of noise sample |

### 4.2.4 Results Folder

Whenever automatic event detection is run, BWD will create a folder within the analysis folder to store output for that particular combination of data and detection criteria. This folder can have any name. However, If a folder with the same name already exists, then BWD will create a new one with a unique integer appended to its name. Existing results will never be overwritten.

### 4.2.5 DetectionProtocol.txt File

At the end of an event detection run, BWD will create a file called *DetectionProtocol.txt*. All this file contains is the name of the detection protocol used. Its only purpose is to inform the next step (i.e. event validation) of which protocol to use. Users do not need to interact with this file.

### 4.2.6 “RawEvents” Spreadsheet Files

Automatic event detection results will be saved in Microsoft Excel spreadsheets with the name:

*DepName\_Target\_RawEvents.xlsx*

where *Target* is the name of the corresponding target in the detection protocol. If a protocol has multiple targets, then each target will have its own *RawEvents* file. *RawEvents* files are used as input for the interactive part of BWD.

Each row in the spreadsheet corresponds to one segment which was deemed to be an event. The columns are explained in [Table 4.3](#Table_4_3).

|  |  |
| --- | --- |
| **Table 4.3:** Description of columns within “RawEvents” spreadsheets | |
| **Column** | **Description** |
| StartTime | Start time of the segment, in the format *YYYYMMDD\_hhmmss*. |
| EndTime | End time of the segment, in the format *YYYYMMDD\_hhmmss*. |
| NClicksAll | Total number of clicks detected by Triton within the segment |
| NClicksBeaked\_EventDet | Number of clicks that met the criteria for beaked whale clicks (according to the event-level click discrimination criteria) |

### 4.2.7 “Validated” Spreadsheet Files

When the interactive part of BWD prepares to analyze a new set of events (as marked within a *RawEvents* spreadsheet), it creates a second spreadsheet named:

*DepName\_Target\_Validated.xlsx*

where *DepName* and *Target* are identical to those of the corresponding *RawEvents* file.

Unlike *RawEvents* which only lists individual event segments, the rows of a *Validated* spreadsheet correspond to redefined event periods (e.g. calendar durations). The columns are explained in [Table 4.4](#Table_4_4).

|  |  |
| --- | --- |
| **Table 4.4:** Description of columns within “Validated” spreadsheets | |
| **Column** | **Description** |
| StartTime | Start time of the event, in the format *YYYYMMDD\_hhmmss*. |
| EndTime | End time of the event, in the format *YYYYMMDD\_hhmmss*. |
| TimeWithBeaked | Total duration of all segments within the event that had positive detections. This may be most useful when events are defined based on calendar durations. The format here is *hh\_mm\_ss* (because Excel makes things complicated otherwise). |
| NClicksAll | Total number of clicks detected by Triton, summed across all segments within the event that had positive detections. |
| NClicksBeaked\_EventDet | Total number of clicks that met the criteria for the event-detection round of click discrimination, summed across all segments within the event that had positive detections. |
| NClicksBeaked\_Validation | Total number of clicks that met the criteria for the validation round of click discrimination (default scheme), summed across all segments within the event that had positive detections. Unlike the number of clicks presented in the interactive application, there is no limit on this number. |
| Species | Numeric codes for species presence within each event |
| Comment | Comment about the event |

### 4.2.8 “Presence” Spreadsheet Files

Once completed, “Validated” spreadsheets may be converted into “Presence” spreadsheets. These have exactly the same file name as “Validated” files, except that the *Validated* part is replaced with *Presence*.

Like “Validated” files, “Presence” files have a row for each event, but columns consist mainly of presence/absence scores for each species ([Table 4.5](#Table_4_5)).

|  |  |
| --- | --- |
| **Table 4.5:** Description of columns within “Presence” spreadsheets | |
| **Column** | **Description** |
| StartTime | Start time of the event, in the format *YYYYMMDD\_hhmmss*. |
| EndTime | End time of the event, in the format *YYYYMMDD\_hhmmss*. |
| Presence\_*X* | Presence/absence score of a species; *X* corresponds to the species name as listed in the *SpeciesCodes.xlsx* file. There may be several columns of this type, one for each species whose code was entered somewhere in the “Validated” table. Values in these column may be one of three numbers:  1 = present  0 = absent/undetected  -1 = unknown  Note that if the “Uncertain” code was entered alone, all species will be marked as -1 for that event. Furthermore, if a code is not recognized, then all presence scores for that event will be blank. |

“Presence” spreadsheets are not created automatically. They must be created by running a separate function in the BWD code directory called *createPresenceTable.m*. This is discussed later in subsection [4.4.1 Converting Species ID Codes](#_4.4.1_Converting_Species).

