Inhalt

[1 Functionality: 2](#_Toc138689464)

[1.1 Universal file format treatment 4](#_Toc138689465)

[1.2 Tab Wav Header: 5](#_Toc138689466)

[1.3 Tab Resampler: 5](#_Toc138689467)

[1.4 Tab Scanner: 9](#_Toc138689468)

[1.5 Tab Annotate: 9](#_Toc138689469)

[1.6 Tab YAML Header 10](#_Toc138689470)

[1.7 Tab Player 11](#_Toc138689471)

[2 QT Tricks gelernt: 11](#_Toc138689472)

[3 Bugliste: 11](#_Toc138689473)

[4 Requirements: 17](#_Toc138689474)

[5 Programmstruktur 20](#_Toc138689475)

[6 Methods 20](#_Toc138689476)

[6.1 Klassendiagramm automatisch erstellen: 20](#_Toc138689477)

[6.2 Player: 20](#_Toc138689478)

[6.3 ann\_spectrum(self, dummy, data): 21](#_Toc138689479)

[6.4 %Annotation anhand der MWList-Tabelle alt: 22](#_Toc138689480)

[7 GIT 23](#_Toc138689481)

[8 conda 23](#_Toc138689482)

[9 Threading know how: 23](#_Toc138689483)

[10 WAV-Header Details zu wBlockalign und wFormatTag 25](#_Toc138689484)

[11 virtual environment: 32](#_Toc138689485)

[12 Dokumentation 32](#_Toc138689486)

[12.1 Generelles 32](#_Toc138689487)

[12.2 Aufgabe/Tab ‚Scanner‘: Aufnahme visuell analysieren 33](#_Toc138689488)

[12.3 Aufgabe/Tab ‚Annotate‘: Sendersuche und Annotation 34](#_Toc138689489)

[12.3.1 Schritt 1: ‚Scan‘: Automatische Peak-Identifikation 35](#_Toc138689490)

[12.3.2 Schritt 2: ‚Annotate‘: Zuweisung der korrekten Stationen 36](#_Toc138689491)

[13 Doku Methoden und Variablen 38](#_Toc138689492)

[13.1 38](#_Toc138689493)

[Generelle Strukturdiagramme: UMLET, Programm umlet-exe direkt aus Directory 38](#_Toc138689494)

[13.2 Class statlst\_gen\_worker(QtCore.QThread): 39](#_Toc138689495)

[13.3 Class class autoscan\_worker(QtCore.QThread): 39](#_Toc138689496)

[13.4 Class class StemlabControl(QObject): 40](#_Toc138689497)

[13.5 Class WizardGUI(QMainWindow) 41](#_Toc138689498)

[14 Generelle Strukturregeln: 46](#_Toc138689499)

# Wichtige Details:

**Verzeichnisse aller auf Platten gespeicherten Aufnahmen in Verzeichnisdateien im COHIRADIA-Folder gespeichert:**

dir > C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\COHIRADIA\verzeichnis\_MERKUR.txt

Interessante Info: Alternative AM-Sendermod-Software auf GNURadio:

<https://radio-bastler.de/forum/showthread.php?tid=13599>

## Files ins Archiv einspielen und aktivieren:

**Preprocessing:**

GGf resampeln auf STEMLAB-SR;

bei Fileserien dzt noch mit MATLAB SDR\_resampler\_full\_v3; nach Programmstart immer

cd C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\

Ggf auch Gain-Korrektur

Nextfile-Pfade mit dem COHIWizard ausbessern:

* rel Pfade
* letzte Datei hat einen leeren Eintrag)

**Files über WINSCP in ein neues Dir PF\_JJJJX unter /data/ hochladen**

9^h3%lRnGaCp

PF: prefix WB, HS, GM, SR, oder andere (Initialen Autor)

LLLL: Jahr (2023)

X: Postfix, Buchstaben in alphabet Reihenfolge A,B,C,….

**RM:** [**https://github.com/radiomuseum**](https://github.com/radiomuseum)

**Dort ins Dir** <https://github.com/radiomuseum/cohiradia-metadata/tree/main/yaml> gehen für Einspielen der yaml-Files:

Filename immer PF\_JJJJX-ersteswavFilename\_ohne\_Extension.yaml

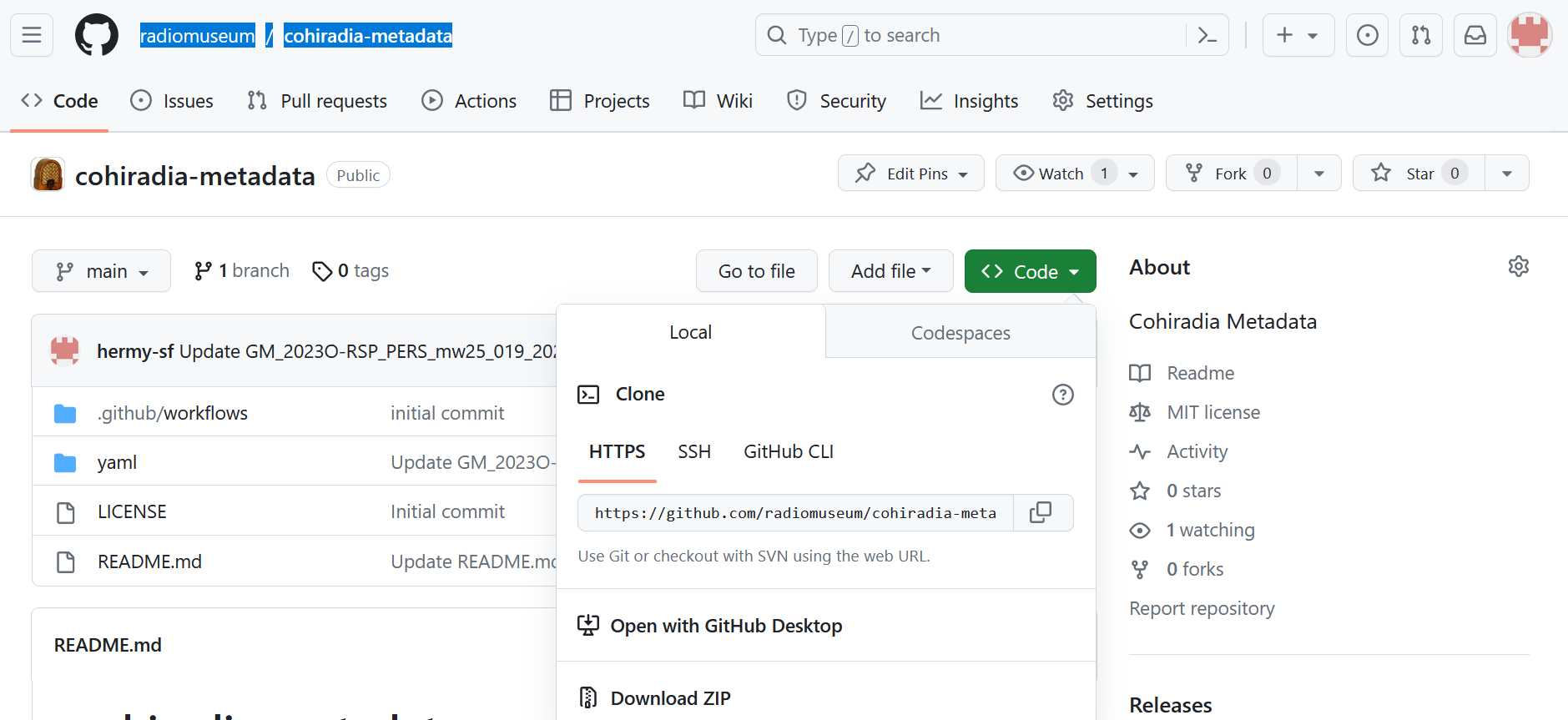
**Danach auf GITBash:**

curl -XPOST <https://cohiradia.radiomuseum.org/api/metadata/import>

von Zeit zu Zeit:

git clone ###Addr###

###Adr### von radiomuseum/cohiradia-metadata über ‚Code



Lokale **Yaml-Dateien** findet man im COHIRADIA-Verzeichnis unter git\_cohiradia-metadata\_03\_12\_2023

## Nützliche Info:

**Zeitberechnungen in MATLAB:**

a= datetime(2023,10,24,18,58,00) - datetime(2023,10,24,20,00,37)

seconds(a)

**Rekompilation Details:**

Im GUI###.py File die Namen der zu ladenden IconBilder als relative Pfade zu haben; muss man immer prüfen.

Dann pyinstaller --icon=RFCorder\_Icon.ico –F RFCorder\_v2\_exWizardv21.py

Dann pyinstaller --icon=COHIWizard\_ico4.ico –F SDR\_COHIWizard\_v21.py

**INFO:**

++ Gibt es ein Filterdesigntool in Python wie in MATLAB ?: Ja: https://pypi.org/project/pyfda/; Start von der Commandline mit 'pyfdax'

# Dokumentation Webpage:

Hallo Hermann  
  
ich habe noch etwas weiter entwickelt und der "neue Weg" kann bald produktiv gehen. Die Idee ist, dass die einzelnen Seiten als Markdown File (ein sehr einfaches Beschreibungsformat, das die wesentlichen Formatierungsregeln kennt und in 5 Minuten zu lernen ist) in einem Git-Repo erfasst werden können.  
  
Git Repo: <https://github.com/radiomuseum/cohiradia-website>

* Im ./src Ordner sind Dateien im Format name\_sprache.md  
  Die Seite ist dann über <https://www.radiomuseum.org/cohiradia/name.html> erreichbar

Markdown Dokumentation (ist wirklich einfach): <https://commonmark.org/help/>  
  
**Konkretes Beispiel:**  
- Der Einführungstext ist in ./src/software\_de.md und ist über <https://www.radiomuseum.org/cohiradia/software.html> erreichbar.   
  
Die neuen Seiten sind noch nicht "aktiv", aber hier verfügbar:  
<https://www.radiomuseum.org/cohiradia/software.html>  
  
**Was fehlt:**   
- aktuell nur deutsch, ich werde später die englische Version erstellen.   
- Die Detailseite einer Aufnahme fehlt noch (kommt später heute oder morgen)  
  
Somit ist die Möglichkeit geschaffen, für Cohiradia beliebige Seiten zu erstellen.  
- Neues Markdown File erstellen   
- Zu Oberst (siehe bestehende Files) muss ein sogenannter Front Matter sein. Das ist ein mit drei Strichen abgegrenzter YAML Teil. Pflicht ist da die Variable title.   
- Anschliessend kann beliebiger Markdown verwendet werden   
- Nach GitHub push  
- <https://www.radiomuseum.org/git.cfm> öffnen (zum aktualisieren; füge ich dann noch als Github Action hinzu)   
- Die Änderung oder die neue Seite ist verfügbar.   
  
  
ich habe noch etwas weiter entwickelt und zumindest für Demo-Zwecke funktioniert es. Die Idee ist, dass die einzelnen Seiten als Markdown File (ein sehr einfaches Beschreibungsformat, das die wesentlichen Formatierungsregeln kennt) in einem Git-Repo erfasst werden können.   
  
Git Repo: <https://github.com/radiomuseum/cohiradia-website>   
  
- Im ./src Ordner sind Dateien im Format name\_sprache.md   
- Die Seite ist dann über <https://www.radiomuseum.org/cohiradia/name.html> erreichbar   
  
Konkretes Beispiel:   
- Der Einführungstext ist in ./src/software\_de.md und ist über <https://www.radiomuseum.org/cohiradia/software.html> erreichbar.

**Dokumentation SDR\_COHIWizard**

# TODO laufend:

Einbau Endlos-Blockabspielung (Playlist in File) im Gauert-RFCorder.

LO-Bias: Dieser Offset wird ins linedit-Feld als Text eingegeben. Wenn dieser Text nicht als Integer interpretierbar ist (+/- ein integer < oder > als zulässige Grenzen), dann muss irgendwann eine Fehlermeldung kommen.

Annotator: Umstellen auf csv statt xls Tabelle

Annotator: Eigene Eingabe ermöglichen und in Tabelle als neuen Eintrag speichern

Im yamlgenerator: bereits einmal eingegebene Parameter der Aufnahme-Station in Konfigurationsfile speicherbar machen

Resampler: Startcut/Stopcut implementieren

Testroutine für Validity der Kordinateneingaben beim YAML-Editor, bei Country: selectable list; bei Band: ? Selectable list ?

Lo-Level-Modification

* Modif: Stemlabcontrol muss auf eine Variable LO\_offset zugreifen (am besten in systemstate[„LO\_offset“], die zu win.ifreq hinzugezählt wird, bevor dieser Wert an STEMLAB übertragen wird

Bereits implementiert: System\_state[„LO\_offset“] in Hz

GUI-Level-Modification:

* Mache lineEdit Feld immer editierbar,

wenn Togglebutton aktiviert, um dies tatsächlich als einen Sonderzustand auszuweisen (Funktion activate\_LO\_bias:

* Highlight linedit
* Überprüfung auf Validität und Übernahme;
* Schreibe in systemstate

Bei ‚Play‘ aus Grundzustand (nicht nach Pause):

* Wenn Togglebutton active: Warning that offset will be applied, Yes no ?
* Inactivate Togglebuttons, denn während Play kann das Stemlabcontrol nicht stattfinden.
* Grenzen überprüfen, da jetzt erst LO bekannt ist: Betrag wenn negativ muss > LO sein , ansonsten muss LO + offset < 60000 (kHz) sein

Bei Stop und Initialization:

* Activate Togglebutton

Testen:

Abspielen von 32 und 24 bit Dateien ?

Teste alle Resampling Funktionen, die bisher gehen.

Scheduler weiterentwickeln für Resampling

# Functionality:

## Tab Wav Header:

**File open:**

If wav:

* open wavheader and trigger activation of tool
* Save Template header is active
* Overwrite Header active or connected callback gets flag ‘wav\_file = True’

if dat:

* inactivate all tools except wav editor
* make ‘dat > wav’ visible and active
* make button ‘Insert Header’ visible and active
* Overwrite Header inactive or connected callback gets flag ‘wav\_file = False’

Method: insert\_header:

* write current wavheader after consistency check to fisrt 216 bytes of opened dat file;
* then rename file according to SDRUno

**Purpose:**

* Read wav header of a selected recording record#.wav in RECPATH and display in WAVTABLE
* Make several fields editable

Opt: write template wav header: write current (edited) values in WAVTABLE to RECPATH\wizard\ record#\templatewavheader.wav

* Check for compatibility of all WAVTABLE entries wit data formats to be written to header; CRITIAL, errors cause program crash
* CHECK IF THIS FUNCTION IS STILL REQUIRED. DAT-Header are meanwhile automatically generated for dat files.

Opt: overwrite record wav header: overwrite first 216 bytes in to RECPATH\record#.wav with current (edited) values in WAVTABLE

* Check for compatibility of all WAVTABLE entries wit data formats to be written to header; CRITIAL, errors cause program crash
* CHECK IF THIS FUNCTION IS STILL REQUIRED. DAT-Header are meanwhile automatically generated for dat files.

**Elements:**

* Tables for all important wavheader fields,
* radiobutton for ‘edit’ 🡪 toggles ‘active’ ‘inactive’

Method read

If exist RECPATH\wizard\ record#\snrannotation.snr

Set button Scan green: Improve logistics

## Tab Resampler:

Ganze Resampling Funktionalität grafisch:

Parameterliste: Target Samplingrate, Target LO oder Bandunterkante, Target, Gain, Cuttingstart, Cuttingstop

Fileliste

Fenster mit Spektrum für die Positionierung des Resampling-Fensters

(Amplitudenkorrektur-Darstellung ?? nötig ? sox normalisiert ja)

Bei File open:

Start cut = Starttime: check ob time oder datetime

Stop Cut = Stoptime

Target LO = aktuelles LO

Target SR = nächsthöhere als im aktuellen File

Gain: 1

Spektrum plotten nicht nur in View Spectra sondern auch in diesem Canvas (Canvas neu definieren); eigenes plot spectrum in diesem Canvas

STRUKTURIERUNG:

Zentrale Liste von Tabs, die bei Erweiterungen einfach erweitert wird, evt ein Dictionary **TabRef**, dessen Einträge Referenzen auf allgemeine Objekte der einzelnen Tabs sind, z.B. ein erzeugter canvas, axes, GUI\_elemente, Grid-Koordinaten. TabRef wird beim init erzeugt: jeder Tab erhält eine tab\_init\_Funktion, die von \_\_init\_\_ aufgerufen wird. In dieser tab\_init wird TabRef erweitert und individuelle Einträge generiert. Damit muss man bei Tab-Änderungen oder Neuerstellungen nur die Liste der tag\_init-Funktionen und diese inits selbst verändern.

Funktionen im Ablauf des Programms greifen auf GUI\_elemente immer nur mehr über setters und getters oder globals zu und rufen immer nur eine GUI\_update Funktion auf, die für alle in TabRef eingetragenen Tabs die individuellen Refresh(TabRef) aufrufen.

Zentrale (self.) Variablen für die Kommunikation beliebiger Tabs miteinander

Jeder Tab hat GUI-Elemente, die bei Gelegenheit (nach File open oder manchmal auch bei Änderungen in danderen Tabs) updatet werden müssen: Jeder Tab hat eine refresh-Funktion, die von einer GUI\_update\_funktion aufgerufen werden können. Die Referenzen auf diese refresher werden in der zentralen Tab-Liste eingetragen, um Tabs an einer Stelle zu konfigurieren.

eigene Funktion init\_Tabref, die auf Anfrage Canvas und ax objekte erzeugt

Grunsätzliche Funktionen definieren, die jeder neue Tab hat:

Plot(**TabRef**[Name], canvas#, datax, datay, xlabe, ylabel): z.B.

