User Manual BatInspector

SoftwareVersion: **0.2**
Date: **23-05-30**

BatInspector is a software to analyze and manage recordings of bat calls.
It supports Elekon BatCorder projects as well as plain WAV files.

Main Features:

- manage bat recordings
- Analyze bat calls:
  -frequency max
  - frequency min
  - duration
  - many more
- Waterfall diagrams:
  - zoom in and out
  - zoom to detected call
- play recordings in original and 10x stretched speed
- Automatic species detection (experimental)
- Manual species detection
- Powerful data filtering
  - user defined filter expressions
- Generate reports

Index

1 Document History

Date	Author	Changes
22-03-15	CMU	Initial version

2 Introduction

3 Installation and setup

3.1 Installation

- Install python 3.10
  https://www.python.org/downloads/
  It has to be installed in the location C:\Program Files\python310

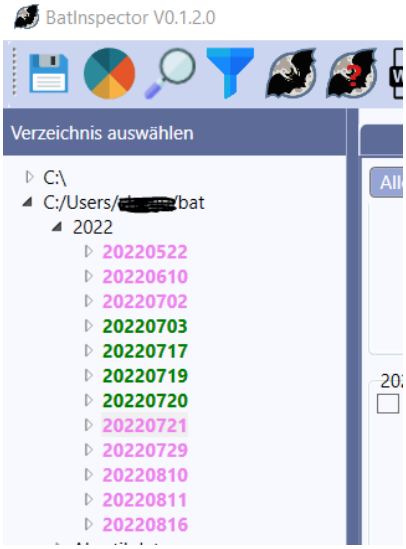
- Install tensorflow 2
  https://www.tensorflow.org/install
- Install BatInspector:
  call setup.exe and follow the instructions

4 Basic Workflow

4.1 Copy Data from recorder to PC

In order to work with the recorded data, they must first be copied to the PC. Copy the data to a folder in your hard disk. Each project or recording session should have its own directory.

4.2 Open a Project

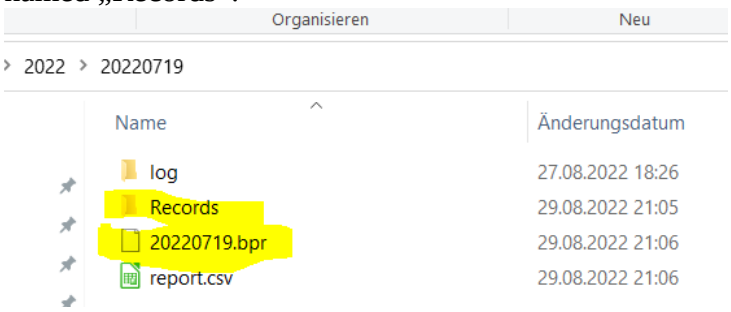


Select a directory in the file browser on the left side of the main window and click the directory containing the project to open.

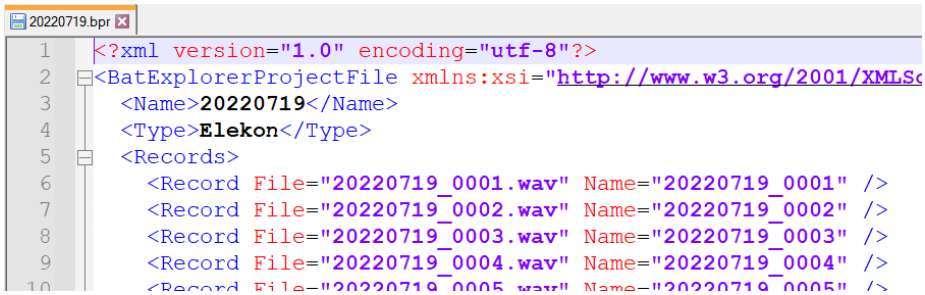
Directories containing projects with project files are marked in cyan or green color.