### 4.2.9 JPEG Files and Plot Folders

During the interactive species ID process, several plots will be saved as JPEG files, either automatically or on request. All plots will be saved within a new folder called:

*plots\_Target*

Summary and spectral overlay plots ([Figure 3.1](#Figure_3_1) and [Figure 3.2](#Figure_3_2)) will be saved automatically upon loading each event. These files will conform to the following naming system:

*DepName\_YYYYMMDD\_hhmmss\_Target\_Type.jpg*

Where the timestamp corresponds to the event start time, and *Type* is either “*summary*” or “*overlay*”.

Plots of individual clicks and their parameters ([Figure 3.3](#Figure_3_3) and [Figure 3.4](#Figure_3_4)) may also be saved on user request. These files will be written to a subfolder within the plots folder called *clicks*. The naming style for individual click plots is:

*DepName\_YYYYMMDD\_hhmmss\_click\_i\_SortType.jpg*

Where *i* indicates the (sorted) click number, and *SortType* is either “*pp*” for peak-to-peak sorting, or “*pf*” for peak frequency sorting. For corresponding images of click parameter values and discrimination scores, these will have the same filename plus “*\_discrim*” appended at the end.

## 4.3 “master” Interface Scripts

The two main steps of the beaked whale detection code (automatic detection and manual validation) are run using dedicated interface scripts called “master”. These scripts define input parameters and are also used to launch each process. They are the only code files you need to interact with.

### 4.3.1 BWD\_master.m

This script controls the automated part of BWD, including compilation of clicks into MAT files and event detection. Set input parameters by editing the marked variables as indicated in [Table 4.6](#Table_4_6).

|  |  |
| --- | --- |
| **Table 4.6:** Editable parameters in *BWD\_master.m*. These control click compilation (extraction of features into MAT files) and automatic event detection. | |
| **Parameter** | **Description** |
| depName | Name of the deployment/dataset |
| dirPath\_analysis | Full path to your output directory |
| dirPath\_audio | Full path to your WAV file directory |
| detProtocol | Name of the detection protocol you want to use (one of the folders within *DetectionCriteria*) |
| dirName\_detResults | Name of the results folder |
| segDur | This parameter controls segmentation of audio files for continuous data. Must be specified as a MATLAB duration object, which uses the syntax *duration(h,m,s)*.   For example, to divide a recording into 1-minute segments, type *duration(0,1,0)*.  To use recording durations as segments, type *duration(Inf,Inf,Inf)*. |
| nfft | Number of points to use for FFT |
| nMATClicksMax | Maximum number of clicks that can be stored in an output MAT file. If a recording has more clicks than this, it will be divided into multiple MAT files. |

Note that parameter *dirPath\_audio* is required only for the click compilation step. If MAT files have already been produced, then this parameter may be left empty. Also note that if MAT file creation was interrupted, the routine will automatically pick up where it left off on next run by reading the names of existing MAT files.

### 4.3.2 identifySpecies\_master.m

This script controls the interactive event validation/species identification part of BWD. Set input parameters by editing the marked variables as indicated in [Table 4.7](#Table_4_7).