Übergib self.Tabref["Resample"] als tabref

tabref["ax"].plot(datax\_,datay, '-')

tabref["ax"].set\_xlabel(xlabel)

tabref ["ax"].set\_ylabel(ylabel)

Refresh(**TabRef[Name], dict elements**): Updated alle GUI Elemente die in **TabRef[Name]** als upzudaten eingetragen sind. Diese elemente können auch über Tabs hinweg referenziert werden. Könnte bisherige update-Funktionen ersetzen, wenn generalisiert definiert; muss die values der GUI-Elementwerte aus aktuellem Tab einlesen

z.B. TabRef[„elements“][„lineEditposition”].setvalue(something)

TabRef.[„NAME“][„canvas“]: liste von canvas elementen

###TabRef.canvaspositions: liste dazugehöriger grid Koordinaten

TabRef.ax: Liste der zu den canvas gehörenden axes

Noch keine Listen erst Einzelelemente

TabRef.GUI\_read: Liste von GUI-Elementen, deren Werte in entsprechende Globals zu lesen sind sowie referenzen auf die entsprechenden Globals: Dictionary ?

TabRef.GUI\_write: Liste von Elementen, deren Werte aus den globals upzudaten sind

TabRef.

Aktuelle updater functions:

scanupdateGUI

Progressbarupdate

status\_writetableread ??????????

update\_resample\_GUI

reset\_GUI

inactivate\_tabs

activate\_tabs

scan\_activate

scan\_deactivate

check: Tab-übergreifende functions aktuell:

minSNRupdate

minPeakDistanceupdate

minPeakwidthupdate

??

minSNRupdate\_ScannerTab

set\_baselineoffset

Resampler State Machine:

Methode resample

Abonniert: sigR

Erzeugt soxstring

??? noch nötig nach scheduler ? targetfn 🡪 Sourcefn; target\_fh 🡪 source\_fh

Konfiguiert und Startet soxworker

Soxworker: slots: soxstring

SigPupdate

SigRFinished

SigSch

Deabonniert sigR

Startet sox

Startet progress-loop

Schreibt progress auf system\_flags[„progress“] (oder: auf slots[„progress“])

Methode LOshift

Abonniert: sigLOSH

??? noch nötig nach scheduler ? targetfn 🡪 Sourcefn; target\_fh 🡪 source\_fh

Erzeugt soxstring

Konfiguiert und Startet soxworker

Schreibt target\_fn auf sys\_status[„remfile“]

Shiftworker: slots: soxstring

SigPupdate

SigLOFinished

SigSch

Deabonniert sigLOSH

Macht die shifting-loop

Schreibt progress auf system\_flags[„progress“] (oder: auf slots[„progress“])

Methode accomplish

Abonniert: sigA

## nicht mehr nötig, wenn scheduler: Sourcefn 🡪 targetfn; source\_fh 🡪 target\_fh

Reset progressbar : könnte auch im scheduler passieren

Korrigiert den wavheader durch Überschreiben mit target\_fh

Rename das File aus system\_state[„newname“]

Remove file aus system\_state[„remfile“]

Emittiert Signal für scheduler weiterschalten

Methode scheduler:

Slot cnt

Abonniert SigSch

Befehlsketten\_liste[cnt]: sf; tf; sh; Referenz auf Signal f nächsten Task: SigR, SigAcc, SigLOSH, SigPU…;

th 🡪 sh Umschreibung; Labeltext f Progress updater, Blinkstate

in Schritt cnt:

disconnect SigSch

sf 🡪 system\_state[sf] kann beim ersten Mal ausserhalb geschrieben werden (in callback)

sheader 🡪 system\_state[sheader] kann beim ersten Mal ausserhalb geschrieben werden (in callback)

system\_state[tf] wird von nächstemTask selbst erzeugt,

system\_state[sf] = system\_state[tf]: check wenn wavfile, dann in nächstem Schritt höchstens header modifizieren, sonst neu schreiben

system\_state[labeltext] = schedule[cnt][„actionlabel“]

system\_state[blinkstate] = schedule[cnt][„blinkstate“]

? gui\_Label36 set to actionlabel ? hier oder gleich im progressbarupdate

System\_state[„newname“] = schedule[cnt][„newname“] ?check if necessary, File wird umbenannt

if not end of list: signal für start des nächsten Tasks

schedule[cnt][“startsignal”].connect(schedule[cnt][“method”])

schedule[cnt][“startsignal”].emit()

eventuell Waitstate

schedule[cnt][“startsignal”].disconnect(schedule[cnt][“method”])

cnt ++

Methode progress\_updater

Abonniert SigPU

Holt aktiven thread ab, der steht im schedule[cnt][updater\_thread]

Testet, ob alive

Holt progress von sys\_state.getprogress()

Aktualisiert progressbar

Aktualisiert progresslabel

NEXT STEPS:

* LOShifter mit Workerthread implementieren, damit Progressupdate funktioniert: 2h
* Schedule definieren
* Schedule mit scheduler testweise abfahren

## Tab Scanner:

**Purpose:**

Step through different timepoints oft he record, show spectrum,

**Elements:**

* Plotcanvas,
* Toolbar for figure
* Scrollbar
* Time Edit menu for entering time string, on click advance to the entered time

## Tab Annotate:

**Purpose:**

* identify peaks and SNRs
* Scan RECPATH\record#.wav for peaks above a certain threshold SNR in dB
* Write annotation file

**Click Button ‘Scan’**

Call autoscan():

Method autoscan():

If not exist RECPATH\wizard\ record#\snrannotation.snr

Scan recording at self.NUMSNAPS points (from spinBoxNumScan) and identify peaks via method autoscan()

If not exist RECPATH\wizard\ record#\ 🡪 generate RECPATH\wizard\ record#\

Writes yaml file self.annotation\_filename: snrannotation.yaml in einen Ordner RECPATH\wizard\ record#\

Call annotate()

**click** **Button ‘Annotate’**

Call annotate()

Method annotate()

* If not exist RECPATH\wizard\ record#\stationslist.yaml
  + Open MWList database in ROOTPATH\
  + Set status to read MWLIST: hängt beim refreshen
  + Step through MWList by Peak-frequency and generate raw yaml annotation file RECPATH\wizard\ record#\stationslist.yaml: here all stations have an appended index inside each frequency block
  + Write info of completed scan to RECPATH\wizard\ record#\status.yaml, Label ‘Scanned’: True, ‘LASTFREQ’: index = 0
  + Update status and statusbar hängt beim refreshen
* Else: call annotate\_interactive\_part()

Method annotate\_interactive\_part():

* If RECPATH\wizard\ record#\ status.yaml, Label ‘Scanned’: True and ‘annotated’ = False🡪
  + Read RECPATH\wizard\ record#\stations\_list.yaml
    - Set frequency counter to ‘LASTFREQ’
  + Warning and question if scan should be repeated from scratch
    - for each frequency list individual station/programme entries as items in clickable list
    - on selection write info to an info window and ask for confirmation or rejection
    - write selected item to annotation yaml RECPATH\wizard\ record#\ cohiradia\_metadata.yaml
    - Update status and statusbar hängt beim refreshen
* ??? else warning that already done, ask if new scan, if yes 🡪 Scan
* after last frequency set ‘annotated’ True

Button ‘Discard’:

* + advance: current frequency index +1

Status bar: show progress

* If to RECPATH\wizard\ record#\config.yaml, Label ‘Scanned’: True
  + show progress of raw file generation acc freq index/number of freqs hängt beim refreshen
* else:
  + show progress of annot = frequency index over total num freqs hängt beim refreshen

Status label: report on:

* If to RECPATH\wizard\ record#\config.yaml, Label ‘Scanned’: False
  + Show ‘Read MWtable’ during reading process
  + show ‘status raw scan’
* else:
  + show ‘ status annotate’

clickable list:

TODO: Wenn in Stationstabelle kein Eintrag 🡪 schreib Dummyeintrag in die Felder und ermögliche den Eintrag eigener Daten, wenn vorhanden

## Tab YAML Header

Dzt ist in der fun annotate() ein Aufruf von write\_yaml\_header() eingebaut, FRAGE: ist das dort optimal ? NEIN !

Geplant: Aktivierung bei Button Press WRITE YAML: Schreiben des Grundheaders wie bereits jetzt in

write\_yaml\_header(). Weiterer Ausbau: Lesen der Tabelleninhalte und schreiben in die fehlenden yaml-Felder.

FEHLT: UTC-Offset automatisch erzeugen

Erzeugen eines Tailer-Files

Zuletzt Zusammenkopieren von yamll-Header, Stationsannotationsfile und yaml-Tailer zum endgültigen yaml.

Feature: Aktiviere den TAB, wenn unter ‚ANNOTATE‘ die Annotation fertiggestellt ist. Aber dann kann niemand bereits zu Beginn einen, wenn auch unvollständigen, Header erzeugen.

Evt automatisches Durchlaufen der entsprechenden Tabs beim Aufnahmeverarbeitungsverfahren anhand automatischer Tabsprünge ?

## Tab Player

Threads für STEMLAB Control und Playloop übertragen und mit Play-Taste verbinden (Play-Callback)

Scrollbar für Positionswechsel, dann fragt man sich, ob die FF und FW Tasten noch gebraucht werden

SLOTs in die Workerfcns einbauen

Recloop

Timer/Scheduler

Zeitbalken manchmal falsch (check wie das Update erfolgt, am besten wäre nach Filesize etc)

## Universal file format treatment

Wurde weitgehend umgesetzt, ich glaube, da ist kaum mehr was zu tun.



TODO Self.DATABLOCKSIZE anpassen !

TODO Auch readsegment anpassen mit wFormattag und Bittiefe !

# QT Tricks gelernt:

Threads mit Slots \_\_slots\_\_ für die Variablen-Kommunikation mit dem Main thread. Siehe die implementierten Threads

Verhindern, dass eine Spinbox beim Eingeben eines Werte mit der Tastatur bereits feuert: keyboardTracking Property auf False setzen !

Locken eines Threadworker-Abschnittes mit mutex, siehe entspr. Workerfunktion

Probleme mit Sonderzeichen bei Filemanipulationen:

file = open(filename, encoding="utf8")

# Schlecht programmiert:

Filenamenlabels auf verschiedenen Tabs sollten sich besser bei einem Zentraltool registrieren und auf Event updaten als von Programmteilen aus gesetzt zu warden. Passiert in showfilename.

        self.ui.label\_Filename1.setText(self.my\_filename + self.ext)

        self.ui.label\_Filename2.setText(self.my\_filename + self.ext)

        self.ui.label\_Filename3.setText(self.my\_filename + self.ext)

        self.ui.label\_Filename4.setText(self.my\_filename + self.ext)

        self.ui.label\_Filename\_resample.setText(self.my\_filename + self.ext)

# Bugliste:

?? Center frequency offset verhält sich komplett wirr. Größere Analyse !

Wenn der Player noch nicht gestoppt ist, und man ein Shutdown auslöst, stürzt das Programm beim Remotehost-Close ab. Lösung: Entweder bei Shutdown ein SDR-STOP erzwingen oder den Shutdown nicht freigeben, bevor Player gestoppt wurde. Eher ersteres.

Abfangen von Zugriffsverletzungen, wenn ein File bereits durch andere App geöffnet ist: Frage ChatGPT

Resample lässt sich nicht mehr einstellen, nachdem man ein annotateScan 🡪 wav header editing und danach ein yaml-Header writing gemacht hat. Muss man neues File laden

Beim Stoppen von Play/Ende einer Playlist wird der wav-header editor nicht zurückgesetzt. Edit ist zwar inaktiv, aber das ist dennoch verwirrend. Irrelevant für den Player.

Centerfrequency offset sollte während des Abspielens entweder nicht zugänglich sein, oder unmittelbar akiv werden wenn was eingetragen wird. Das Offenbleiben passiert allerdings nur beim ersten Mal nach dem Laden eines neuen Files.

Maßnahme: Im Initatlisierungsteil von ui wurde die Zeile

self.reset\_LO\_bias()

auf L 786 hinzugefügt

Abfangen: Wenn ein Filepfad länge als 260 Zeichen wird, und das Programm versucht, dort ein File zu schreiben, crasht es. Abfangen: Pathlengths überprüfen oder Schreibmöglichkeit prüfen.

Instanz: write\_yaml\_header

In

        with open(self.cohiradia\_yamlheader\_filename, 'w', encoding='utf-8') as f:

Nach Wavheader-schreiben in dat-File wird die Playlist im Player-GUI nicht automatisch updated. Das File wird zwar im Filenamesfeld angezeigt

* + - Allgemeiner Playlist-Updater nach diversen Aktionen

TESTEN

Überprüft 29-11-2023, der Editor lässt das aktuell eh nicht zu; Wenn Fehler wieder auftritt, muss er woanders liegen: Beim wav-editor ist es erlaubt, 60 als Minutenwert einzutragen. Muss abgefangen werden

Bei Frage ob F-Offset exekutiert werden soll (bei nonzero offset eintrag) solle bei Drücken von ‚no‘ sofort der Offset deaktiviert werden

Nach der Abfrage wurde ebenfalls die Zeile

self.reset\_LO\_bias()

auf L 1462 hinzugefügt

**KONSEQUENZ: Jeder bereits eingetragene Offset wird gelöscht. Überlegen, ob das im laufenden Betrieb so praktisch ist.**

Wenn man den Scrollbar bewegt, ohne ein File geladen zu haben, oder wenn man ein nicht abspielbares File hat (falsche SR), dann stürzt alles ab. 🡪 Rollbalken für solche fälle deaktivieren.

Nach Start Player mit falscher Samplingrate: Filenamen bleibt, solle aber geschlossen werden. Daher open file bei ‚Resample‘ gefordert

Bei Playversuch und Fehler ‚STEMLAB not connected, cannot connect to Host: Play Button bleibt auf Pause Mode –Reset Playbuttongroup !

            self.reset\_playerbuttongroup()

            self.SigGUIReset.emit()

kann aber nicht von innerhalb des Fehlerauslösers gerufen werden 🡪 Signal an GUI aus der Instanz stemlabcontrol der Klasse StemlabControl. Die Connection wird in der cb-Funktion play\_manager hergestellt (in init ist stemlabcontrol noch nicht instanziert).

        stemlabcontrol.SigError.connect(self.reset\_playerbuttongroup)

        stemlabcontrol.SigError.connect(self.reset\_GUI)

Nach Bisamberg resample gibt es ein Image bei tiefen Frequenzen (um 550 kHz)

Walter: ich habe RFCorder\_v2beta aufgespielt. Zunächst habe ich in der

Anleitung nicht weit genug gelesen und die sox-Dateien vergessen.

Aber leider macht das Programm nicht weiter, weil StemLab

nicht gefunden wird. Die IP-Adresse habe ich aus der 1.Version

übernommen und korrigiert. Dort klappt auch alles.

Das Laden von Dateien klappt wohl auch, zumindest werden

plausible Grafiken angezeigt. Zum Resampeln kommt die

Aufforderung die Datei zu öffnen. Entweder bin ich blind

oder es fehlt eine Möglichkeit sie zu öffnen. Schliesslich

wird ja der richtige Dateiname bereits angezeigt.

Stan:

I did try the new RFCorder 2.0beta and here are my early comments:

1. The app took a little longer to load than the original, or so it seemed to me.
2. I didn't see any Playlength (Running Time) box on this beta release. One assumes that will be added in due course.
3. The Fast Forward and Fast Reverse buttons operated, but only momentarily. By that I mean that holding down the button did not allow for the advance or reverse to continue until the button was released. Instead, I had to keep letting up and the pressing down again repeatedly. It should only take one click and hold down to advance or reverse continuously if that makes sense.

I don't have sox installed on this laptop, though I tried doing so using the Terminal mode, but without success. Afterwards, I tried using your suggestion in the Resample mode and, yes, I   observed the dialog box for downloading and installing sox from the webpage, but the path location to success is not obvious. This is my problem, of course, not yours, the reason being that I am not a software oriented person. I'm just an old person now as well as a hardware person, perhaps just like many RMorg members, so unusual applications such as sox, and especially their path locations during installation, make them difficult to implement. This could be a drawback for those who want to Resample.

On the positive side, raw/dat and wav files are easy to select and load successfully. Consequently, if you eventually make the dashboard similar to RFCorder 1.0, then I think you have a good improvement on your hands with the added advantage of being able to play wav files. The Resample option can then come later.

To conclude, I notice that you stated that Vers. 2.0beta 'is a very simple tool which can only playback and resample', so I think my comments on the Fast Forward and Fast Reverse functions may not be valid at this stage. The same for the Playlength dialog box. If so, then you have accomplished your main goal of incorporating the choice for playing wav files.

By the way, except for one small paragraph in German, the user manual reads well and makes good sense.

Kind regards,

Stan

Fixed 29-11-2023: Beim wav-header editor wird nach Einfügen eines wav-headers in ein dat file eine Umbenennung eingeleitet und das neue File auch offenbar geladen. Allerdings wird nun bei erneutem Editieren des wav-Headers bei overwrite header eine weitere Umbenennung gemacht. Offenbar kriegt

overwrite\_header(self)

nicht mit, dass der Filename geändert wurde, usix scheint

Fixed 29-11-2023: Wenn man ein wav-File öffnet, das keinen wav-header hat, stürzt alles ab. Wahrscheinlich werden irrsinnige Werte aus dem Headebereich eingelesen und auf die wavheaderstruktur geschrieben oder Formatfehler: Abfrage ob es sich um ein RIFF-File handelt !

Nach STOP ist File nicht mehr offen, Resampler verlangt neues File Öffnen, obwohl alle Namen noch da sind und das Spektrum in der Anzeige auch.