4.2.1 Format of the project file

BatInspector is able to read project files in the Elekon data format.
The project file must have the extension *.bpr and the recordings must be located in a subfolder named „Records“:



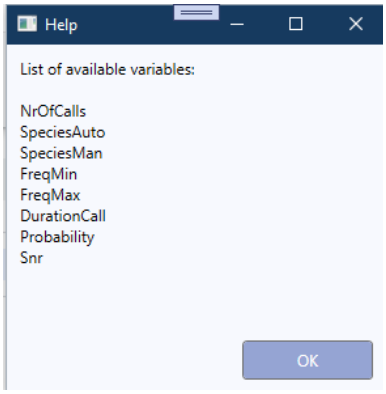
The project file has an XML format, containing a list with all the files and some additional information:



4.2.2 Format of the Report File

4.3 Analyze calls automatically

4.4 Working with Filters



4.5 Analyze calls manually

5 Menues and Screens

6 Scripting Language

6.1 Syntax

The syntax of each script line is:

```
<command> <Par1> <Par2> … <Parn>
```

Each line contains a command with parameters.
A parameter may be

- a fixed value: `22`
- a string enclosed in „“: `„this is a sample string“`
- an expression: `=(cos(a) + 5)`

An expression is surrounded by the start character „=(„ and the end character „)“:

```
SET a =(cos(b) * 2)
```

A detailed description of the expression syntax is provided in chapter 7

Before and between command and parameters abitrary white space (space or tab) is allowed.

A line starting with the character ‚#‘ is considered as a comment

```
# this is a comment and will not be executed
```

6.2 Commands

6.2.1 SET

Function: Assigns a value to a variable. If the variable does not exist, it will be created
Syntax: SET <VarName> <Value>
Parameter: VarName: Name of the variable
Value: Value to assign the variable
Example: SET a 33
SET b =(a + 5)

6.2.2 CALL

Function: calls a script. All variables of the calling script are accessible in the called script.
Syntax: CALL <FileName>
Parameter: FileName: Name of the script file (full path or only name. If only the name is provided, the file must b located in the directory <ScriptDir> from the application settings.

Example: CALL C:\scripts\script1.scr
CALL script2.scr

6.2.3 FOR … END

Function: For Loop

Syntax: FOR <iteratorName> <start> <end>
  # a block of code
:
END

Parameter: iteratorName: name of the iterating variable
start: start value for the iterator
end: end value for the iterator

Example:

```
FOR i 1 10
        LOG i
END
```

6.2.4 WHILE … END

Function: While Loop

Syntax: WHILE <expression>
  # a block of code
:
END

Parameter: expression: an expression with boolen result. As log the expressio is true, the loop will be executed

Example:

```
SET i 0
WHILE =(i <  10)
        LOG i
        SET i =(i + 1)
END
```

6.2.5 BREAK

Function: breaks the execution of a loop (applicable for FOR and WHILE loops
Syntax: BREAK
Parameter: none
Example:

```
WHILE =(a < 100)
  LOG a
  IF =((b >0) && ( a > 50)
    BREAK
  END
  SET a =(a + 1)
END
```

6.2.6 LOG

Function: write a message to the log window
Syntax: LOG <message> <type>
Parameter: message: the message to log (fixed val. Or expression)
type: optional parameter for message type:
- INFO (default)
- ERROR
- WARNING
- DEBUG

Example:

LOG „this should never happen“ ERROR

7 Expression Parser Syntax

With the integrated formaula parser it is possible to express complex logic formulas or mathematical calculations.
This is useful to create filter expressions and scripts.

7.1 Operators

Operator	**Description**
+	Addition
-	Subtraction
*	Multiplication
/	Division
&&	Logical AND
||	Logical OR
!	Logical NOT
<	Lesser than
<=	Lesser or equal than
>	Greater than
>=	Greater or equal than
==	Equal
!=	Not equal
()	Braces for math terms or function arguments
[]	Square braces for array index
„“	Double quotes to enclose strings

7.2 Data Types of Expressions

Data Type	Range	Examples
RT_BOOL	True, False	(a <= 3.0)
`RT_INT64`	- 2^{63} … 2^{63}	-236, 224
`RT_UINT64`	0 … 2^{64}	23456
RT_HEX	0x0 …. 0xFFFFFFFFFFFFFFFF	0xFA10
RT_FLOAT	64 bit floating point number	-3.5, 66.8767, -3.23E-12
RT_COMPLEX	Complex numbers	1+2.0i, -4.5 – 3i, 2.3 + 1i
RT_STR	String	„this is a string“, „1.0“

7.3 Supported Functions

7.3.1 Trigonometrical Functions

sin(x)
Function returns sinus of a number",
Params x: a number
Returns sinus of x
asin(x)
Function returns arcus sinus of a number
Params x: a number
Returns arcus sinus of x

sinh(x)
Function returns sinus hyperbolicus of a number",
Params x: a number
Returns sinus hyperbolicus of x

cos(x)
Function returns cosinus of a number",
Params x: a number
Returns cosinus of x

acos(x)
Function returns arcus cosinus of a number",
Params x: a number
Returns arcus cosinus of x

cosh(x)
Function returns cosinus hyperbolicus of a number",
Params x: a number

Returns cosinus hyperbolicus of x

tan(x)
Function returns tangens of a number",
Params x: a number
Returns tangens of x

atan(x)
Function returns arcus tangens of a number",
Params x: a number
Returns arcus tangens of x