|  |  |
| --- | --- |
| **Table 4.7:** Editable parameters in *identifySpecies\_master.m*. These control event redefinition and the interactive event validation/species identification process. | |
| **Parameter** | **Description** |
| dirPath\_analysis | Full path to your output directory |
| dirName\_detResults | Name of the results folder |
| targetName | Name of target to analyze. There must be a *RawEvents* and/or *Validated* spreadsheet containing this target name. |
| dirPath\_analysis | Full path to your output directory.  Make sure this is consistent with what you entered in *BWD\_master*. |
| eventMergeOpt | A string that controls which definition to use for the start and end time of a detection event. Options are as follows:  ‘none’ – use raw segment durations as the basis for event durations.  ‘timegap’ – segments with positive detections that occur within a certain time period of one another will be merged into single large events  ‘calendar’ – use fixed calendar durations as the basis for event durations. A given calendar period will be considered a detection event if it contains at least one segment with beaked whale detections. |
| eventMergVal | Complementary parameter to *eventMergeOpt*. It represents different things depending on what you entered for *eventMergeOpt*:  If *eventMergeOpt* = ‘none’, then this parameter is not used.  If *eventMergeOpt* = ‘timegap’, then this parameter is a duration object representing the minimum separation that two segments with positive detections may have to be considered separate events.  If *eventMergeOpt* = ‘calendar’, then this parameter is a string specifying the calendar duration. Valid options are the same as for the *unit* argument of MATLAB’s *dateshift* function; this includes ‘hour’, ‘day’, ‘week’, ’month’, and ‘year’. |
| nClicksMax | Maximum number of clicks to include for event validation. Set to *Inf* to disable. |

As discussed in section [3.4.3 Quitting and Resuming](#_3.4.3_Quitting_and), if events have been partially validated for a given deployment and target, the routine will automatically pick up where it left off on next run by reading the “Validated” spreadsheet.

## 4.4 Additional Scripts

The BWD code directory contains additional scripts for processing data after event validation. This includes conversion of species ID codes to presence/absence scores, and isolation of particular events.

### 4.4.1 Converting Species ID Codes

The function *createPresenceTable.m* is used to read species ID codes within a “Validated” spreadsheet and create a corresponding “Presence” spreadsheet. Use this function as you would any of the “master” scripts: open it in the MATLAB editor, edit the marked variables, and run. In this case, there is only one variable to edit, corresponding to the full path to the “Validated” file you want to convert. The output “Presence” spreadsheet will be saved in the same directory as the “Validated” sheet.

### 4.4.2 Isolating and Revisiting Events

*subsetValidated.m* is a script for isolating events of interest and creating new “Validated” spreadsheets. The new sheets can then be used as input for the BWD validation code, allowing you to process these events without having to go through all of them. This is useful, for example, if you want to revisit all events that contained unknowns.

Subsetting is done based on species ID codes. At present, *subsetValidated.m* is somewhat limited in that it operates on individual code digits. This means, for example, that all occurrences of the “Uncertain” code are treated the same; it cannot distinguish between unknowns for different species. To use the script, edit the marked variables directly as indicated in [Table 4.8](#Table_4_8).

|  |  |
| --- | --- |
| **Table 4.8:** Editable parameters in the *subsetValidated.m* function. These control which spreadsheets to create a subset from and how to make the subset. | |
| **Variable** | **Description** |
| filePath | Path to “Validated” spreadsheet you want to subset |
| codeNum | Vector of species ID digits you want to include in the subset. For example, [0;8] (or [0,8]) will include all events whose Species ID numbers contain the digits 0 or 8. |

Output files produced by *subsetValidated.m* will not overwrite existing spreadsheets. Their filenames will be identical to the source “Validated” file, plus *\_Sub* appended at the end. Furthermore, if a subset file already exists, the routine will add a unique integer to the filename. Thus, you can create as many subsets as you want.

Note that *identifySpecies\_master.m* does not ask for the path to “Validated” spreadsheets directly. The validation code is set to automatically search for files ending with *\_Validated.xlsx*, and thus will not recognize subset files as they are. To use a subset, you will need to edit its file name such that it conforms to *DepName\_Target\_Validated.xlsx*, replacing the original file. It’s recommended that you keep a copy of the original by renaming it first.

# APPENDIX A: IMPLEMENTING NEW DETECTION PARAMETERS

In the original SBP/JS version of the code, click discrimination and event detection criteria were defined and evaluated in sequence directly within the main scripts. The new BWD takes a completely different approach: all manipulation of criteria (definition and evaluation) is handled separately from the main code sequence. This is accomplished via an object-oriented construct, whereby detection criteria are managed entirely within class definition files. In other words, the new BWD declares special types of variables that are responsible for storing and using data relating to detection criteria.