Abhilfe: STOP löst ein Signal signal.reset GUI aus. Bei diesem ist aktuell reset\_GUI() abonniert (mit connect), in dem zusätzlich Filenamen und alle Canvas Inhalte gelöscht werden. Fileopened wird weiterhin auf False gesetzt

Verbesserungen:

Jeder Tab sollte eine individuelle Tab-Reset-Funktion haben, die jeweils beim reset\_GUI Signal abonniert ist. Dann kann man beliebig Tabs ändern und muss nur die entsprechenden TAB\_Resetter ändern, nicht ein zentrales Element. Daher muss das aktuelle reset\_GUI() dementsprechend zerlegt werden. Reser\_playerbuttongroup könnte bereits ein solcher resetter für den Tab ‚Player‘ sein, muss nur entsprechend erweitert werden

Allgemein: Abonnementprinzip durchziehen

Wenn man nach Play und dann Frage nach File Abbrechen drückt, geht die Playergruppe auf ‚Pause‘ (spielt allerdings nichts ab)

Geplante Abhilfe: FileOpen() gibt False zurück, wenn bei Wahl eines Files ‚Abbrechen‘ gedrückt wurde, also kein gültiger Filenamen vorliegt. Abfrage darauf und bei False Cb\_Butt\_toggleplay bricht mit Return False ab.

Funktioniert, aber nach dem Abbrechen ist der Play-Button ‚gedrückt‘ und reagiert erst wieder bei erneutem Drücken mit einem Filemenü

Abhilfe: reset playerbuttongroup() in derselben Abfrage wie oben

Wenn man nach Fileopen durch drücken von PLAY im Player im Filemenü ‚Abbrechen‘ wählt, kriegt man die Fehlermeldung ‚Negative invalid frequency offset..‘. Danach zwar normale Funktion, aber irriierend.

Geplante Abhilfe: siehe voriger Bug

TESTEN !

autorepeat soll Vorrang vor Nextfile haben

Nach Resample: Player spielt möglw nicht korrekte Spieldauer ab:

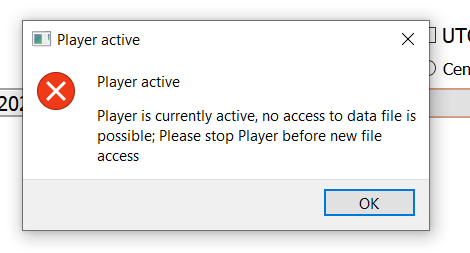
Nach Bismberg-Resample nicht bemerkt

Wenn man aus der Annotation unmittelbar in den resampler wechselt, ohne die Annotation abzuschließen, funktioniert die Anzeige des Schneidefensters zunächst (beim ersten Mal) nicht. Zwar läuft das Resampeln korrekt und das Ergebnis stimmt, danach wird auch das richtige Fenster geladen. Danach Wechsel in Annot ohne weiteres möglich und auch Rückkehr zum resampler. Nach Öffnen eines neuen Files geht auch alles.

Evt. Lösung: Man sollte den Resampler inaktivieren, wenn man annotiert und die Annotation inaktivieren, wenn man resampelt. Evt einen ‚Exit‘ Button bei Annotation einbauen.

Nach Player stop ist geladenes File offenbar nicht mehr aktiv, Scanner reagiert nicht mehr. Neues Fileopen aktiviert wieder. Nach Filestop automatisch ein Scan mit aktuellem File starten ?

Restart nach Pause wirft Warning,



wenn in endlosloop einmal wiederholt

Slider bei Player hat inkorrekte Gesamtlänge assoziiert, wenn Zeiteinträge im Wavheader inkorrekt. Man könnte sich aber da an Filesize, avgbytespSample etc orientieren: TESTEN

Wenn im Wavheader ein falscher Filesize drinsteht, und zwar zu groß, dann funktionieren die updates bei readsegment etc nicht mehr, auch der Annotator bleibt stecken. Lösung: Konsistenzcheck auf Filesize und Einträgen im wavheader bei write header und dann auch wieder beil Lesen des headers: Filesize und Eintrag im wavheader müssen zusammenpassen. Check am besten entweder im wavheadertool oder bei fileOpen sowie in entweder write\_wavheader oder overwrite header/generate header

Der Statusbar in Annotate wird nicht immer updated, man sollte auch threading während der Erstellung der Stationsliste (nach Laden der MWlist) verwenden.

Erstellen des neuen Filenamens bei wav-Header-Generierung (dat files): Check, ob die SDRUno-Namensteile das Datum aus dem Wavheader richtig enthalten

Beim Erstellen des wavheaders für dat-Files werden händisch falsch eingetragene Samplingraten nicht abgefangen (Popup-Menü).

24 bit Abspielen: Volumen ist viel zu klein. Bei plotSpectrum ist die Signalstärke normal, beim Abspielen um ca 1e-12 zu klein.

24-bit resampling: LÖSUNG: sox -L -e signed-integer -t wav -r 125000 -b 24 -c 2 -v 0.05 Bisamberg\_closure\_1476\_027.wav -e signed-integer -t raw -r 125000 -b 16 -c 2 Bisamberg\_closure\_1476\_027\_resampled.dat

Upsampling auf 32 Bit aktuell nicht vorgesehen. Könnte manchmal gewünscht sein.

Resample: Wenn LO-shift mit SR > Original-SR und Shift angefordert wird: Error dass noch nicht implementiert wird ausgegeben, aber dann wird resampling trotzdem gestartet. Müsste aber abgewürgt werden. Passiert nicht, da ein Finished-Signal abgesetzt und daduch der Scheduler weitergetrieben wird.

Inaktiviere ‚edit‘ im Wavheadertab nach Öffnen eines neuen Files

Dat file header geschrieben. Danach Header editiert und ‚overwrite header‘ 🡪 Absturz, da noch altes Filename aktuell, aber inzwischen wurde dieser geändert.

Centerfreq offset: Wenn play im Offset Mode, und Pause oder Play, kann Wert im gelb markierten Fenster umgestellt werden, aber nichts reagiert. Nach Stop wird der neue Wert nicht ins Alert-fenster „Wollen Sie wirklich Offset…“ übernommen.

AUSWEG ? : Sperre Eingabemöglichkeit während Play

Toggle Play reagiert nach manchen Abspielprozesse nicht auf Neuabpielen (unter Mode >), erst auf den 2. Klick. Dann oft Abspielen eines geladenen Files, anstelle Neustart.

Z.B. wenn ein Offsetwert ungültig ist, und ein diesbezügl. Error ausgegeben wird. Dann muss Play 2x gedrückt werden

Oder: nach do you really want to play with that offset ?= 🡪 NO

Absturz wenn Gain im Testmodus verändert wird

Bei Eingabe sinnloser Zeichen in LO-Fenster beim resampler stürzt alles ab

Header wird bei Verwendung von tmp\_header nach automatischem Schreiben des wav-headers nach resamplingirgendwie verkackt, es steht Unsinn drin. -🡪 evt diese Funktion überhaupt deaktivieren, hab ich nie mehr gebraucht.

Resample: Bei Versuch, Bisamberg-closure-Aufnahme (Header rcvr) mit LO-shiftzu resampeln, nach über (899 940 568 Bytes) Absturz: File "C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\COHIRADIA\Softwareentwicklung\SDR\_Postprocessor\SDR\_COHIWizard\_v12.py", line 1732, in readsegment

dataraw = unpack('<%ul' % 1 ,d + (b'\x00' if d[2] < 128 else b'\xff'))

IndexError: index out of range

Mögl Ursache: verkutzt sich bei Lesen von 3 Bytes und merkt es doch nicht

                d = self.fileHandle.read(3)

                if d == None:

                    self.fileHandle.close()

                    #return invalid

                    size = 3\*(lauf-1)

                else:

                    #dataraw = unpack('<i', d + (0x00 if d[2] < 128 else 0xff))

                    dataraw = unpack('<%ul' % 1 ,d + (b'\x00' if d[2] < 128 else b'\xff'))

Ausserdem: Versetztes temp-File wird korrekt erzeugt (ist in SDRUno abspielbar) aber EXTREM langsam bei 24 bit !

Lösung: Inactivate LOSHIFT für diesen Fall !

Beim Abspielen Dat-File über neuen wav-headergenerator wird die Stopzeit offenbar falsch berechnet: 2x soviel eingetragen wir wirklich abgespielt. Die Spielzeit stimmt, es ist also ein Berechnungsfehler der STOPTIME; bytes per second werden im wavheadergenerator falsch berecnet

Für RFCorder only: Inactivate Tabs with:

self.ui.tabWidget.setTabVisible(index, False)

Weiterschalten auf nextfile funktioniert nach dem 2. Mal nicht mehr, neues File hat beim Auslesen des Wav-headers einen falschen Eintrag, der in Wirklichkeit aber richtig ist. Traceback:

Beim zweiten Mal wird ‚file opened in playloop…‘ wie beim ersten Mal aufgerufen, auch wird ein Playthread gestartet. Unterschied ab jetzt: fetch nextfile, danach erfolgt SigFinished und Stop, statt weiterspielen

EOF wird beim zweiten Mal gar nicht mehr erreicht

Liegt daran, dass im wavheader ein ‚b'SDRuno\_20220910\_210448Z\_1125kHz.wav‘ bei nextfile gelesen wird, obwohl das im wav-headerreader nicht angezeigt wird.

Problem: nextfilename wird manchmal nicht als utf-8 decodiert und der String ennthält ein „b‘ zu Beginn. Lösung:

Statt # schreib die 2. Zeile:

            #aaa = str(self.fileHandle.read(96))

            aaa = (self.fileHandle.read(96)).decode('utf-8')

Wenn neuer Tab „Resampler“ eingebaut werden soll, müssen etliche Stellen angepasst werden. Wie kann man das programmtechnisch vereinfachen ?

Neuvergabe von Namen nach Namenskonvention:

Alle Elemente der Tabs Player und View Spectra wurden umbenannt; GUI\_Version = 6, Wizard = Ver10;

Fehlen alle anderen Tabs

Umbenennen alle GUI\_Callbacks: erledigt für Player und View Spectra

Zeitfresser: Bei plot\_spectrum sehr viel Zeitverlust wegen: ann\_spectrum

Dort wieder die Ursache ist medfilt !

Nach Recherche:

from scipy.ndimage.filters import median\_filter

Anstatt: #databasel = sig.medfilt(datay,kernel\_length)

databasel = median\_filter(datay,kernel\_length, mode = 'constant')

ist > Faktor 5 schneller

#Die tatsächliche File-Time wird noch nicht angezeigt im Zeitfenster des Players

False gesetzt in: open\_template\_file(self):

Open\_file(),

Layout Header editor schlecht: erste Spalte zu schmal

self.myfilename und self.ext werden an 2 Stellen erzeugt: FileOpen() und showfilename()

IP address wird immer wieder mal auf 0 zurückgesetzt, Ursache ??? CHECK

Evaluation time Label über den Scrollbar setzen

Tab f Resampler nach hinten schieben

Deaktiviere alle Zugriffe auf das File, wenn der Player läuft oder deaktiviere den Player, wenn ein anderes Widget mit Dateizugriff aktiviert wird !

Bei wavheader wurde ein Feld [„starttime\_dt“] hinzugefügt, das die startzeit nun unabhängig von AUXI oder RCVR als datetime enthält. Im Hauptcode sollen nun alle Instanzen, wo je nach SDRTyp starttime unterschiedlich behandelt wird, vereinheitlicht werden.

Wenn noch kein File geladen und man benutzt Spinbox für Filterkernel 🡪 Absturz: inactivate until file loaded

Baue im Scannerteil eine Amplitudenanzeige ein (f. Korrekturmöglichkeiten)

Übernimm bei Annotation die Edit-Line im Kopf anstatt der Original-Tabelleneinträge ins Cohiradia-yaml, um nacheditieren zu können.

Neue Version: 3 Edit Lines für Country, program und tx-Site, alle 3 werden editiert, 3 Knöpfe mit ‘ENTER‘ und ‚DISCARD noch nötig ???‘

Bei Senderanzeigen im Edit Feld wird right alligned, sollte left sein um Frequenz nicht zu verdecken

            self.ui.lineEdit.setAlignment(QtCore.Qt.AlignLeft)

            self.ui.lineEdit.home(True)

Zum Peakfinden müssen self.DELTAF und self.PEAKWIDTH brauchbar gesetzt sein. Aktuell kann man diese Settings nicht von aussen ändern. Idee: Nach ‚metadata‘ verschieben oder editable machen in eigenem ‚CONFIG‘ Tab

Einfache Dat-wav Methode:

Zuerst mit normalem File open aber neues Untermenü ‚Open Template Header‘ den Template Header laden. Danach wird das Dat File geladen: Open file liest dabei nur den Filenamen und tut sonst nichts.

Wenn beim Dateityp dat gewählt wird, und man Scan macht 🡪 Absturz

* Deaktiviere ganze Tabs bis auf wavedit : self.ui.tab.setEnabled(False)
* self.ui.tab\_1.setEnabled(False)…etc, auch mit True

es gibt tabs tab, tab\_1, tab\_2, tab\_3, entspr dem QTDesigner

* Schreib irgendwo in den Status, dass jetzt eine Datfilemanipulation erfolgt
* Lade wav-Templateheader
* Aktiviere Button ‚write header‘ nach Headerkonsistenzcheck
* Konsistenzcheck: Melde, wenn Felder nicht ausgefüllt, z.B. Starttime, Stoptime, ?nextfilename ? ?Sonderzeichen in nextfilename
* Benenne File um nach wav-Header Info
* Bei Klick oder auf ‚write header‘ schreib wavheader in die ersten 216 bytes des geöffneten Files und benenne das File um

Overwrite header für wav files im Filemenü belassen

Updating von Statusbars, Scrollbars: evt über Funktionsaufruf und schreiben ins Editfeld/Statusfeld

**Am besten Threads:**

Worker Klasse mit den Funktionen Autoscan = jetziges Autoscan

* Er wird im Callback von Button ‚Autoscan‘ instanziert, einem Thread zugeordnet derselbe wird gestartet
* Autoscan sendet ein Signal, wenn Statusbar und Editfenster zu aktualisieren sind,
* Signal connects zu einer Updatemethode für Statusbar und Editfenster
* Autoscan sendet ein Signal SigFinished, wenn Task beendet, damit wird dann der Thread gestoppt

Ebenso eine Worker-Klasse für plot\_spectrum(self,position) wenn der Slider bewegt wird

**Threading:**

***In \_\_main\_\_:***

app.aboutToQuit.connect(win.stop\_worker) #graceful thread termination on app exit

***in Hauptklasse:***

def stop\_worker(self):

if self.playthreadActive:

self.playrec.stop\_loop()

self.timertick.stoptick()

***Thread-Workerklasse:***

class timer\_worker(QObject):

SigTick = pyqtSignal()

SigFinished = pyqtSignal()

def \_\_init\_\_(self, \*args, \*\*kwargs):

super().\_\_init\_\_(\*args, \*\*kwargs)

def tick(self):

while True:

time.sleep(1 - time.monotonic() % 1)

self.SigTick.emit()

def stoptick(self):

self.SigFinished.emit()

Aktivierung des Threads wenn genehm entw in einer Methode oder einem \_\_init\_\_:

self.timethread = QThread()

self.timertick = timer\_worker()

self.timertick.moveToThread(self.timethread)

self.timethread.started.connect(self.timertick.tick)

self.timertick.SigFinished.connect(self.timethread.quit)

self.timertick.SigFinished.connect(self.timertick.deleteLater)

self.timethread.finished.connect(self.timethread.deleteLater)

self.timertick.SigTick.connect(self.HAUPTFUNKTION)

self.timethread.start()

if self.timethread.isRunning():

self.timethreaddActive = True

**TESTEN: Wann welche Buttons der Annotation aktivieren/deaktivieren:**

**Bei Programmstart im init:**

Annotate und Scan und Discard inactivate

**Nach Programmöffung**

if exist Annotationdir

If status annotated == True

* + Annotation completed
  + Scan completed

Else

* Scan completed

Else

* Scan activate
* (Annotate inactivate)
* (Discard inactivate)

**Nach fertigem Scan**

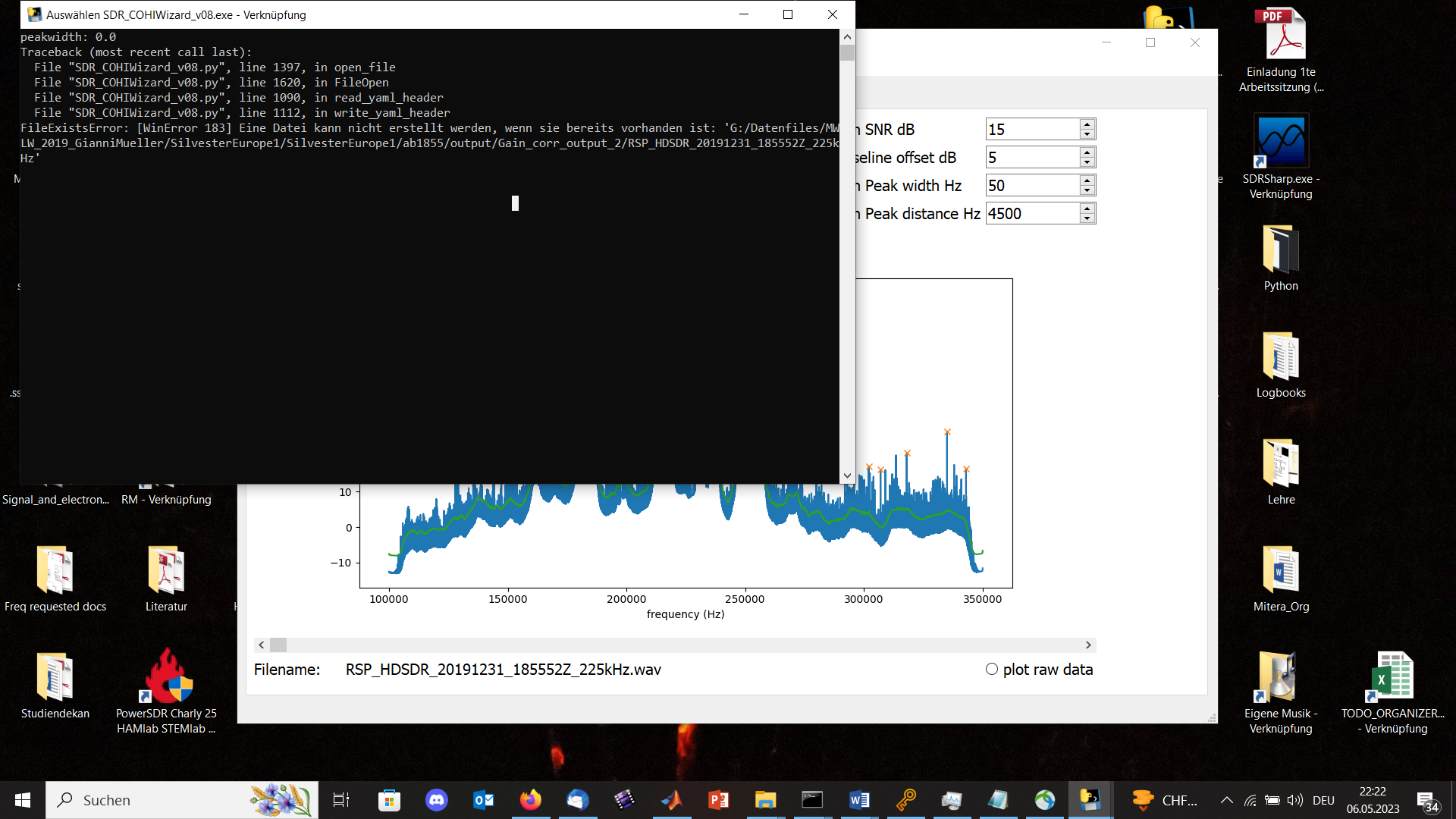
* Scan completed
* Discard inactivate
* Annotate activate