7.3.2 Mathematical Functions

sqrt(x)
Function returns square root of a number
Params x: a number
Returns square root of x

pow(x, e)
Function calculate power of a number",
Params x: a number
e: exponent
Returns x^e (x to the power of e)

exp(x)
Function calculate power of e",
Params x: a number
Returns e^x (e to the power of x)

ln(x)
Function calculate the natural logarithm of a number
Params x: a number
Returns natural logarithm of x

abs(x)
Function calculate the absolute value of a number (real or complex)
Params x: a number
Returns absolute value of x

arg(x)
Function calculate the argument of a complex number
Params x: a complex number
Returns arg x

7.3.3 String functions

strlen(s)
Function calculate the length of a string
Params s: a string
Returns length of string s

strListCnt(str)
Function counts the number of items in a ‚;‘ separated list of strings
Params str: a string
Returns number of items in list

strListItem(str, item)
Function returns the selected item in a ‚;‘ separated list of strings
Params str: a string
item: index of selected item (0..n)
Returns selected item of list

substr(str, start, length)
Function calculate the substring of a string
Params str: a string
start: start character (0... <length of str – 1>)
length: length od substr (0... <length of str – 1 - start>)
Returns the sub string

toupper(str)
Function convert string to upper case
Params str: a string
Returns the string in upper case

7.3.4 Miscellanous Functions

cast(exp, type)

Function	cast an expression to a specific type
Params	exp: an expression type: a type („RT_BOOL“, „RT_FLOAT“, „RT_INT64“, „RT_UIN64“, „RT_STR“, „RT_HEX“ „RT_COMPLEX“)
Returns	length of string s

if(exp, argTrue, argFalse)

Function	evaluate boolean expression and return corresponding argument
Params	exp: a boolean expression argTrue: an expression for exp == true argFalse: an expression for exp == false
Returns	argTrue or argFalse depending on result of exp

vars()

Function	return a list of variables and their values
Params	none
Returns	list of all defined variables and their values

consts()

Function	return a list of constants and their values
Params	none
Returns	list of all defined constants and their values

7.3.5 Functions to Handle CSV Files

`openCsv(fileName,withHeader,separator)`

Function:	opens a csv file and reads it to memory	
Parameter:	fileName:	Name of the script file (full path or only name. If only the name is provided, the file must b located in the directory <ScriptDir> from the application settings.name of the file (full path)
	withHeader:	(optional, default 0) 1: first row contains header with column names. The colums can be accessed via the names if opened in this mode.
	Separator	(optional, default=";") separation character in lines for the cells
Returns:	**>=0:** a handle to access the file, <0: error opening the file	

`closeCsv(handle, write)`

Function:	closes a csv file and releases the memory"	
Parameter:	handle:	file handle to access csv file
	write:	(optional, default:0) 1: write file to disk
Returns	nothing	

`getCell(handle, row, col)`

Function:	get cell content of a previously opened file	
Parameter:	handle:	file handle to access csv file
	row	row number (1..n)
	col	col name (if opened with option ‚withHeader') or col number(1..n)
Returns	content of cell as string	

`setCell(handle, row, col, value)`

Function:	set a cell content of a previously opened csv file	
Parameter:	handle:	file handle to access csv file
	row	row number (1..n)
	col	col name (if opened with option ‚withHeader') or col number(1..n)
	value	value to write to cell (string)

`getRowCount(handle)`

Function:	get row count of a previously opened csv file (including header)	
Parameter:	handle:	file handle to access csv file
Returns:	number of rows in opened file	

7.3.6 Functions to deal with projects

`createPrjFromFiles(name, srcDir, destDir, maxFiles, maxFileLen, latitude, longitude, landscape, weather)`

Function: create a project from a list of WAV files. If the number of WAV files exceeds the specified maximum number of files the projects gets split into multiple projects. The name of the subprojects are supplemented with an index nr: <prjName_<nnn>. Files that exceed the specified maximum file length are split into several file names. The resulting file names are supplemented with index numbers: <fileName>_<nnn>.wav

Parameter: name: name of the project (without extension)
srcDir: source folder containing WAV files
destDir: destination folder containing project folders
maxFiles: maximum number of files per project
maxFileLen: maximum file length per WAV file [s] (float)
latitude: geographical position
longitude: geographical position
landscape: short description of the landscape of the recording location
weather: short description of weather conditions during recording

Returns: error code: 0=OK

`inspectPrj()`

Function: identify calls, parameters of calls and the species in the recordings of the currently opened project. The results of the inspection process are stored in a report file named „report.csv“. The result file is stored in a sub folder with the name of the selected modell and is located in the project folder.