One advantage of this system is that it makes it much easier to modify criteria, and also to define multiple sets of criteria. For example: in the old code, deactivating certain parameters required commenting-out the relevant sections and then rerunning the entire script(s). With the new code, parameters can be activated or deactivated simply by changing the values of a variable. Also with the old code, trying out different sets of criteria (i.e. new thresholds and/or combinations of active parameters) required modifying the code and then rerunning the whole process for every set. With the new code, it’s possible to create an array of variables where each element defines a different set of criteria, and then simply loop through that array.

The disadvantage to an object-oriented system is that it may seem less intuitive than the procedural (i.e. traditional) approach. Furthermore, modifying this part of the code could be difficult without an adequate understanding of object-oriented programming in MATLAB. Since the list of parameters available to test is limited to those that have been implemented ([Table 2.1](#Table_2_1)), code modification will be required should you wish to add new parameters. Fortunately, this manual is here to help.

The following sections give a quick overview of object-oriented programming (OOP), explain how OOP is used in BWD, and give a walkthrough of how to implement the calculation/assessment of new detection parameters.

## Very Brief Introduction to Object-Oriented Programming in MATLAB

The central feature of object-oriented programming is that it allows you to create your own types, or “classes” of variables. Just as the standard built-in variables have classes (e.g. *double* for numbers, *char* for strings, *cell* for containers, etc.), you can create your own classes too. Custom classes are useful for representing literally any noun you can think of, both tangible and intangible; for example, in a separate application, I once created an array of *Bird* variables, which were manipulated by a *BirdManager*. Variables of a custom class are usually referred to as “objects” or “instances”.

Objects basically do two things: 1) they hold other variables (called **properties**), and 2) they execute internal functions (called **methods**). If you’ve used MATLAB’s “datetime” function, this is a good example of how objects work. When you execute something like:

*dt = datetime(Y,M,D,h,m,s)*

it creates a variable (*dt*) which is in fact an object of class *datetime*. This *datetime* object holds several variables (e.g. *format*, *year, month, day,* etc.), and you can use various functions like *exceltime* or *dateshift* to derive information from those variables. Those variables are properties, and the functions, which are actually implemented within the definition of *datetime*, are methods. Properties and methods also have **attributes**, which control things such as read/write access to the properties or methods for code that is external to the class. A list of attributes I’ve used for detection criteria classes is shown in [Table A1](#Table_A1).

|  |  |  |
| --- | --- | --- |
| **Table A1:** List of property and/or method attributes used in the definition of detection criteria classes. This is far from exhaustive relative to the complete list of attributes available in MATLAB. If they are not specified explicitly, attributes also have default values. For example, the default for “Access” attributes is *‘public’*, and for “Constant” it is *false*. | | |
| **Attribute** | **Applies to** | **Description** |
| SetAccess | properties | Controls which code can modify properties directly; may be set to ‘public’ (all code has permission) or ‘private’ (only code implemented within the class has permission). ‘protected’ is another option, but not used here. |
| Access | properties or methods | Controls which code can retrieve and modify properties, or execute methods |
| Constant | properties | Constant properties cannot be modified. Values of Constant properties are the same across all instances of a class, and they can also be accessed without an instance. |
| Static | methods | This is similar to Constant, but applies to methods. Static methods cannot depend on property values (except for Constant ones), but they can be accessed without an instance of the class. Essentially, they are like normal functions that happen to be defined within a class file. They are useful for carrying out general procedures that are relevant to the class. |

In MATLAB, classes are defined in dedicated m-files that start with a *classdef* statement. Unlike standard scripts or functions, these files cannot be “run”, because they do not consist of a single, sequential list of instructions. Instead, class files contain several, potentially unrelated blocks of code implemented in various places, which are accessed as needed by other code. The typical structure of a class definition and how it differs from traditional scripts/functions is illustrated by the pseudocode in [Figure A1](#Figure_A1).