**Bei Beginn Annotation:**

* Discard activate
* (Scan completed)

**Nach fertigem Annotate:**

* Annotation completed
* Discard inactivate
* (Scan completed)



**Könnte hier passieren:**

if os.path.exists(self.cohiradia\_yamlheader\_filename) == False:         #exist yaml file: create from yaml-editor

            self.cohiradia\_yamlheader\_dirname = self.my\_dirname + '/' + self.my\_filename

            os.mkdir(self.cohiradia\_yamlheader\_dirname)

# Requirements:

C:\Users\scharfetter\_admin>pip list

Package Version

----------------------------- -----------

aiofiles 22.1.0

aiosqlite 0.18.0

alabaster 0.7.12

altgraph 0.17.2

anyio 3.6.2

argon2-cffi 21.3.0

argon2-cffi-bindings 21.2.0

arrow 1.2.3

asttokens 2.2.1

async 0.6.2

asyncio 3.4.3

asyncssh 2.9.0

attrs 22.2.0

auto-py-to-exe 2.17.0

Babel 2.12.1

backcall 0.2.0

bcrypt 3.2.0

beautifulsoup4 4.11.1

bleach 6.0.0

bottle 0.12.19

bottle-websocket 0.2.9

bs4 0.0.1

certifi 2021.10.8

cffi 1.15.0

charset-normalizer 2.0.12

click 7.1.2

colorama 0.4.4

comm 0.1.2

cryptography 36.0.0

cx-Freeze 6.6

cx-Logging 3.0

cycler 0.10.0

debugpy 1.6.6

decorator 5.1.1

defusedxml 0.7.1

docutils 0.17.1

Eel 0.12.4

et-xmlfile 1.1.0

executing 1.2.0

fastjsonschema 2.16.3

fqdn 1.5.1

future 0.18.2

gevent 21.12.0

gevent-websocket 0.10.1

greenlet 1.1.2

h5py 3.6.0

idna 3.3

imagesize 1.3.0

importlib-metadata 4.11.3

install 1.3.5

ipykernel 6.21.3

ipython 8.11.0

ipython-genutils 0.2.0

iso8601 1.0.2

isoduration 20.11.0

jedi 0.18.2

Jinja2 3.1.1

json5 0.9.11

jsonpointer 2.3

jsonschema 4.17.3

jupyter\_client 8.0.3

jupyter\_core 5.2.0

jupyter-events 0.6.3

jupyter\_server 2.4.0

jupyter\_server\_fileid 0.8.0

jupyter\_server\_terminals 0.4.4

jupyter\_server\_ydoc 0.6.1

jupyter-ydoc 0.2.3

jupyterlab 3.6.1

jupyterlab-pygments 0.2.2

jupyterlab\_server 2.20.0

kiwisolver 1.3.1

MarkupSafe 2.1.1

matplotlib 3.4.1

matplotlib-inline 0.1.6

mistune 2.0.5

mpmath 1.2.1

nbclassic 0.5.3

nbclient 0.7.2

nbconvert 7.2.9

nbformat 5.7.3

nest-asyncio 1.5.6

networkx 1.11

notebook 6.5.3

notebook\_shim 0.2.2

numpy 1.22.4

openpyxl 3.1.2

packaging 21.3

pandas 1.5.3

pandocfilters 1.5.0

paramiko 2.8.0

parso 0.8.3

pefile 2021.9.3

pickleshare 0.7.5

Pillow 8.2.0

pip 22.0.2

platformdirs 3.1.1

ply 3.11

prometheus-client 0.16.0

prompt-toolkit 3.0.38

psutil 5.9.4

pure-eval 0.2.2

pycparser 2.21

Pygments 2.11.2

pyinstaller 4.8

pyinstaller-hooks-contrib 2021.5

PyNaCl 1.4.0

pyparsing 2.4.7

PyQt5 5.15.4

pyqt5-plugins 5.15.4.2.2

PyQt5-Qt5 5.15.2

PyQt5-sip 12.8.1

pyqt5-tools 5.15.4.3.2

pyrsistent 0.19.3

PySide2 5.15.2.1

PySoundFile 0.9.0.post1

python-dateutil 2.8.2

python-dotenv 0.19.2

python-json-logger 2.0.7

pytz 2022.1

PyVISA 1.11.3

PyVISA-py 0.5.2

pywin32 305

pywin32-ctypes 0.2.0

pywinpty 2.0.10

PyYAML 6.0

pyzmq 25.0.1

qt5-applications 5.15.2.2.2

qt5-tools 5.15.2.1.2

requests 2.28.2

rfc3339-validator 0.1.4

rfc3986-validator 0.1.1

scipy 1.7.3

Send2Trash 1.8.0

serial 0.0.97

setuptools 59.2.0

shiboken2 5.15.2.1

six 1.15.0

smop 0.41

sniffio 1.3.0

snowballstemmer 2.2.0

soupsieve 2.3.2.post1

source 1.2.0

Sphinx 4.5.0

sphinx-rtd-theme 1.0.0

sphinxcontrib-applehelp 1.0.2

sphinxcontrib-devhelp 1.0.2

sphinxcontrib-htmlhelp 2.0.0

sphinxcontrib-jsmath 1.0.1

sphinxcontrib-qthelp 1.0.3

sphinxcontrib-serializinghtml 1.1.5

stack-data 0.6.2

sympy 1.11.1

terminado 0.17.1

tinycss2 1.2.1

tomli 2.0.1

tornado 6.2

traitlets 5.9.0

typing-extensions 3.10.0.0

uri-template 1.2.0

urllib3 1.26.9

wcwidth 0.2.6

webcolors 1.12

webencodings 0.5.1

websocket-client 1.5.1

wheel 0.37.0

whichcraft 0.6.1

xlrd 2.0.1

y-py 0.5.9

ypy-websocket 0.8.2

zipp 3.4.1

zope.event 4.5.0

zope.interface 5.4.0

# Programmstruktur

# Methods

## Klassendiagramm automatisch erstellen:

Pip install pylint,

pyreverse laufen lassen, benötigt Graphviz

pyreverse -o png <PATH>

funktioniert leider nicht, da Graphviz nicht erkannt wird.

## Player:

Click Play button: toggle function/icon to Pause state and vice versa

def togglePlay(self):

if self.ui.pushButton\_Play.isChecked() == True:

if not self.fileopened:

self.open\_file()

if not self.checkSTEMLABrates():

return False

self.ui.pushButton\_Play.setIcon(QIcon("pause\_v4.PNG"))

if self.playthreadActive == True:

self.playrec.pausestate = False

self.GuiClickPlay()

self.pausestate = False

#TODO: implement pause

else:

self.ui.pushButton\_Play.setIcon(QIcon("play\_v4.PNG"))

self.pausestate = True

if self.playthreadActive == True:

self.playrec.pausestate = True

Endlosplay:

Playloop: sendet

Wenn ‘nextfilename’ :

**Playthread\_start()**

startet den nächsten Playthread mit den Signalconnections:

self.playrec.SigFinished.connect(self.playthread.quit)

self.playrec.SigFinished.connect(self.playrec.deleteLater)

self.playrec.SigFinished.connect(self.GuiClickStop)

self.playthread.finished.connect(self.playthread.deleteLater)

self.playrec.SigIncrementCurTime.connect(

lambda: self.updatecurtime(1))

………

if self.playthread.isRunning():

self.playthreadActive = True

Wenn also ein File zu ende ist, wird über SigFinished self.GUIClickStop ausgelöst.

Allerdings wird dieses auch direkt über den STOP Button ausgelöst.

In self.GUIClickStop passiert:

if self.playthreadActive:

self.playrec.stop\_loop()

if self.TEST is False:

stemlabcontrol.sdrserverstop()

Ich möchte nun im Fall, dass nextfile existiert das File nextfile abspielen.

Vorgeschlagene Version: über SigFinished wird ausgelöst, dass im Falle eines nextfiles ein neuer Playthread gestartet wird wie von der Playtaste aus. Dazu könnte man einfach self.f1 auf nextfile setzen und playthread\_start() aufrufen.

In diesem Zuge könnte man auch die Autorepeat-Taste ausführen, indem statt nextfile der Pfad auf das aktuell geladene File in self.f1 bleibt. Man muss also gar nichts ändern, nur playthread\_start() aufrufen.

Bis dato: GUIClickStop bei SigFinished; Das beendet aber die gesamte Serverkommunikation.

Stattdessen: SigFinished connects an neue Methode playloop\_EOFmanager()

If nextfile 🡪 self.f1 = nextfile

## ann\_spectrum(self, dummy, data):

This is a Python function that takes in two arguments, a dummy variable and a data array. The function is designed to perform spectral analysis on the data and identify peaks in the resulting spectrum.

The function first extracts the real and imaginary parts from the complex data array using numpy's "arange" function. It then calculates the spectrum using numpy's "fft" function and takes the absolute value of the result using numpy's "abs" function. The resulting spectrum is then shifted and rescaled appropriately using numpy's "fftshift" function and some calculations based on the header information stored in a dictionary named "wavheader".

Next, the function applies a filter to the spectrum to remove all data below the baseline. The filter is based on a moving median and is applied using numpy's "medfilt" function. The filter kernel size is determined by dividing the spectrum size by the "FILTERKERNEL" attribute of the class.

Finally, the function uses scipy's "find\_peaks" function to identify all peaks in the filtered spectrum that are above a certain prominence threshold (set by the "PROMINENCE" attribute of the class) and have a minimum distance (set by the "DELTAF" attribute of the class) and a minimum width (set by the "PEAKWIDTH" attribute of the class). The function returns a dictionary containing various arrays related to the spectral analysis, including the frequency and amplitude data, the filtered amplitude data, the locations and properties of the identified peaks, and the baseline data used in the filtering process.

## %Annotation anhand der MWList-Tabelle alt:

Listen clickable Machen: https://gist.github.com/Orizzu/e47393efe37c9e4846f7c23f2b10c4a7

# you can copy paste and run this code for test

from PyQt5.QtGui import \*

from PyQt5.QtCore import \*

from PyQt5.QtWidgets import \*

import sys

class window(QMainWindow):

def \_\_init\_\_(self):

super(window, self).\_\_init\_\_()

listWidget = QListWidget()

listWidget.resize(300,120)

listWidget.addItem("Item 1")

listWidget.addItem("Item 2")

listWidget.addItem("Item 3")

listWidget.addItem("Item 4")

listWidget.itemClicked.connect(self.Clicked) # connect itemClicked to Clicked method

self.setCentralWidget(listWidget)

def Clicked(self,item):

QMessageBox.information(self, "ListWidget", "You clicked: "+item.text())

def main():

app = QApplication(sys.argv)

w = window()

w.show()

sys.exit(app.exec\_())

if \_\_name\_\_ == '\_\_main\_\_':

main()

# GIT

git init

git status

git add <FILE>

git commit -m "SOME WEIRD MESSAGE" -m "SOME DESCRIPTION"

git remote add origin https://github.com/hermy-sf/COHIRADIA\_AUX.git (Link ist nur ein Beispiel !)

git remote -v

git branch -M main

git push origin main

#Um immer eine Kurzpush-Version zu haben:

git push -u origin main

#dann reicht 'git push' allein

# conda

AKTUELLE Interpreter-Version:

conda info –env

conda activate <environment>

conda deactivate

# Threading know how:

Zum Thema, wie Threads mit anderen oder dem Main window kommunizieren sollen hier:

<https://stackoverflow.com/questions/33081529/in-pyqt-what-is-the-best-way-to-share-data-between-the-main-window-and-a-thread>

**Vorschlag eines Users:**

I'm in the process of writing my very first GUI application with PyQt4 and I've come upon a question that seems very basic, yet I don't seem to find a good answer:

I'm using a thread to continuously perform a repeated task without blocking the main window. The thread needs some information from the main window (e.g. the current value of a spinbox) that can also change during the runtime of the thread. In this situation, what is the proper way to share such data between the main window and the thread?

Naively, I could come up with the following possibilities:

1. Pass a reference to the host window to the thread and use this to retrieve the current value of the variable in question (see example below).
2. Keep a copy of the variable in the thread and keep it synchronized with the main window by emitting signals whenever it changes.
3. Use global variables.

All three options would most likely work for my particular use case (though 2 would be a bit complicated), but I have a feeling there should be a better/more Pythonic/more Qt-like way.

Here is a minimum working example illustrating what I want to do, in this case using option 1:

from PyQt4 import QtGui, QtCore

import time, sys

class MainWindow(QtGui.QWidget):

def \_\_init\_\_(self):

super(MainWindow, self).\_\_init\_\_()

self.layout = QtGui.QVBoxLayout(self)

self.spinbox = QtGui.QSpinBox(self)

self.spinbox.setValue(1)

self.layout.addWidget(self.spinbox)

self.output = QtGui.QLCDNumber(self)

self.layout.addWidget(self.output)

self.worker = Worker(self)

self.connect(self.worker, QtCore.SIGNAL('beep'), self.update)

self.worker.start()

def update(self, number):

self.output.display(number)

class Worker(QtCore.QThread):

def \_\_init\_\_(self, host\_window):

super(Worker, self).\_\_init\_\_()

self.host = host\_window

self.running = False

def run(self):

self.running = True

i = 0

while self.running:

i += 1

self.emit(QtCore.SIGNAL('beep'), i)

sleep\_time = self.host.spinbox.value()

time.sleep(sleep\_time)

def stop(self):

self.running = False

app = QtGui.QApplication(sys.argv)

window = MainWindow()

window.show()

app.exec\_()

**Antwort:**

Widgets are not thread safe, see [Threads and QObjects](http://doc.qt.io/qt-4.8/threads-qobject.html):

Although QObject is reentrant, the GUI classes, notably QWidget and all its subclasses, are not reentrant. They can only be used from the main thread.

And see more definitions here: [Reentrancy and Thread-Safety](http://doc.qt.io/qt-4.8/threads-reentrancy.html)

You should only use widgets in the main thread, and use signal and slots to communicate with other threads.

I don't think a global variable would work, but I honestly don't know why.

How to use signals in this example:

#in main

self.worker = Worker(self.spinbox.value())

self.worker.beep.connect(self.update)

self.spinbox.valueChanged.connect(self.worker.update\_value)

class Worker(QtCore.QThread):

beep=QtCore.pyqtSignal(int)

def \_\_init\_\_(self,sleep\_time):

super(Worker, self).\_\_init\_\_()

self.running = False

self.sleep\_time=sleep\_time

def run(self):

self.running = True

i = 0

while self.running:

i += 1

self.beep.emit(i)

time.sleep(self.sleep\_time)

def stop(self):

self.running = False

def update\_value(self,value):

self.sleep\_time=value

# WAV-Header Details zu wBlockalign und wFormatTag

Info von Gianni Müller:

Was die Tags im Header angeht, bin aber bei einer Recherche danach auf folgende Infos gestossen:

So wie ich das nun sehe:

[Wave File Specifications (mcgill.ca)](https://www.mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html)

[DIGITAL AUDIO: RIFF WAVE file format (unimi.it)](https://www.lim.di.unimi.it/IEEE/VROS/RIFF.HTM)

Auch sehr gut (generell für AUDIO-Files):

<https://www.mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html>

**wFormatTag**     A number indicating the WAVE format

                              category of the file. The content of

                              the <format-specific-fields> portion

                              of the `fmt' chunk, and the

                              interpretation of the waveform data,

                              depend on this value.

**wBlockAlign**        The block alignment (in bytes) of the

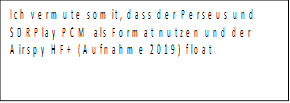
                              waveform data. Playback software needs

                              to process a multiple of wBlockAlign

                              bytes of data at a time, so the value

                              of wBlockAlign can be used for buffer

                              alignment.