Parameter: none

Returns: error code: 0=OK

`openPrj(directory, name)`

Function: open a project.

Parameter: directory: directory that contains the project
name: project name (*.bpr)

Returns: projectList: a list of ‚;‘ separated project names

7.3.7 Functions to deal with project informations

`calcProbabilityRatios()`
Function: calculate the ratio for the probabilities of classification for the 1st and 2nd rank for each call in the currently opened project.
Parameter: none
Returns: nothing

`getCallCount(index)`

Prerequisite: a record file must be generated first
Function: get number of calls for a specified file in currently opened project
Parameter: index: index of file (0..n)
Returns: number of calls

`getCallInfo(fIndex, cIndex, infoType)`

Prerequisite: a record file must be generated first
Function: get call information for a specified call in a specified file in currently opened project
Parameter: fIndex: index of file (0 … n)
cIndex: index of call (0 … m)
infoType: type of information
„SPEC_AUTO“: automatically detected species, classifier 1
„SPEC_MAN“: manually detected species
„PROB_RATIO“: ratio betw. 1St and 2nd rank of classification
„PROBABILITY“: probability for classified species
Returns: the requested information

`getPrjFileCount()`
Function: get nr of files in open project
Parameter: none
Returns: number of files in currently opened project

`getFileName(index)`

Function: get full path file name of file in project opened project
Parameter: index: index of file (0… n)
Returns: full path of wav file

`getFileInfo(index, infoType)`

Prerequisite: a record file must be generated first
Function: get file specific information in currently opened project
Parameter: index: index of file (0… n)
infoType: type of information to set
„NAME“: name of the file
„DURATION: duration of the recording [s]
„SAMPLE_RATE: sampling rate [Hz]
Returns: the specified data

`setFileInfo(index, infoType, data)`

Prerequisite: a record file must be generated first
Function: set file specific information in currently opened project
Parameter: index: index of file (0… n)
infoType: type of information to set
„SELECT“: set the state of the checkbox „Select“ („TRUE“ or „FALSE“)
data: the data
Returns: error code

`setCallInfo(fIndex, cIndex, infoType, data)`

Prerequisite: a record file must be generated first
Function: set call information for a specified call in a specified file in currently opened project
Parameter: fIndex: index of file (0 … n)
cIndex: index of call (0 … m)
infoType: type of information
„SPEC_MAN“: manually detected species
data: the data
Returns: the requested information

`getNrOfSpecies(fIndex)`

Prerequisite: a record file must be generated first
Function: get number of auto detected species for spec. file in open project
Parameter: fIndex: index of file (0 … n)
Returns: nr of automatically detected species

`getRankSpecies(fIndex, rank)`

Prerequisite: a record file must be generated first
Function: get the species name for the specified rank in specified file in open project
Parameter: fIndex: index of file (0 … n)
rank: rank (1 … n, 1= most calls)
Returns: species name for specified rank

`getRankCount(fIndex, rank)`

Prerequisite: a record file must be generated first
Function: get the nr of calls of species for the specified rank in specified file in open project
Parameter: fIndex: index of file (0 … n)
rank: rank (1 … n, 1= most calls)
Returns: nr species in recording for specified rank

7.3.8 Functions to deal with WAV files

`bandPass(file, fMin, fMax)`

Function: sets the sampling rate of a file
Parameter: file: name of the file (full path) or alternatively file index of file in currently open project
fMin: low frequency limit for band pass [Hz]
fMax: high frequency limit for band pass Hz]
Returns: error code

`getSampleRate(fileName)`

Function: returns the sampling rate of a file
Parameter: fileName: name of the file (full path)
Returns: the sampling rate

`rescaleSampleRate(fileName, factor)`

Function: multiplies the sampling rate of a file with a factor
Parameter: fileName: name of the file (full path)
factor: factor to multiply sampling rate
Returns: the new sampling rate

`SetSampleRate(fileName, sampleRate)`

Function: sets the sampling rate of a file
Parameter: fileName: name of the file (full path)
sampleRate: new value for sampling rate
Returns: nothing