|  |  |
| --- | --- |
| **Script/Function File** | **Class File** |
| function [outArgs] = myFuncName(inArgs)  % for function definitions only  *Sequence of code to execute*  end % for function definitions only | classdef myClassName < mySuperClass  % superclass is optional, but as a rule of thumb  % it’s almost always better to specify “handle”;  % the reason why is beyond the scope of this  % lesson  properties (*attributes*)  *List of properties*  % Often this is just a list of variable  % names; properties do not need to  % be given values here, unless they  % are Constant  end  methods (*attributes*)  function obj = myClassName(args)  % This is called a “constructor” – it’s  % a special method that creates an  % an instance of the class.  % It must be in a methods block  % with default attributes.  *Construction sequence*  *%* typically assigns property  % values, or calls other   % methods that do so  end  *other method definitions*  end  end |
| **Figure A1**: Comparison of the code structure between MATLAB script/function files and class definition files. In accordance with MATLAB syntax, everything preceded by a % sign is a comment. Words in italics are pseudocode. Note also that the class structure here only defines one properties and one methods block, but there could be more than one, with each having different attributes. | |

These are the basics of object-oriented programming in MATLAB as relevant here. Of course, this is only scratching the surface. For more information on the topic, refer to the MATLAB documentation.

## Object-Oriented Handling of Detection Criteria

In BWD, the two types of criteria (click discrimination and event detection) are each implemented as their own class, called *ClickDiscriminator* and *EventDetector*. Both classes follow the same structure and have many properties and methods in common1.

The structure for both *ClickDiscriminator* and *EventDetector* can be summarized in [Figure A2](#Figure_A2).

**Figure A2**: Shared structure of the *ClickDiscriminator* and *EventDetector* class definitions. Property or method names that are in italics here are placeholders, with ellipses representing incomplete names.

One thing to take away from [Figure A2](#Figure_A2) is that the assessment for each parameter is implemented separately as its own method. This means that adding a new parameter is basically just a matter of adding a new method.

Within the main BWD series of scripts, objects of class *ClickDiscriminator* or *EventDetector* are created by specifying the path to a spreadsheet file as an input argument to the constructor method. The scripts then call the *evalBeaked…* methods (using info on a specific dataset as input) to run click discrimination or event detection. Thus, it essentially becomes a two-line operation within the main code. The syntax looks something like:

*clickDiscrim = BWD.ClickDiscriminator(discrimSpreadsheetPath) % creates a ClickDiscriminator object*

*isBeaked = clickDiscrim.evalBeakedClicks(data) % runs click discrimination*

[Figure A3](#Figure_A3) summarizes the flow of code that happens when executing such statements.

**Figure A3**: Flow of code (or function call stack) when executing click discrimination or event detection in the new BWD. The large cells essentially represent code files, while the small ones are functions (thus, each dark blue cell is a method). The example here is for click discrimination, but the same logic applies to event detection.

## How to Add New Parameters

While the object-oriented system may appear complicated, another advantage to it is that parameter definitions are nicely compartmentalized, so the number of places where you need to edit code is minimal.

Since the two class definitions (*ClickDiscriminator* and *EventDetector*) have the same structure, the implementation of new parameters for each of them works very similarly too. This is summarized in the steps below:

1. Open the class definition file in the MATLAB editor
2. Go to the static methods block
3. Within this block, create a new function definition. This is where you will implement the calculation of your new parameter. The function must follow a few rules (use the other functions in the block as a reference):
   1. The function name starts with *assess\_* and ends with the name of your parameter
   2. There must be two input parameters, one for data and one for thresholds:
      1. *(data,th)* for ClickDiscriminator; *data* holds the contents of a MAT file, and *th* is a two-element vector corresponding to lower and upper threshold values
      2. *(clickPassed,th)* for EventDetector; *clickPassed* is a vector of true/false values (one for each click), and *th* is the threshold (one element only)
   3. There must be one or two output parameters depending on the type of criteria:
      1. *[pass,val]* for ClickDiscriminator; *pass* is a vector of true/false values (one for each click), and *val* is a vector of each click’s parameter value
      2. *[pass]* only for EventDetector; *pass* in this case is a single true/false value
4. Go to the constant properties block at the top of the file
5. Add the function you created to the property *critFuncs*, following exactly the same convention as for the other functions2:  
    *‘ParamName’, @BWD.ClassName.assess\_ParamName*

That’s it!