C:\Users\scharfetter_admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\7BE38E2F.tmpC:\Users\scharfetter_admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\E3B32715.tmpC:\Users\scharfetter_admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\2301BEEB.tmpC:\Users\scharfetter_admin\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\9D44E331.tmp*The standard format codes for waveform data are given below.*

|  |  |  |
| --- | --- | --- |
| ***Format Code*** | ***PreProcessor Symbol*** | ***Data*** |
| *0x0001* | *WAVE\_FORMAT\_PCM* | *PCM* |
| *0x0003* | *WAVE\_FORMAT\_IEEE\_FLOAT* | *IEEE float* |
| *0x0006* | *WAVE\_FORMAT\_ALAW* | *8-bit ITU-T G.711 A-law* |
| *0x0007* | *WAVE\_FORMAT\_MULAW* | *8-bit ITU-T G.711 µ-law* |
| *0xFFFE* | *WAVE\_FORMAT\_EXTENSIBLE* | *Determined by SubFormat* |

***PCM Format***

*The first part of the Format chunk is used to describe PCM data.*

         *For PCM data, the Format chunk in the header declares the number of bits/sample in each sample (wBitsPerSample). The original documentation (Revision 1) specified that the number of bits per sample is to be rounded up to the next multiple of 8 bits. This rounded-up value is the container size. This information is redundant in that the container size (in bytes) for each sample can also be determined from the block size divided by the number of channels (nBlockAlign / nChannels).*

o    *This redundancy has been appropriated to define new formats. For instance, Cool Edit uses a format which declares a sample size of 24 bits together with a container size of 4 bytes (32 bits) determined from the block size and number of channels. With this combination, the data is actually stored as 32-bit IEEE floats. The normalization (full scale 223) is however different from the standard float format.*

         *PCM data is two's-complement except for resolutions of 1-8 bits, which are represented as offset binary.*

***Non-PCM Formats***

*An extended Format chunk is used for non-PCM data. The cbSize field gives the size of the extension.*

         *For all formats other than PCM, the Format chunk must have an extended portion. The extension can be of zero length, but the size field (with value 0) must be present.*

         *For float data, full scale is 1. The bits/sample would normally be 32 or 64.*

         *For the log-PCM formats (µ-law and A-law), the Rev. 3 documentation indicates that the bits/sample field (wBitsPerSample) should be set to 8 bits.*

         *The non-PCM formats must have a fact chunk.*

***Extensible Format***

*The WAVE\_FORMAT\_EXTENSIBLE format code indicates that there is an extension to the Format chunk. The extension has one field which declares the number of valid bits/sample (wValidBitsPerSample). Another field (dwChannelMask) contains bits which indicate the mapping from channels to loudspeaker positions. The last field (SubFormat) is a 16-byte globally unique identifier (GUID).*

         *With the WAVE\_FORMAT\_EXTENSIBLE format, the original bits/sample field (wBitsPerSample) must match the container size (8 \* nBlockAlign / nChannels). This means that wBitsPerSample must be a multiple of 8. Reduced precision within the container size is now specified by wValidBitsPerSample.*

         *The number of valid bits (wValidBitsPerSample) is informational only. The data is correctly represented in the precision of the container size. The number of valid bits can be any value from 1 to the container size in bits.*

         *The loudspeaker position mask uses 18 bits, each bit corresponding to a speaker position (e.g. Front Left or Top Back Right), to indicate the channel to speaker mapping. More details are in the document cited above. This field is informational. An all-zero field indicates that channels are mapped to outputs in order: first channel to first output, second channel to second output, etc.*

         *The first two bytes of the GUID form the sub-code specifying the data format code, e.g. WAVE\_FORMAT\_PCM. The remaining 14 bytes contain a fixed string, \x00\x00\x00\x00\x10\x00\x80\x00\x00\xAA\x00\x38\x9B\x71.*

*The WAVE\_FORMAT\_EXTENSIBLE format should be used whenever:*

         *PCM data has more than 16 bits/sample.*

         *The number of channels is more than 2.*

         *The actual number of bits/sample is not equal to the container size.*

         *The mapping from channels to speakers needs to be specified.*

* C Funktion aus dem Code von dem SDR Programm Linrad gestossen:

Weiter bin ich beim Thema WAV verarbeiten noch auf folgenden C Code aus dem SDR Programm Linrad gestossen:

[linrad/wsetad.c at master · peterwillcn/linrad · GitHub](https://github.com/peterwillcn/linrad/blob/master/wsetad.c)

 SIEHE sox: https://sox.sourceforge.net/sox.html

**−e** *ENCODING*, **−−encoding** *ENCODING*

The audio encoding type. Sometimes needed with file-types that support more than one encoding type. For example, with raw, WAV, or AU (but not, for example, with MP3 or FLAC). The available encoding types are as follows:  **signed-integer**

PCM data stored as signed (‘two’s complement’) integers. Commonly used with a 16 or 24 −bit encoding size. A value of 0 represents minimum signal power.

**unsigned-integer**

PCM data stored as unsigned integers. Commonly used with an 8-bit encoding size. A value of 0 represents maximum signal power.

**floating-point**

PCM data stored as IEEE 753 single precision (32-bit) or double precision (64-bit) floating-point (‘real’) numbers. A value of 0 represents minimum signal power.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **a-law** |  | International telephony standard for logarithmic encoding to 8 bits per sample. It has a precision equivalent to roughly 13-bit PCM and is sometimes encoded with reversed bit-ordering (see the **−X** option). |

**u-law, mu-law**

North American telephony standard for logarithmic encoding to 8 bits per sample. A.k.a. μ-law. It has a precision equivalent to roughly 14-bit PCM and is sometimes encoded with reversed bit-ordering (see the **−X** option).

**oki-adpcm**

OKI (a.k.a. VOX, Dialogic, or Intel) 4-bit ADPCM; it has a precision equivalent to roughly 12-bit PCM. ADPCM is a form of audio compression that has a good compromise between audio quality and encoding/decoding speed.

**ima-adpcm**

IMA (a.k.a. DVI) 4-bit ADPCM; it has a precision equivalent to roughly 13-bit PCM.

**ms-adpcm**

Microsoft 4-bit ADPCM; it has a precision equivalent to roughly 14-bit PCM.

**gsm-full-rate**

GSM is currently used for the vast majority of the world’s digital wireless telephone calls. It utilises several audio formats with different bit-rates and associated speech quality. SoX has support for GSM’s original 13kbps ‘Full Rate’ audio format. It is usually CPU-intensive to work with GSM audio.

|  |  |  |
| --- | --- | --- |
|  |  |  |

|  |
| --- |
| void fill\_wavefmt(int channels, int bits, int flag, int speed)  {  fmt.Format.nSamplesPerSec=speed;  if(bits==16)    {    fmt.Samples.wValidBitsPerSample=16;    fmt.Format.wBitsPerSample=16;    if( flag == 0)      {      fmt.Format.wFormatTag=WAVE\_FORMAT\_PCM;      fmt.Format.cbSize=0;      }    else      {      fmt.Format.wFormatTag = WAVE\_FORMAT\_EXTENSIBLE;      fmt.Format.cbSize=sizeof(WAVEFORMATEXTENSIBLE);      fmt.SubFormat = KSDATAFORMAT\_SUBTYPE\_PCM;      }    }  else    {    if( flag == 0)      {      lirerr(2654844);      return;      }    fmt.Format.wFormatTag = WAVE\_FORMAT\_EXTENSIBLE;    fmt.Samples.wValidBitsPerSample=24;    fmt.Format.wBitsPerSample=32;    fmt.Format.cbSize=sizeof(WAVEFORMATEXTENSIBLE);    fmt.SubFormat = KSDATAFORMAT\_SUBTYPE\_PCM;    }  fmt.Format.nChannels=channels;  switch (channels)    {    case 1:    fmt.dwChannelMask = 1;    break;      case 2:    fmt.dwChannelMask = 3;    break;      case 4:    fmt.dwChannelMask = 0x33;    break;      default:    lirerr(2963432);    return;    }  fmt.Format.nBlockAlign=fmt.Format.nChannels\*fmt.Format.wBitsPerSample/8;  fmt.Format.nAvgBytesPerSec=fmt.Format.nBlockAlign\*fmt.Format.nSamplesPerSec;  } |

Eventuell hilft dir diese Info für dein Projekt etwas weiter. Leider bin ich mit der Materie nicht seht

Konsequenz in Resampler:

**if Bits\_p\_sample == 16**

**reflevel = 32767;**

**timescaler=C\_\_SOURCESR\*4;**

**else**

**if Bits\_p\_sample ~= 32**

**error('bits per sample neither 16 nor 32, cannot handle such files')**

**end;**

**reflevel = 1;**

**timescaler=C\_\_SOURCESR\*8;**

**end**

**if Bits\_p\_sample == 16**

**[data1,count]=fread(fidr,C\_\_SAMPLELENGTH,'int16');**

**data=cast(data1/32767,'double');**

**else**

**%read two int32 for 1 int64**

**%[data,count]=fread(fidr,C\_\_SAMPLELENGTH,'single');**

**[data,count]=fread(fidr,floor(C\_\_SAMPLELENGTH),'float32');**

**end;**

# virtual environment:

Für venv:

Generieren eines requirement-Files aus der aktuellen Basisinstallation:

pip list --format=freeze > requirements.txt

venv erstellen: im Arbeitsdirectory:

python –m venv venv

venv aktivieren:

~\.venv\Scripts\activate

In .venv\Scripts befindet siahc auch das lokale pip, nehme an, das installiert dann dort alles

**ERFOLGREICHE IMPLEMENTIERUNG 12-2023**

**Ab jetzt: Installieren neuer Packages nur in aktivierten venv !**

In der Commandline gehe zum Working-Directory

Erstelle ein neue venv mit:

python –m venv venv

Habe aus dem Default-Python-Mode (ohne Aktivierung ds venv) ein requirements.txt erzrugt mit

pip list --format=freeze > requirements.txt

aktiviere das venv:

.venv\Scripts\activate

Nun alle Dependencies installieren:

pip install -r requirements.txt

Das ist einigemale schiefgegangen, weil Packages enthalten waren, die nicht aufgelöst wurden, oder dann zur Setup-Zeit zu Errors oder Konflikten führten. Ich habe alle durch Entfernen der betroffenen Packages gelöst, danach war das Setup erfolgreich und COHIWizard\_v26.py konnte mit pythyon ### von der commandline gestartet werden.

In VScode starte ich auf dem Terminal:

conda deactivate

.venv\Scripts\activate

Danach ctrl-shift P (Command Palette): Select Interpreter 🡪 Wähle den .venv-Pfad aus.

Hat nicht gkeich beim ersten Mal geklappt, aber nach Wiederholen dann doch. Scheint inzwischen zu funktionieren.

Requirements-File ist gepurged und sollte beibehalten werden.

**Ab jetzt: Installieren neuer Packages nur in aktivierten venv !**

############END sofar

conda init cmd.exe um cmd für conda zu konfigurieren

conda activate <virtual environment> zum Aktivieren 🡪 PATH wird gesetzt

conda deactivate

conda env export > environment.yml

# Dokumentation

## Generelles

Bis Version 19 mit QT\_GUI COHIWizard\_GUI\_v7 gibt es die beiden Funktionen ‚load template header‘ und ‚save header to template‘. Ab V20 mit COHIWizard\_GUI\_v8 sind diese beiden Funktionen obsolet.

Starten des Programms durch Doppelklick auf das File-Symbol ###########. Es öffnet sich das Programmfenster auf dem Tab ‚Scanner‘ und zeigt ein leeres Diagramm-Fenster sowie ein paar Steuer- und Anzeigeelemente.

|  |
| --- |
|  |

Vor jeder weiteren Aktion muss ein File geöffnet werden. Dazu gibt es das üblich ‚File‘-Menü links oben.

|  |
| --- |
|  |

## Aufgabe/Tab ‚Scanner‘: Aufnahme visuell analysieren

Im Tab ‚Scanner‘ kann man die Aufnahme mit einem Rollbalken durchlaufen und das Spektrum zu unterschiedlichen Zeitpunkten anzeigen lassen. Dazu muss beim Öffnen des Files ein wav-File mit dem SDRUno-Format (IQ-Datenfile) geöfnet worden sein. Der Filename wird an der Unterkante des Fensters angezeigt. Die Anzeige eines Spektrums erfolgt jeweils, wenn man den Knopf des Rollbalkens loslässt. Wie in Abb. 3 ersichtlich, kann man kann im Werkzeugmenü oberhalb der Grafik zoomen (Lupe), mit Maustasten zoomen/schieben (Kreuzsymbol für Pan/Zoom) und einige Grafikeigenschaften ändern. Auch ein Abspeichern der Grafik ist möglich.

|  |
| --- |
| Mindestabstand zw. peaks  Mindestpeakbreite  Gewünschte Lage der grün dargestellten Basislinie  Minimale für die Annotation gewünschte SNR  Werkzeuge zum Zoomen, Schieben, Speichern, … der Grafik  Zeitpunkt des Spektrums in der Aufnahme |
| Abb. 3: Anzeige eines Spektrums in der Aufnahme |

Der Zeitpunkt des Spektrums wird im grauen Feld oberhalb der Grafik in UTC eingeblendet.

Fährt man mit dem Mauszeiger über einzelne Peaks des Spektrums, werden die Koordinaten des Mauszeigers eingeblendet, so kann man das Spektrum grob ‚ausmessen‘.

Die beiden Spinboxen ‚min SNR‘ und ‚baseline offset‘ sind in diesem Fenster noch nicht von zentraler Bedeutung, aber wichtig für die spätere Annotation. Daher sollte man die Werte bereits hier einstellen: Mit ‚baseline offset‘ kann man die grüne im Spektrum eingezeichnete Basislinie nach oben/unten schieben, wenn sie nicht gut passt. Mit den Pfeiltasten der Spinboxen kann man in 5 dB-Schritten springen, für Zwischenwerte muss man entsprechende Zahlen ins Zahlenfeld eingeben. Von der so festgelegten Basislinie aus wird dann die SNR für die Senderpeaks in dB gemessen.

In der Spinbox ‚min SNR‘ stellt man den Schwellenwert ein, ab dem ein Peak beim Annotieren als Sender identifiziert werden soll.

Die beiden Spinboxen ‚min Peak width‘ und ‚min Peak distance‘ legen fest, welche Mindestbreite und welchen Mindestabstand zu den Nachbarn ein Peak haben sollte. Mindestwerte sind 1 Hz bzw. 100 Hz. Standardmäßig sind 50 Hz und 4500 Hz (das berücksichtigt das Kurzwellenraster) eingestellt. Bei jedem Verändern eines dieser beiden Werte wird die Grafik aktualisiert, sodass man sieht, welchen Effekt die Einstellung auf die Peakerkennung hat.

## Aufgabe/Tab ‚Annotate‘: Sendersuche und Annotation

Wenn ein File geladen wurde, kann hier automatisch nach Senderpeaks gesucht und den Peaks eine Station zugewiesen werden. Die Bedienelemente dieses Tabs werden erst freigegeben, wenn ein gültiges wav-File geladen wurde und für dieses File die Annotation nicht bereits zu einem früheren Zeitpunkt abgeschlossen wurde. Abb. 4 zeigt das Bedienfenster mit einigen wichtigen Einstellungselementen.

Anzahl der Auswertezeitpunkte

Spektren während der Sendersuche anzeigen

Minimale für die Annotation gewünschte SNR

Anzeige Basislinienoffset

|  |
| --- |
|  |
| Abb. 4: Annotations-Tab mit wichtigen Voreinstellungs-Elementen |

Vor der Annotation sollte man noch die Werte der beiden Spinboxen ‚min SNR‘ und ‚#scan steps‘ kontrollieren und ggf. verstellen. ‚min SNR‘ hat dieselbe Funktion wie im Tab ‚Scanner‘, kann also hier wie dort eingestellt werden. ‚#scan steps‘ bestimmt die Anzahl der Zeitpunkte, an denen die Aufnahme bezüglich Senderpeaks ausgewertet werden soll, standardmäßig sind es 10 Punkte. Der Baseline-Offset wird hier nur mehr angezeigt. Falls man das nicht bereits vorher gemacht hat, sollte man im Tab ‚Scanner‘ noch kontrollieren, ob die Basislinie zufriedenstellend eingestellt ist (siehe Anmerkungen dazu im Abschnitt 13.2) Die Auswertung erfolgt so, dass Peaks, die mindestens einmal die Mindest-SNR erreichen, als gültig betrachtet werden. Die SNR für den Eintrag ins Annotations-yaml-File wird in der derzeitigen Version über alle Punkte gemittelt.

Nun sollte man auch noch festlegen, ob man während des Scans kurz Plots der Spektren zu den Scanzeitpunkten sehen möchte. Dazu kann man den Radiobutton ‚plot preview‘ rechts unten im Fenster aktivieren. Standardmäßig ist diese Funktion ausgeschaltet, um schnellere Scans zu ermöglichen.