1Advanced note: it would be entirely possible, and perhaps more elegant from an OOP point of view, to implement these two classes as subclasses of some superclass that defines the common bits. However, I decided against doing this to keep things simple.

2You may be wondering why this step is necessary. The reason is that the assessment functions for each parameter will be called in a loop when the full click discrimination/event detection function is run (i.e., the function denoted *evalBeaked…*). To do this however, there needs to be a way of knowing which function should be run at each iteration. The best way to do this is by using function handles, i.e. variables that point to a function. The property *critFuncs* is a structure that acts as a library of function handles for all parameter assessment functions (the @ symbol is MATLAB syntax for defining a function handle). The *evalBeaked…* function(s) then references that library to call the appropriate function for all parameters to be used (i.e. those assigned a category > 0).

# APPENDIX B: THE PACKAGE SYSTEM

The BWD code is organized using MATLAB packages. Packages are basically special folders that contain a collection of related code files. Aside from organization, packages have a practical advantage in that they help with disambiguation of redundant code files. In other words, if there are multiple M files on your MATLAB path with the same name, packages can ensure that you are using the right versions. The original motivation for using packages was to differentiate SBP/JS’s code from WB’s.

Packages follow a few simple rules that are important to know should you need to understand how BWD works:

* All package folder names are marked by a + sign preceding the package name
  + Example: *+BWD*
* Package directories do not need to be on the MATLAB path themselves (in fact they cannot), but their parent folder must be on the path
* In MATLAB, code within a package must be referenced by typing the package name followed by a dot and then the function/class name.
  + Example: *BWD.detectBeaked*
* Packages may be nested within one another. To access code in nested packages, reference the full package stack with dot notation.
  + Example: *BWD.Utilities.getFileNames*

All routine processes of BWD are coded within the *+BWD* package. However, the files within this package should be stable and are not intended to be accessed directly on a regular basis. To use BWD, two “master” scripts located outside *+BWD* serve as an interface to the package. Use these files to set input parameters and run the code within *+BWD*.

# APPENDIX C: COMMENTS ON THE CODE AND IDEAS FOR THE FUTURE

WB’s implementation of the BWD code works significantly differently from the original SBP/JS version, but the basic idea (i.e. detecting events through threshold-based discrimination of clicks) has not changed. This is a good thing for the sake of consistency with previous research. However, as a consequence of its origin, there do remain a few minor elements in the new BWD code that could be improved.

One thing to note is the contents of MAT data files. In part to maintain backwards compatibility, the MAT files produced by the new BWD are identical to those produced in the original code (first batch, i.e. those with data on all Triton-detected clicks). That is, they contain the variables described in [Table 4.2](#Table_4_2), in the same format as before. This is somewhat awkward and inefficient especially from a disk usage point of view, because not all variables are actually used. Furthermore, the new code relies on some parameters that were not present in the old one (like ZCR) which may be recalculated multiple times. Those variables would benefit from being added to the MAT files (assuming the MAT file system is kept at all in the future). However, as noted, implementing these changes would break backwards compatibility: files produced by the SBP/JS version (and older WB versions) may not work with the new code, requiring that data be recompiled from scratch.

A few other minor but useful potential changes are:

* Generalizing the system for loading reference spectra in the species ID process. This is likely the most inelegant section of code in the new BWD, as it relies on hardcoded filenames and sampling rates. Thus it cannot accommodate changes in the reference spectra as it is.
* Allow saving of non-default summary and spectral overlay plots in the species ID process. This would require a new, well-thought-out naming system for plot image files.
* Automatic resumption of click discrimination during event detection in the case of interruption. While it’s possible to recover from an interruption during click compilation (MAT file creation) and event validation, this is not the case for event detection

# APPENDIX D: CAVEAT ON SAMPLING RATE

The original SBP/JS version of BWD was limited to a particular sampling rate (*Fs*), and different versions of the code existed to accommodate different rates. With the new BWD, sampling rate dependence should no longer be an issue, as *Fs*-sensitive operations are conducted on the basis of time rather than number of samples.