Die Annotation selbst erfolgt in 2 größeren Schritten:

### Schritt 1: ‚Scan‘: Automatische Peak-Identifikation

Sobald man auf den Knopf ‚Scan‘ drückt, durchläuft das Programm die Aufnahme mit der Anzahl eingestellter Schritte und identifiziert die Peaks. Der Fortschritt wird durch den Statusbalken angezeigt. Danach wird die Senderliste (das mitgelieferte Excel-File) gelesen und anschließend werden für jeden Senderpeak die Kandidaten-Stationen in ein Hilfsfile gespeichert. (diese File findet man bei Interesse im Ordner root/Filename/stations\_list.yaml). Sobald dieser Vorgang beendet ist, wird wie in Abb. 5 ersichtlich in der zweiten Zeile des Tabs gelb unterlegt die Information ‚ autoscan has been completed, peaks and SNRs identified‘ angezeigt. Weiters wird der Knopf ‚Scan‘ grün und der Knopf ‚Annotate‘ wird freigegeben.

|  |
| --- |
|  |
| Abb. 5: Status nach Schreiben der Stationskandidatenliste. |

### Schritt 2: ‚Annotate‘: Zuweisung der korrekten Stationen

Wenn man auf ‚Annotate‘ klickt, wird die Liste der Kandidatenstationen zum nächsten noch nicht annotierten Peak angezeigt. Idealerweise sieht sie wie in Abb. 6 aus.

|  |
| --- |
| Liste der potenziellen Stationen zur angezeigten Frequenz, hier 153 kHz |
| Abb. 6: Fensteransicht bei Annotation |

Wenn zur angezeigten Frequenz aber kein Eintrag in der Senderliste gefunden wurde, erscheint nur eine Zeile mit ‚not identified | not identified | not identified‘, siehe Abb. 7. Eine Annotation muss hier, falls es sich doch um ein bekanntes Signal handelt, händisch im yaml-File nachgetragen werden.

|  |
| --- |
|  |
| Abb. 7: Anzeige einer nicht identifizierbaren Station |

Wenn eine der angezeigten Stationen korrekt ist, kann man auf sie klicken und sie damit auswählen. Der Name der ausgewählten Station wird nun in der Anzeigezeile neben der Frequenz angezeigt. Man wird von einem Popup-Fenster gefragt, ob man diese Station wirklich aufnehmen möchte (siehe Abb. 8), wenn man mit ‚yes‘ bestätigt, erfolgt ein entsprechender Eintrag in ein für COHIRADIA bereits korrekt formatiertes yaml-File mit Namen root/Filename/ cohiradia\_metadata.yaml.

Hat man irrtümlich auf eine falsche Station geklickt, muss man die Frage mit ‚no‘ beantworten und kann dann neu auswählen.

Möchte man eine Frequenz überhaupt nicht übernehmen, hat man 2 Möglichkeiten:

1. Klicken auf den Button ‚Discard‘ ohne auf eine der angebotenen Stationen zu kliken
2. Klicken auf ‚remove from record, nachdem man auf eine der Stationen geklickt hat

Danach springt das Programm auf den nächsten Peak weiter.

|  |
| --- |
| Korrekte Station per Mausklick auswählen, falls in der Liste angeboten  Frage, ob die Station zugewiesen werden soll (yes) oder nicht (no). Wenn man ‚remove from record‘ wählt wird die Frequenz übersprungen |
| Abb. 8: Auswahl einer Station aus der angebotenen Kandidatenliste. |

**Hinweis:** Leider ist derzeit keine Funktion eingebaut, die es erlaubt, in der bereits erzeugten Liste rückwärts zu springen. Sollte man eine Station übersprungen haben, muss man sie entweder später händisch ins yaml-File eintragen, oder man beendet das Programm, trägt im File root/Filename/status.yaml beim Eintrag ‚freqindex‘ einen entsprechend niedrigeren Wert (Minimum 0) ein und startet das Programm neu. Diese Option ist mit Vorsicht zu genießen, da dabei Doppeleinträge passieren können, die man auch händisch korrigieren muss.

Wenn man die ganze Frequenzliste fertig annotiert hat (Status wird am Statusbar angezeigt), wird in der Zeile neben der Frequenz gelb hervogehoben angezeigt ‚ Record has already been annotated. For re-annotation delete annotation folder ‘. Die Buttons ‚Scan‘,‘Annotate‘ und ‚Discard‘ werden

|  |
| --- |
|  |
| Abb. 9: Fensteransicht nach vollendeter Annotation |

Wenn man mit der Annotation nicht in einer Sitzung fertig wird, kann man das Programm jederzeit schließen und zu einer anderen Zeit fortsetzen. Das Programm merkt sich, an welcher Stelle man unterbrochen hat und setzt dort nach Neustart fort.

# Doku Methoden und Variablen

## 

## Generelle Strukturdiagramme: UMLET, Programm umlet-exe direkt aus Directory

D:\Documents\Installer\umlet-standalone-15.1\Umlet aufrufen

## Generalstruktur:

### Hauptmodule:

SDRCOHIWizard\_v#.py:

GUI-Klasse

…

Hauptschleife

System\_module.py:

Class status(): Zentrales Datensegment (Commonblock)

Class WIZ\_auxiliaries():

Standard\_errorbox

Standard\_infobox

Resampler\_module.py:

Class soxwriter(QObject): Worker Klasse für soxwriter\_thread

Class resampler(): Alles was mit resampling zu tun hat, reine State Machine

Def resample()

Def accomplish\_resampling()

Def LO\_shifter: auf threading umstellen: mach dies zu einem worker und starte einen thread der das abfährt

Geplant: scheduler

Hauptmodul:

Class playrec\_worker

Def play\_loop16

Def stop\_loop

Def resetCounter

Def read24

Class statlst\_gen\_worker(QtCore.QThread)

Def stationsloop NOCH self.host-Referenzen !

Class autoscan\_worker

Def autoscan\_fun: extract SNRdaea, write intermediate yamls auf self.host.annotation\_filename NOCH self.host-Referenzen !

Class timer\_worker

Class WizardGUI

### Zentrales Datensegment (Common Block)

## Class statlst\_gen\_worker(QtCore.QThread):

\_\_slots\_\_ = ["status\_position","T","freq","closed"]

Signale:

SigProgressBar1

SigFinished

Methoden:

set\_status\_position(self,\_value):

set\_T(self,\_value):

set\_freq(self,\_value):

set\_closed(self,\_value):

get\_status\_position(self): ##TODO: obsolete

stationsloop(self):

## Class class autoscan\_worker(QtCore.QThread):

\_\_slots\_\_ = ["slot\_0", "slot\_1"]

Signals:

SigUpdatestatus = pyqtSignal()

SigUpdateGUI = pyqtSignal()

SigScandeactivate = pyqtSignal()

SigFinished = pyqtSignal()

SigProgressBar = pyqtSignal()

SigStatustable = pyqtSignal()

SigPlotdata = pyqtSignal()

Methoden:

set\_0(self,\_value):

get\_0(self):

set\_1(self,\_value):

get\_1(self):

autoscan\_fun(self):

## Class class StemlabControl(QObject):

'''

Class for STEMLAB ssh connection, server start and stop,

data stream socket control and shutdown of the STEMLAB LINUX

'''

Signals:

None

Methoden:

set\_play(self):

set\_rec(self):

monitor(self):

config\_socket(self): ##TODO: atgument (self,modality)

'''

initialize stream socket for communication to sdr\_transceiver\_wide on

the STEMLAB

returns as errorflag 'False' if an error occurs, else it returns 'True'

In case of unsuccessful socket setup popup error messages are sent

Returns:

True if socket can be configures, False in case of error

requires self.modality to have been set by set\_play() or set\_rec()

'''

startssh(self):

'''

login to Host and start ssh session with STEMLAB

Returns False if a connection error occurs, returns True if

successful

'''

sshsendcommandseq(self, shcomm):

'''

send ssh command string sequence via command string list shcomm

'''

sdrserverstart(self):

'''

Purpose: start server sdr-transceiver-wide on the STEMLAB.

Stop potentially running server instance before so as to prevent

undefined communication

'''

def sdrserverstop(self):

'''

Purpose: stop server sdr-transceiver-wide on the STEMLAB.

'''

def RPShutdown(self):

'''

Purpose: Shutdown the LINUX running on the STEMLAB

Sequence: (1) stop server sdr-transceiver-wide on the STEMLAB.

(2) send 'halt' command via ssh, track result via stdout

(3) communicate steps and progress via popup messages

'''

## Class WizardGUI(QMainWindow)

**Variables: see also some in other fields marked in blue**

self.DATA\_FILEEXTENSION = ["dat","wav",'raw']

self.CURTIMEINCREMENT = 5

self.DATABLOCKSIZE = 1024\*32

self.DELTAF = 5000 #minimum peak distance in Hz for peak detector

self.PEAKWIDTH = 10 # minimum peak width in Hz for peak detector

self.PROMINENCE = 15 # minimum peak prominence in dB above baseline for peak detector

self.FILTERKERNEL = 2 # length of the moving median filter kernel in % of the spectral span

self.NUMSNAPS = 5 #number of segments evaluated for annotation

self.STICHTAG = datetime(2023,2,25,0,0,0)

self.autoscan\_ix = 0

self.timescaler = 0

#TODO: remove self.autoscan\_active = False

self.locs\_union = []

self.freq\_union = []

self.oldposition = 0

self.progressvalue = 0

self.cohiradia\_yamlheader\_filename = 'dummy'

self.cohiradia\_yamltailer\_filename = 'dummy'

self.cohiradia\_yamlfinal\_filename = 'dummy'

self.annotationdir\_prefix = 'ANN\_'

self.rates = {20000:0, 50000:1, 100000:2, 250000:3,

500000:4, 1250000:5, 2500000:6}

self.irate = 0

self.ifreq = 0

self.icorr = 0

self.curtime = 0

self.position = 0

self.ext #nicht initalisiert # Extension of filename

self.CURTIMEINCREMENT = 5

self.ui = MyWizard()

self.ui.setupUi(self)

## TODO: wavheader-Writing zum Button Insert Header connecten

**Im Programm ohne Initialisierung erzeugte selfs:**

Self.ext

Self.f1

Self.myfilename

Self.mydirname

**status flags: see also some in other fields marked in blue**

# initialize status flags

self.TEST = True # Test Mode Flag for testing the App ##NOT USED

self.fileopened = False

self.scanplotcreated = False

self.ui.pushButton\_Play.setIcon(QIcon("play\_v4.PNG"))

self.ui.pushButton\_Play.clicked.connect(self.togglePlay)

self.ui.pushButton\_Stop.clicked.connect(self.GuiClickStop)

self.open\_template\_flag = False

self.ovwrt\_flag = False

self.autoscanthreadActive = False

self.flag\_ann\_completed = False

self.lock\_playthreadstart = True

self.playthreadActive = False

self.timechanged=False

self.fileopened = False

self.ismetadata = False

# connect menubar events

**File menu connects**

self.ui.actionFile\_open.triggered.connect(self.open\_file)

self.ui.actionSave\_header\_to\_template.triggered.connect(self.save\_header\_template)

self.ui.actionLoad\_template\_header.triggered.connect(self.open\_template\_file)

self.ui.actionOverwrite\_header.triggered.connect(self.overwrite\_header)

self.ui.pushButton\_FastForward.clicked.connect(

lambda: self.updatecurtime(self.CURTIMEINCREMENT))

self.ui.pushButton\_rew.clicked.connect(

lambda: self.updatecurtime(-self.CURTIMEINCREMENT))

self.ui.ScrollBar\_playtime.sliderReleased.connect(self.jump\_to\_position)

**Buttons**

self.ui.pushButton\_Scan.setEnabled(False)

self.ui.pushButtonAnnotate.setEnabled(False)

self.ui.pushButton\_Scan.clicked.connect(self.autoscan)

self.ui.pushButtonAnnotate.clicked.connect(self.ann\_stations)

self.ui.pushButtonDiscard.clicked.connect(self.discard\_annot\_line)

self.ui.pushButtonDiscard.setEnabled(False)

**Spin Boxes**

self.ui.spinBoxKernelwidth.valueChanged.connect(self.setkernelwidth)

self.ui.spinBoxKernelwidth.setEnabled(False)

self.ui.spinBoxKernelwidth.setProperty("value", 15)

self.ui.spinBoxminSNR.valueChanged.connect(self.minSNRupdate)

self.ui.spinBoxminPeakwidth.valueChanged.connect(self.minPeakwidthupdate)

self.ui.spinBoxminPeakDistance.valueChanged.connect(self.minPeakDistanceupdate)

self.ui.spinBoxminSNR\_ScannerTab.valueChanged.connect(self.minSNRupdate\_ScannerTab)

self.Baselineoffset = self.ui.spinBoxminBaselineoffset.value()

self.ui.spinBoxminBaselineoffset.valueChanged.connect(self.set\_baselineoffset)

self.ui.tabWidget.setCurrentIndex(2)

self.ui.pushButton\_InsertHeader.setEnabled(False)

self.ui.pushButton\_InsertHeader.clicked.connect(self.overwrite\_header)

self.ui.label\_8.setEnabled(False)

self.ui.lineEdit.setAlignment(QtCore.Qt.AlignLeft)

#self.ui.lineEdit.returnPressed.connect(self.enterlinetoannotation)

self.ui.pushButtonENTER.clicked.connect(self.enterlinetoannotation)

self.ui.pushButtonENTER.setEnabled(False)

# initialize some GUI elements

self.ui.radioButton\_WAVEDIT.setEnabled(True)

self.ui.radioButton\_WAVEDIT.setChecked(False)

self.ui.radioButton\_WAVEDIT.clicked.connect(self.activate\_WAVEDIT)

#self.ui.pushButton\_ScanAnn.clicked.connect(self.listclick\_test)

self.ui.Annotate\_listWidget.itemClicked.connect(self.ListClicked)

self.ui.Annotate\_listWidget.clear()

self.ui.tableWidget.setEnabled(False)

self.ui.tableWidget\_3.setEnabled(False)

self.ui.tableWidget\_starttime.setEnabled(False)

#self.ui.horizontalScrollBar.sliderMoved.connect(self.plot\_spectrum\_evth)

#self.ui.horizontalScrollBar.valueChanged.connect(self.plot\_spectrum\_evth)

self.ui.progressBar\_2.setProperty("value", 0)

#self.ui.progressBar.setEnabled(True)

self.ui.horizontalScrollBar.sliderReleased.connect(self.plot\_spectrum\_evth)

self.ui.radioButton\_plotraw.clicked.connect(self.plot\_spectrum\_evth)

#self.ui.horizontalScrollBar.actionTriggered.connect(self.plot\_spectrum\_evth)

self.SigToolbar.connect(lambda: self.plot\_spectrum(self,self.position))

self.ui.spinBoxNumScan.setProperty("value", 10)

self.ui.spinBoxminBaselineoffset.setProperty("value", 5)

# elements of yaml header generator tab

self.ui.pushButton\_Writeyamlheader.setEnabled(False) # activate after completion of the annotation procedure

self.ui.pushButton\_Writeyamlheader.clicked.connect(self.yaml\_header\_buttonfcn)

self.ui.lineEdit\_IPAddress.returnPressed.connect(self.set\_IP)

self.ui.lineEdit\_IPAddress.setInputMask('000.000.000.000')

self.ui.lineEdit\_IPAddress.setText("000.000.000.000")

self.ui.lineEdit\_IPAddress.setEnabled(False)

self.ui.lineEdit\_IPAddress.setReadOnly(True)

#pushButton->setIcon(QIcon(":/on.png"));

self.ui.pushButton\_Configuration.clicked.connect(self.editHostAddress)

self.ui.lineEdit\_IPAddress.returnPressed.connect(self.set\_IP)

#read config file if it exists

self.standardpath = os.getcwd()

self.metadata = {"last\_path": self.standardpath}

try:

stream = open("config\_wizard.yaml", "r")

self.metadata = yaml.safe\_load(stream)

stream.close()

self.ismetadata = True

if 'STM\_IP\_address' in self.metadata.keys():

self.ui.lineEdit\_IPAddress.setText(self.metadata["STM\_IP\_address"])

except:

#return False

print("cannot get metadata")

self.HostAddress = self.ui.lineEdit\_IPAddress.text()

# init figure plot in Grid of Tab ‘View Spectrum’

self.figure = Figure()

self.canvas = FigureCanvasQTAgg(self.figure)

self.ui.gridLayout\_4.addWidget(self.canvas,4,0,1,5)

self.toolbar = NavigationToolbar(self.canvas, self)

self.ui.gridLayout\_4.addWidget(self.toolbar,2,1,2,1)

self.ax = self.figure.add\_subplot(111)

self.ax.plot([], [])

self.canvas.draw()

self.timeref = datetime.now()

self.autoscanthread = QThread()

self.autoscaninst = autoscan\_worker(self)

self.autoscaninst.moveToThread(self.autoscanthread)

#TODO: Implement further slot communicationa shown here

self.autoscaninst.set\_0([self.ui.spinBoxNumScan.value(),self.ui.spinBoxminSNR.value(),[]])

# start timer tick

self.timethread = QThread()

self.timertick = timer\_worker()

self.timertick.moveToThread(self.timethread)

self.timethread.started.connect(self.timertick.tick)

self.timertick.SigFinished.connect(self.timethread.quit)

self.timertick.SigFinished.connect(self.timertick.deleteLater)

self.timethread.finished.connect(self.timethread.deleteLater)

self.timertick.SigTick.connect(self.updatetimer)

self.timethread.start()

if self.timethread.isRunning():

self.timethreaddActive = True

**Methoden:**

def updatetimer(self):

"""updates timer functions

shows date and time

changes between UTC and local time

manages recording timer

:param [none]: none

:type [none]: none

...

:raises [ErrorType]: none

...

:return: none

:rtype: none

"""

Interactions:

Polls:

ui.checkBox\_UTC

ui.checkBox\_TESTMODE

ui.radioButton\_timeract

ui.radioButton\_Timer.isChecked():

Acts on:

ui.label\_showdate.setText

ui.label\_showtime.setText

ui.radioButton\_Timer.setChecked

ui.dateTimeEdit\_setclock.setEnabled

ui.checkBox\_UTC.setEnabled)

ui.dateTimeEdit\_setclock.dateTime().toPyDateTime()

ui.checkBox\_UTC.setEnabled

def checkSTEMLABrates(self):

"""\_checks if self.ifreq, self.irate, self.icorr have the proper values acc. to RFCorder filename convention

checks if self.irate has a value out of the values defined for the STEMLAB in self.rates

:param [none]: none

:type [none]: none

...

:raises [popup error message]: corresponding to different format errors

...

:return: True/False acc to success of the check

:rtype: bool

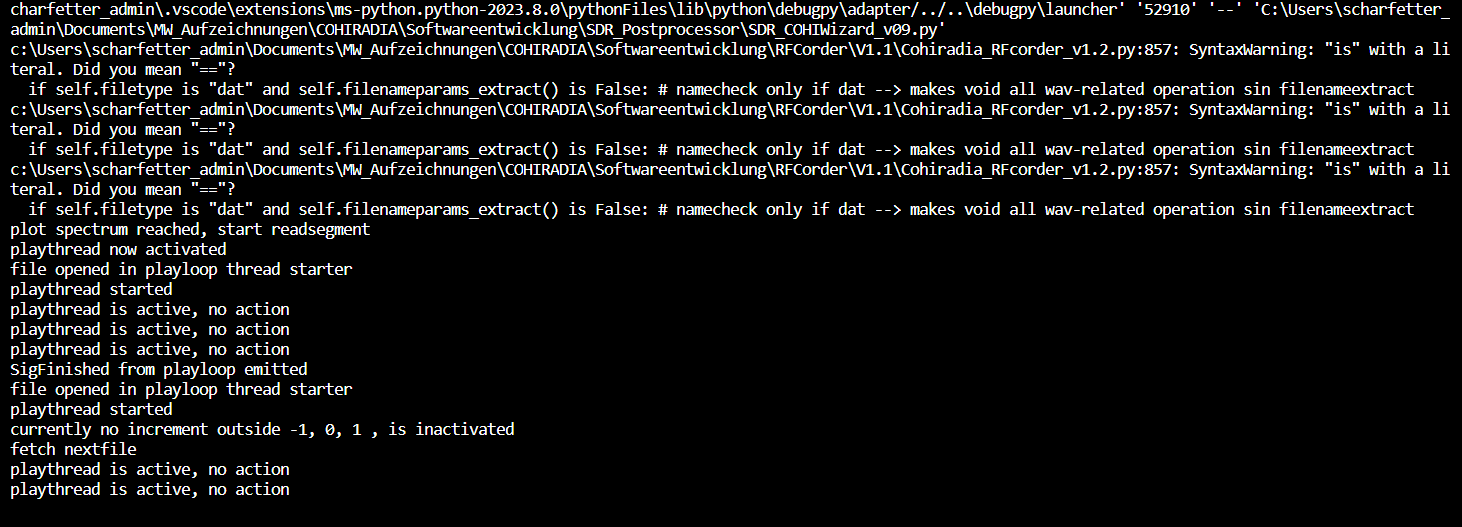
"""

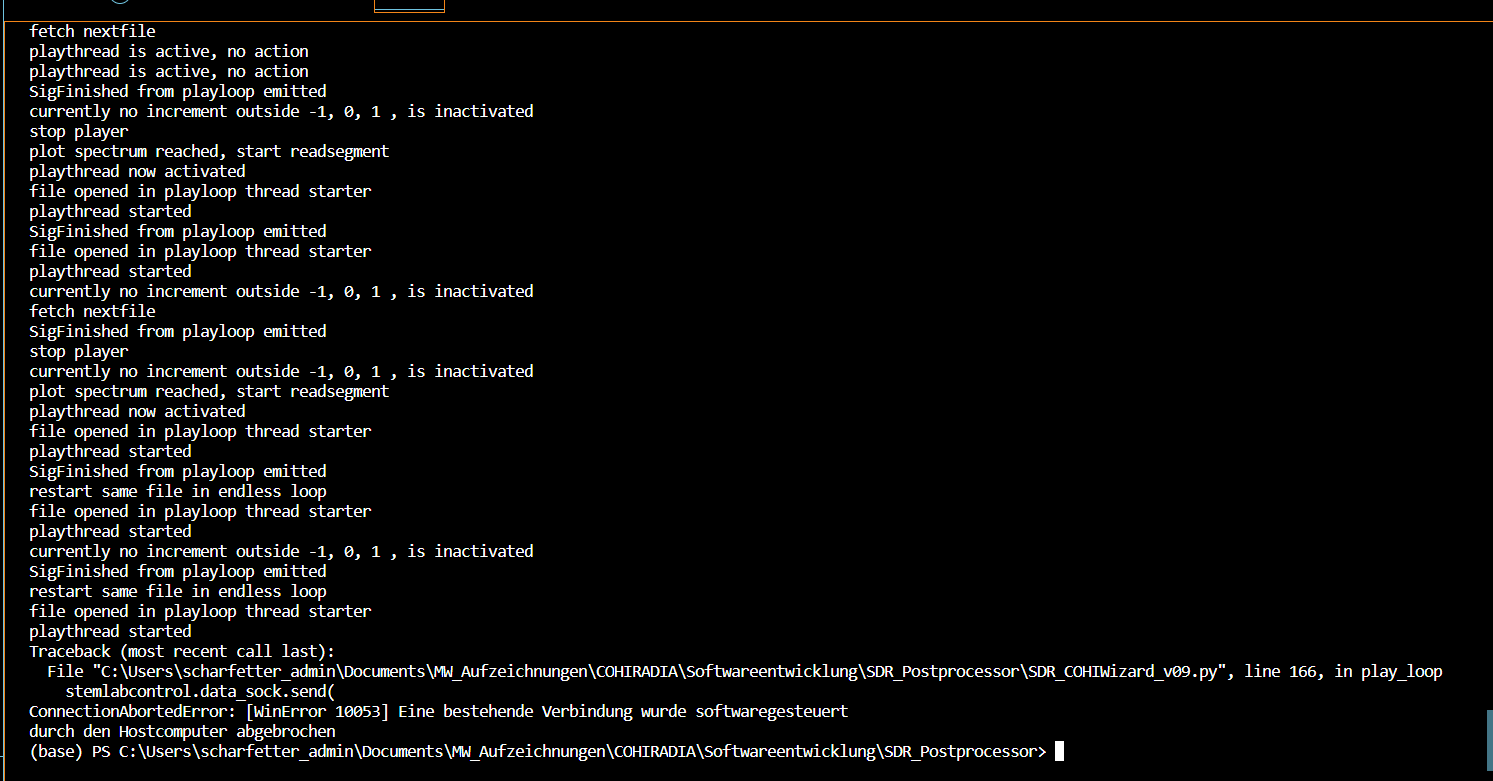
Polls: None

Acts on: None

def autoscan(self):

configures and starts the autoscan thread





# Generelle Strukturregeln:

**Threads:**

* Thread X worker methods , usually parts of a Thread X class
* Thread X starter method
* Thread X terminator method

Each thread class has \_\_slots\_\_ for communication, communication entirely via set and get functions

Nomenclature: set\_[variablename](value), value = get\_[variablename]

Threadworker.set\_[variablename](value)

*Namenskonvention:*

[threadname]\_tworker

[threadname]\_tstarter

[threadname]\_terminator

**Window-Elements:**

Class for window; maybe even for subwindows (Tabs ) ??

Callback functions for each element: call sequence of related operations

cb\_[abbr\_windowelementobjectname]: abgekürztes Windowelement im Namen:

Button 🡪 Butt

Spinbox 🡪 SpBx

Sliderbar 🡪 SlBr

Status bar 🡪 StatBr

Label 🡪 Label

LinEdit 🡪 Ledit

make all window elements with standard names:

Buttons: pushButton\_[purpose]

**Methoden für die Aufgabenverteilung nach Callbacks:**

[Aufgabe]\_manager

**Methoden für GUI\_Resets:**

**Complex operations aside window element manipulations:**

Separate classes (e.g. wavheaderfuncions)

Put command sequences with the same goal in a single method and call it from outside

Communication via self: reduce to necessary, keep most variables encapsulated in methods

GUI\_Designer:

Erzeugt alle windowelementobjectname mit Standardnomenklatur

[GUIDesigner\_Standardname ohne Nummer]\_[klingende Funktionsbezeichnung]\_[optional: Nummer]

**Farbcodes:**

Method

Signal

Thread starter

Thread worker

GUI Callback

GUI Main self variables

Normal functions

GUI Buttons

GUI Display Elements

Amphibic GUI Elements

File ext

## Versuch einer Strukturierung:

Resampler Elemente und Doku

21-11-2023: Versuch einer Mitdokumentation **mit Scheduler\_Struktur\_Resampling.pptx**

Stand: Bis version 21: letzte funktionsfähige Version. Plan: Scheduler voll implementieren

Bisher wird vom Scheduler nur ‚resample‘ ausgelöst

**Grundidee**

Signal SigIncrScheduler löst Aufruf von **resamp.res\_scheduler** aus.

In einem Dictionary **schedule[i]** ist je Index abgelegt, was als nächstes passieren soll und welche Parameter für den nächsten Schritt gesetzt werden sollen. Diese sind:

action: nächste Methode; blinkstate: wie das Status-Label gefärbt sein soll; actionlabel: Text für die nächste Aktion; sSR und tSR: Samplingraten für source und target; sBPS, wFormatTypes, wFormatTag, filesize

Es gibt weiters ein Doppel-Dictionary, in dem Signale und die sie abonnierenden Methoden definiert werden. Die ist eine Metastruktur, in der man ganz zentral umstellen kann, wenn eine Funktion der resampler-Pipeline ausgetauscht werden soll (auch gut für Testen)

schedule\_objdict = {}

schedule\_objdict["signal"] = {}

schedule\_objdict["signal"]["resample"] = self.SigResample

schedule\_objdict["signal"]["accomplish"] = self.SigAccomplish

schedule\_objdict["signal"]["LOshift"] = self.SigLOshift

schedule\_objdict["signal"]["updateGUI"] = self.SigUpdateGUI

schedule\_objdict["connect"] = {}

schedule\_objdict["connect"]["resample"] = resamp.resample

schedule\_objdict["connect"]["accomplish"] = resamp.accomplish\_resampling

#TODO: Umstellen auf schedule\_objdict["connect"]["resample"] = resamp.LOshifter\_new

schedule\_objdict["connect"]["updateGUI"] = resamp.res\_update\_GUI

Aktuell setzt die resampler Methode Parameter von oben plus die Filenamen für den nächsten Schitt

Im neuen Schema soll das der Scheduler tun, indem er die entsprechenden Einträge aus schedule[i] entnimmt. Realisiert vor 8-12-2023

1. Vorspann: schaue ob sox existiert, File open check
2. Abonniere Signale, hier neu definiert, check ob obsolet
3. Definiere schedule\_objdict
4. Set wav-header options, format options, error-checking etc.
5. Define target filename (Z1186)
6. Define system\_states for the next step (LOSHIFTER) which keep resampling parameters; Question if they should be defined in scheduler ?
7. Setze explizit GUI-Elemente um (🡪 scheduler)
8. Starte LOSHIFTER über Signal self.SigLOShift.emit()
9. Write wavheader to output file
10. Inactivate all other tabs
11. Define scheduler table:

|  |
| --- |
|  |
| Scheduler-Ablaufstruktur. Der scheduler muss nach Definition der schedule und des Metaobjekts Objdict einmal durch SigIncrScheduler gestartet werden  Grafik in: [..\Dokumentation\Scheduler\_Struktur\_Resampling.pptx](file:///C:\Users\scharfetter_admin\Documents\MW_Aufzeichnungen\COHIRADIA\Softwareentwicklung\Dokumentation\Scheduler_Struktur_Resampling.pptx) |

**Cb\_resampler:**

In der Metastruktur Objdict werden die Signale und die sie abonnierenden Methoden festgelegt (Referenztabelle). Man kann dann ggf. leicht andere Methoden verwenden, indem man sie lediglich in dieses dictionary einträgt (Abonnierungstabelle). Dann werden die anderen Tabs durch inactivate\_tabs gesperrt.

Der Scheduler muss nach Definition der schedule und des Metaobjekts Objdict einmal von cb\_resampler() durch SigIncrScheduler.emit() gestartet werden. Danach verabschiedet sich cb\_resampler.

**Scheduler-Loop:**

Dann holt er sich aus der schedule die jeweils nächsten Aktionen, i.e. löst die Actionfunctions durch die von ihnen abonnierten Signale aus, nachdem die Systemstates gemäß der schedule updatet worden sind.

Jede Actionfunction liest den für sie relevanten aktuellen Systemzustand, konfiguriert mindestens einen Worker-thread, ggf auch einen progress-thread für den progressbar und Startet dann den workerthread. Das SigFinished des workerthreads wird abonniert (connect) von (lambda: SigIncrScheduler.emit()) , so wird gewährleistet, dass dieses Signal sicher bei Beenden des workerthreads ausgelöst wird. Dann beendet sich die Actionfunction.

Der Workerthread tut irgend etwas, meist in einer loop. In der Loop kann er den progress updaten (setzt seinen eigenen progress slot mit der setterfunction und emittiert das progress-Signal SigPupdate, dass in seiner Actionfunction zugeordnet worden ist). Am Ende muss er den Scheduler weitertreiben . Die Schedule wird so Schritt für Schritt abgearbeitet, bis ddas ‚terminate‘ erscheint, wonach sich der Scheduler kontrolliert verabschiedet und die gesperrten Tabs wieder freigibt.

Progress-monitor ist ein GUI-Element

**WICHTIG:**

Die View Control-Struktur sieht eine völlige Trennung GUI und controller vor. Klassich sollte jede Aktualisierung des system\_states ein völliges Update des GUI auslösen. Das ist aber zu aufwändig, es sollten oft nur segmente eines tabs aktualisiert werden (z.B. progressbar)

Vorchlag:

1. In system\_state gibt es ein Feld system\_state[„update“] (True, False)

und

system\_state[„tab\_label“] mit item

updatedict[„updatesegment“] [„updatesignal“]

updatedict[„updatesegment“] [„updatemethod“]: hier wird einfach die Referenz auf diese Methode (Methodennamen) eingetragen. Beispiel:

Aktuell gibt es bereits ein Beispiel im resampler\_Modul:

system\_state["sox\_worker"] = self.sox\_worker

könnte man modifizieren zu:

system\_state[„Tab\_Resample“][„progressbar“][“updatemethod”] = … self.updateprogress\_resampling() **self** ist hier **gui**

system\_state[„Tab\_Resample“][„progressbar“][“updatesignal”] = resampler.sox\_worker.SigPupdate

Auf diese Referenz wird im gui.updateprogress\_resampling zurückgegriffen:

progress = system\_state["sox\_worker"].get\_progress()

irgendwo muss stehen:

self.sox\_worker.SigPupdate.connect(gui.updateprogress\_resampling)

1. Zu jedem solchen Eintrag muss es eine methode im GUI-Segment geben, die vom Signal ausgelöst wird (connect muss in der set\_systemstate-Methode gemacht werden). Diese Methode liest dann die zu setzenden **Parameter** aus System\_state aus und updated die zugehörigen GUI\_elemente.
2. Die Signale und Methodennamen müssen in einer entspr Table im Tab-Segment des GUI definiert werden
3. Set\_system muss enthalten:

If system\_state[„update“]

system\_state[„update“] = False

system\_state[„tab\_label“][„updatesegment“] [„updatemethod“].connect …

system\_state[„tab\_label“][„updatesegment“] [„updatesignal“]

system\_state[„tab\_label“][„updatesegment“] [„updatesignal“].emit()

1. Jede action workerfunction mit loop errechnet einen progress, label, labelproperties für den zugeordneten tab. Diese **Parameter** (s.o.) werden über die setters auf entsprechende slots geschrieben. Dann wird ein Signal SigSystemUpdate ausgelöst.
2. Dieses Signal löst eine system\_update Methode aus, die folgendes macht:
3. system\_update mcht:
   1. holt die **Parameter** mit getter von workerfunction und schreibt sie auf auf system\_state geschrieben. System\_state[Tab\_label] [„updatesegment“] wird spezifiziert
   2. system\_state[„update“] wird auf True gesetzt
   3. Danach wird und ein set\_system\_state ausgelöst

Aktuell bei **sox-resampler:**

Generiert soxstring und ein paar Parameter

Konfiguriert soxworker

Signal SigFinished: wird von soxthread quit etc. abonniert sowie: gui\_SigIncrScheduler.emit().

Ein Signal sox\_worker.SigPupdate wird von updateprogress\_resampling abonniert

Startet sox\_worker

Aktuell bei **Sox\_worker:**

Definiert 2 Signale: SigFinished und SigPupdate

Arbeitet Workerloop ab

Macht progress update und setzt progress auf den entprechenden slot via progress\_setter

Emittiert SigPupdate

Emittiert SigFinished

## Änderungsdokumentation:

### Gauertendlosabspieler, ChatGPT:code:

COHIRADIA-Server: unter ‚software/‘ als ‚COHIRADIA\_playback\_filelist\_endless.zip‘ abgelegt

Die Fileliste wird in einem Textfile COHI\_Playlist.txt abgelegt (zeilenweise Filepaths)

Der name dieser Liste wird in config\_sdr.yaml eingetragen:

fileliste: 'C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\COHIRADIA\RM\_Serverspiegel\Software\_vor01042022\COHI\_Playlist.txt'

**WICHTIG:**

1. Alle Files müssen der alten COHIRADIA-Namenskonvention genügen (\_lo####\_r####\_c0 muss enthalten sein), allerdings ist die Extension egal (wav oder dat ist gleichermaßen gültig).