However, our beaked whale detection process as a whole still retains a certain dependence on *Fs*.

One area where *Fs* can present a barrier is with digital filtering. BWD uses a Butterworth bandpass filter, who’s frequency response necessarily differs with *Fs*. At present, filter cutoff frequencies must be hardcoded for each *Fs*, so only a limited number of sampling rates are supported (at time of writing, this includes 128, 192, 200, 250, 300, and 375 kHz). Fortunately, adding support for new sampling rates is fairly straightforward as it essentially just involves adding a new number. To do this, the code file “*extractFilterClick.m*” within *+BWD* will need to be edited.

Though BWD can easily accommodate new sampling rates, *Fs* can still be a problem outside BWD as well. As it turns out, *Fs* can influence the performance of click detection in Triton. This happens mostly because Triton uses spectrograms for its initial click detection stage (Batch Short-Time Spectrum), and detection accuracy actually depends heavily on the spectrogram parameters – especially the number of samples used in FFT. Unfortunately, our version of Triton has no option for setting number of FFT samples without editing the code itself. Number of FFT samples is essentially fixed to the next power of two greater than the number of samples equal to 10 ms (this is one of several quirks I’ve noticed with Triton’s click detection tool, indicating that it is still under development). When used with HARPS with a sampling rate of 200 kHz, this resulted in an FFT window of 2048 points, which worked fine; however, for 250 kHz AMAR, the FFT window size was 4096, which resulted in many missed detections.

As a quick workaround, the Triton code we use has been edited to round to the **nearest** power of 2 rather than the next highest, which brings FFT window size back to 2048 points for 250 kHz sampling rate (implemented in file *Detector/dtST\_batch.m*). However, this is not a very robust solution, so be mindful of this if needing to use other sampling rates.

# APPENDIX E: GLOSSARY OF TERMS AND ACCRONYMS

* **BWD** – Beaked Whale Detector, a tentative name for Team Whale’s beaked whale detection code
* **SBP/JS** – Refers to Simone-Baumann-Pickering and Joy Stanistreet’s version of the BWD code
* **Segment** – A subset of the entire time series. For each segment, the conditions for a detection event are assessed independently. Segments may consist of entire WAV files, or divisions of WAV files as defined by the user (e.g. 1-minute intervals).
* **Event** – A period of time within which clicks of the target species were identified, according to detection criteria. The fundamental unit for an event period is the segment. Segments with positive detections may be merged together to form larger events.
* **Detection Criteria** – A set of rules applied to clicks to ultimately determine if a particular target is present. There are two types of detection criteria, called *Event Detection Criteria* and *Click Discrimination Criteria*.
  + **Event Detection Criteria** – Determine whether or not an event occurred based on the number of candidate target clicks
  + **Click Discrimination Criteria** – Evaluate the parameters of individual clicks and decide if the clicks may correspond to the target or not
* **Detection Parameters** – Sometimes used interchangeably with “criteria”, but generally, a “parameter” refers to a particular metric (e.g. peak frequency), whereas a “criterion” encompasses both the metric and the threshold values applied to it. Parameters may also be called “features”.
* **Event-Level Assessment** – The application of detection criteria (both click discrimination and event detection) used for automatic event detection
* **Validation-Level Assessment** – The application of detection criteria (specifically click discrimination criteria) used for manual event validation and species identification
* **Protocol –** A collection of *detection criteria* for one or more *targets*
* **Target –** A species or group of species of interest. For example, a *target* might encompass all beaked whales (a.k.a. “Beaked”), just bottlenose whales (a.k.a. “Ha”), or all beaked whales except bottlenose whales (a.k.a. “NonHa”).
* **Discriminator –** A set of click discrimination criteria for identifying the clicks of one or more species within a *target*. If desired, it is possible to define *protocols* that apply more than one discriminator for a particular *target*. The effect this has depends on the assessment level (event or validation) and, in the case of validation-level, user options. Within the code itself, a “discriminator” refers to an instance of the *ClickDiscriminator* class.