Einbau des Listen-Abarbeitens in die loop: self.file\_list existiert nun in der Klasse COHplayer\_endless als eine Liste von Filenamen, die abzuarbeiten ist

**TODO :**

**Bereinigen: wird openfile(self) benötigt ?**

**Bisher gemacht:**

Einlesen der Playlist im main des Programms

cp1.file\_list

**Grundlage:**



bitte schreib mir Python coder der von einer Liste von Filenamen der Reihe nach die files-Inhalte einliest und in eine Schleife auf numpy Puffer-Arrays der Länge int16 schreibt . In derselben schleife wird aus dem Pufferarray auf ein Ausgabefile geschrieben

o.k., danke, bitte lies mir noch die file\_list von einem ASCII-File ein, in dem die Filenamen zeilenweise drinstehen

ChatGPT

Natürlich, Sie können den Dateinamen von einer ASCII-Datei lesen, in der die Filenamen zeilenweise aufgeführt sind. Hier ist eine aktualisierte Version des Skripts, das die Dateinamen aus einer solchen Datei liest:

python

import numpy as np

def read\_file\_list(file\_list\_path):

# Lese die Filenamen zeilenweise aus der ASCII-Datei

with open(file\_list\_path, 'r') as file:

file\_list = [line.strip() for line in file]

return file\_list

def process\_files(input\_file\_list\_path, output\_file):

# Lese die Liste der Dateinamen aus der ASCII-Datei

file\_list = read\_file\_list(input\_file\_list\_path)

­­ for input\_file in file\_list:

# Einlesen des Dateiinhalts in ein NumPy-Array (int16)

data = np.fromfile(input\_file, dtype=np.int16)

# Hier können Sie weitere Verarbeitungsschritte auf dem Datenarray durchführen, wenn nötig

# Schreiben des Inhalts des NumPy-Arrays in ein Ausgabefile

with open(output\_file, 'ab') as output\_file:

data.tofile(output\_file)

if \_\_name\_\_ == "\_\_main\_\_":

# Beispiel: Pfad zur ASCII-Datei mit den Filenamen und Ausgabefile

input\_file\_list\_path = "file\_list.txt"

output\_file = "output.dat"

# Funktion aufrufen

process\_files(input\_file\_list\_path, output\_file)

### COHIWizard:

letzte funkt. Version = 25 mit bereits gepurgter stemlabcontrol-Kommunikation und Signaltrennung GUI – resampler

Aktuell v26: Ersatz self.f1 durch generelle system\_state[„f1“]

Start-Stoptime Implementierung:

1. Definiere neue Variablen ‚system\_state[„starttime\_cut\_secs”] , system\_state[„stoptime\_cut\_secs”] system\_state[“starttrim”] und system\_state[“stoptrim”]
2. Initialisiere die Zeitfelder für Start und Stopcut-Time im GUI auf die Startzeit des ersten wav-Files in der Resample-Liste und die Stopzeit des letzten Files. Entry point:

Aktualisierung der res\_playlist in

v\_resamp.reslist\_update

There is already a TODO:

        #TODO: fetch starttime of the first file and stoptime of the last file to copy the values to the starttime\_cut and stoptime\_cut windows of the GUI

1. Lies system\_state[„starttime\_cut\_secs”] und system\_state[„stoptime\_cut\_secs”] aus der Differenz der Start- und Stopcut-Times und der wavheader-Start/stopzeiten ein: wandle diese Zeiten in Sekunden um und schreibe sie auf system\_state[„starttime\_cut\_secs”] , system\_state[„stoptime\_cut\_secs”]. Falls die Differenzen > 0 sind, setze system\_state[“starttrim”] bzw system\_state[“stoptrim”] auf True

Entry point für diese Sequenz ist in cb\_resample bereits markiert

                #TODO: ENTRY for cutting

                #if first listentry and cuttime start is active: set system\_state["starttrim"] True and write starttime cut to system\_state["starttime\_cut\_secs"]

                #if last listentry:set system\_state["stoptrim"] True and write stoptime cut to system\_state["stoptime\_cut\_secs"]

1. Verwende sox extension trim in resampler.resample(self); Code ist bei soxstring Generator bereits auskommentiert vorhanden

Beim Stop Player sollen die Play und res-Listen zurückgesetzt wowie die wav-header-table gelöscht werden

Auch nach Abarbeitung der resampling-Prozedur sollten die resampler-Listen zurückgesetzt werden (clear)

Beim Abspielen einer Playlist wird von Nummer zu Nummer die Gain zurückgesetzt. Das soll nicht sein !

Allgemeiner Playlist-Updater nach diversen Aktionen (wav-Header Einfügung, Resampling, hole letztes gültiges File…)

Baseline offset, SNR, Peakwidth, Peakdistance MF Kernel sollten von der letzten Konfiguration im Statusfile gelesen werden. Nicht essentiell, da nach Abschluss des Autoscans diese Werte nicht mehr benutzt werden, aber es irritiert den User, wenn Werte nicht auf zuletzt benutzten Werten.

#TODO: reade info on spinbox settings MF kernel etc from status file if it

exists

resampler: Bei 500🡪1250 resampling und LOSHIFT einer Fileserie wird immer ein 24/32-Resampling vorausgeschickt ? Das passiert nicht wenn kein LOSHIFTer Hier wird eine Bedingung falsch abgefragt. Grund: Vermeidung von Aliases damit sicher.

**scheduler:**

**schedule Bedingung 3:**

Einfaches resampling mit Speedchange z.B. 3%)

16-bit oder 32-bit Sourcefile,

wFormatTag: PCM oder IEEE:

speedfactor = 1.0x, x = percent speed increase

Standardpipeline:

**Loshift (um LO (upsampling vom Basisband auf LO)!, SBPS, TBPS) 🡪 temp0 in TBPS 🡪 resample(TBPS 🡪 TBPS, SSR 🡪 SSR\*speedfactor) 🡪 temp1 🡪 LOshift(-LO + centershift, TBPS, TBPS) 🡪 temp2 🡪 resample(TBPS 🡪 TBPS, SSR 🡪 TSR) 🡪 temp3 🡪 accomplish**

* + Gui: self.position muss dringend ersetzt und auf einen system state umgebaut werden
  + Neuer Bug : Annotator stürzt bei gewissem Zeitpunkt beim Annotieren ab mit

File "C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\COHIRADIA\Softwareentwicklung\SDR\_Postprocessor\SDR\_COHIWizard\_v26.py", line 3144, in ann\_stations self.interactive\_station\_select()

File "C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\COHIRADIA\Softwareentwicklung\SDR\_Postprocessor\SDR\_COHIWizard\_v26.py", line 3188, in interactive\_station\_select

country\_string = self.stations[freq\_ix]['country' + str(ix2)]

KeyError: 'country0'

Da stand ein Blödsinn ‚country 6650 ‘ im stations\_list.yaml. Muss beim Eintragen bei schlechtem Handling möglich sein ??

File:

Frequenz: 978, nach ‘DISCARD’; Dieser Eintrag wird offenbar nach dem Lesen der MW\_Liste im Zugdes Schreibens von ‚stationslist.yaml‘ im stationsloop\_worker generiert. Scheint zu passieren, wenn der Index des Eintrags ‘OTHER COUNTRY’ gleich 0 ist, also noch nichts anderes in der Liste steht. Evt liegt das an der Zeile ?:

for ix2 in range(len(sortedtable))

                    dum\_cstr = 'OTHER COUNTRY, Please enter manually'

                    dum\_pstr = 'OTHER STATION, Please enter manually'

                    dum\_txstr = 'OTHER TX SITE, Please enter manually'

                    ix2 = ix2 + 1

                    country\_string = '  country' +str(ix2) + ': "{}"\n'.format(dum\_cstr)

                    programme\_string = '  programme' +str(ix2) + ': "{}"\n'.format(dum\_pstr)

                    tx\_site\_string = '  tx-site' +str(ix2) + ': "{}"\n'.format(dum\_txstr)

* Test 32bit Sourcefile-Behandlung
* Nach reasmple wird zwar das f1 File umgestellt, aber die Selectlist, Playlist nicht aktualisiert
* Einbau von GUI Version 10 mit Listenfeldern und %Speedchange-Option
* Signale des Reampler-Moduls von GUI auf Resampler legen
* Bereinigen der Abspieldauer-Berechnung
* **30min** In einigen Instanzen wird filesize aus dem wav-Header gelesen, aber besser wäre ein os.stat Befehl für genause Ermitteln der Filegröße. Manchmal sind ja die Header fehlerhaft bzw die Files größer als 2G.

Lösung: im fileopen die filesize mit os.stats auslesen und auf system\_state[„filesize“] schreiben. Danach mit allen Funktionen, ausser wav-header-editor nur mehr darauf zugreifen.

* Aktivierung von Listenabarbeitungen bei Resampler:
  + das mit File open gewählte File wird nicht in der resampler-Auswählliste angezeigt !
  + Lösche intermediäre ‚resamp‘ Files die der resampler erzeugt.
  + Schreib einen Checker, ob alle Resample Files in der Liste dieselben Settings haben, sonst Errormessage
  + Definiere Reset-Methode für tab player, die auch die listwidgets cleared
  + Problem: f1 darf nicht mit dem targetfilenamen überschrieben werden, da nunmehr ein /out Directory produziert wurde; Dadurch wird aber dann zuletzt nicht mehr das zuletzt produzierte File ins GUI geladen, sondern das zuletzt benutzte roh-resampelte File
  + Allgem Problem: Filenamen > 260 Zeichen können öfters mal vorkommen und gehen dann nicht. Muss man allgemeiner abfangen
  + Eventgesteuert: cb\_resampler hat ein Listensteuer-Segment, das den jeweils nächsten Namen in der Liste abfragt und den system\_state entspr konfiguriert. Dann wird ganz normal die scheduler-Loop abgefahren. Neu. Das Terminate-Segment im scheduler setzt ein Signal SigTerminate\_Finished ab, das wiederum vom cb\_resamler abonniert ist, daher wird der cb-resampler erneut gestartet, sobald ein File fertig resampled ist bis die Liste zu Ende ist. Jedes neu erzeugte resamplede File kommt i eine Liste system\_state[„list\_out\_files\_resampled“]
  + Problem: Resampler-Plot soll vom ersten Listeneintrag aus erstellt werden, nicht gelesenes File
  + Benenne um system\_flags 🡪 system\_state
  + Beim letzten Terminate Signal wird der 2GB\_segmenter gestartet
* Aktivierung von Listenabarbeitungen bei Player:
  + Entry = EOF\_manager(self). Hier ein elif dazu, das die playlist abarbeitet
  + Vorher: Zu beginn muss das gewählte file als erstes in die playlist kopiert werden
  + Komplette Überarbeitung der Stemlabcontrol-Klasse:
    - Win-Referenzen ersetzen durch normale Übergabeparameter
    - LO\_offset wird nun schon im GUI in Irate eingerechnet und nur mehr diese übergeben
* ize config\_socket via slots
  + - TODO:
* #TODO: system\_state Eintragung von icorr etc erfolgt an mehreren STellen, fileopen, extract dat header und hier. Könnte an einer Stelle passieren, immer, sobald ein wav-header extrahiert wird
* + - Bei Abarbeiten einer Playlist muß jedesmal bei Wechsel von irate, LOfreq das STEMLAB rekonfiguriert werden: stop playloop, reconfig, restart playloop : Ausprobieren ob sdrconfig on the fly geht. Man könnte da gleich den play\_manager im eof\_manager aufrufen. Vorher sollte man aber noch einen sdrserverstop durchführen oder ein Flag kreieren, da bei gestartetem sdrserver vor allen Aufrufen von sdrserver\_start() abgefragt wird und bei sdrserver\_stop rückgesetzt wird. Wäre wahrscheinlich effizienter, aber unübersichtlicher.
    - Check Beobachtung Nach Header-yaml editierung/Neuschreiben wurde bei Programmneustart nach load file gefragt, om an das yaml-headerfile überschreiben will. CJECK
    - In der selectlist/playlist wird ein leeres element zu beginn erzeugt, das macht eher probleme beim Schieben.
* Klären, ob Playlist-Politik aktuell sinnvoll ist: Idee: Inakiviere selectlist der playlist Mache Playlist-edit\_feld mit ordentlichem Icon toggle-bar. Wenn akiviert: aktiviere beide Listen und erlaube Ziehen von Elementen, bei Deaktivierung: Lösche Playlist und deaktiviere beide Listen

Aktuell: Wenn ein File abgespielt wurde :

Next file in nextfile ? 🡪 play nextfile

Else: if endlessmode: Wiederhole aktuelles File endless

Else: hole nächstes Playlist-Element

Problem: Wenn nächstes Playlist element ein nextfile hat, wird die nextfile-Kette eingeschoben. Will man das ?

* Abfangen, wenn falsches wav-file in der Liste
* Alle Parameter an die Klassenmethoden der stemlabcontrol-Klasse werden als Parameter system\_state[„sdr\_configparams“] übergeben: 14-12-2023

GELERNT: Verwendung von Signals mit Parametern: Die Klasse Stemlabcontrol hat 2 Signale: SigError(str) und SigMessage(str). Beide werden mit Argument emittiert, z.B.

self.SigError.emit("Cannot establish socket connection for streaming to the STEMLAB")

Damit ist der Klasse egal, was mit der Message passiert und sie muss auf keine weiteren Handler zugreifen.

Im GUI-Methode play-Managerin warden dann Handler auf diese Signale connected:

        stemlabcontrol.SigError.connect(self.stemlabcontrol\_errorhandler) #

        stemlabcontrol.SigMessage.connect(self.display\_status)

Die Handler übernehmen dann die Anzeigen und weitere Schritte

Resampler-Modul auf v4 umstellen

v24: BASIS f kompilierte Version für die Webpage 🡪 Kompilieren

funktionierende Version mit voller Scheduler Funktionalität;

Resampling bereits für 24 bit SourceBPS implementiert

Resampling bei SSR < TSR und LOshift sowie bei 24-Bit-resampling beinhaltet einen resampling Schritt mehr:

*Sourcefile 🡪 resample(sBPS 🡪 tBPS1, SSR 🡪 TSR) 🡪 temp0.dat 🡪 Loshift(centershift, tBPS1, tBPS1) 🡪 temp1.dat 🡪 resample(tBPS1 🡪 tBPS2, TSR 🡪 TSR) 🡪 temp2 🡪 accomplish (temp3, temp4)*

tBPS1 ist in dieser Version fix 32

Die benutzten Packages sind:

from COHIWizard\_GUI\_v9 import Ui\_MainWindow as MyWizard

from SDR\_wavheadertools\_v2 import WAVheader\_tools

from player\_threadworkers import StemlabControl

import system\_module as wsys

import resampler\_module\_v3 as rsmp

GUI Version 9 aktuell in Gebrauch

GUI v10 für Playlists und Resampler-Filelists vorbereitet

**Nächster Schritt:**

* **Check open file strategie von readsegment\_new, da dclose manchmal zu oft passiert ???? und bei Aufruf durch LOshifter sowieso ???? open vorher im LOshifter worker ?**

gui.readsegment modifizieren/upgraden auf universelle Formate

gemacht08-12-2023:

TODO: (1) statt wavheader lesen, sollte man SBPS und TBPS vorsehen und übergeben (2) richtiges re-casting auf TBPS (3) Aufrufe der Funktion überall nachziehen

Resampler-Modul: version v3 ab COHIWizard\_v24 eingeführt !

* Derzeit noch alles auf 32bit im LOshifter, Danach könnte man im LOshifter\_worker y\_sh nicht immer als float32 sondern auch mit TBPS flexibel vorformatieren. Untersuchen, ob np.copy beim Umkopieren der Daten dann das re-casting immer richtig macht. LOshifter muss dann get\_tBPS eingebaut bekommen, um nicht automatisch float32 zu erzeugen.
* **Der Modus: Upsampling UND LOshift ist aktuell nicht erlaubt. Beim Versuch dies durchzuführen, stürzte der resampler am Ende ab mit Zugriffsverletzung auf File** 🡪 **? Timingproblem ?**

**PIPELINES:**

**Bedingung 0: TODO: untersuchen ob in Bed 1 zu subsummieren, da ohnehin LOshifter effizient, er macht dann halt nur ein recopy ohne Frequenzschiebung**

Einfaches resampling ohne Speedchange 8z.B. 3%) und ohne LOshift

16 oder 32-bit Sourcefile,

wFormatTag: PCM oder IEEE:

Python existieren.

Standardpipeline:

**resample(SBPS 🡪 TBPS, SSR 🡪 TSR) 🡪 temp1 in TBPS 🡪 accomplish**

**Wurde implementiert als scheduleA() im Resampler Modul, aktuell ist aber TBPS immer 16**

**Bedingung 1:**

Einfaches resampling ohne Speedchange 8z.B. 3%) und mit LOshift

16 oder 32-bit Sourcefile,

TSR < SSR

wFormatTag: PCM oder IEEE

Hier funktioniert der LOshifter ohnehin effizient, da Einleseroutinen für 16 und 32 bit in Python existieren.

Standardpipeline:

**Loshift (centershift, SBPS, TBPS1) 🡪 temp0 in TBPS1 🡪 resample(TBPS1 🡪 TBPS2, SSR 🡪 TSR) 🡪 temp1 in TBPS2 🡪 accomplish**

**Wurde implementiert als scheduleA() im Resampler Modul, aktuell ist aber TBPS1 immer 32 und TBPS2 immer 16**

**Bedingung 2:**

Einfaches resampling ohne Speedchange 8z.B. 3%)

24-bit Sourcefile,

Oder 16/32 bit sourcefile UND TSR > SSR

wFormatTag: PCM oder IEEE:

Hier funktioniert der LOshifter nicht, da keine Einleseroutinen für 24 bit in Python existieren.

Standardpipeline:

**Sourcefile 🡪 resample(24bit 🡪 tBPS1, SSR 🡪 TSR) 🡪 temp0.dat 🡪 Loshift(centershift, tBPS1, tBPS1) 🡪 temp1.dat 🡪 resample(tBPS1 🡪 tBPS2, TSR 🡪 TSR) 🡪 temp2 🡪 accomplish**

**Wurde implementiert als scheduleB24() im Resampler Modul, aktuell aktuell ist aber TBPS1 immer 32 und TBPS2 immer 16**

**Interessanterweise funktioniert das immer, auch bei Banderweiterung (TSR > SSR) und LOshift**

**Das ist das die universelle Variante, wenn (TSR > SSR) und LOshift**

**LOSHIFT müsste TBPS ermöglichen**

**Problem:** LoShifter benutzt zum Einlesen der Daten die Methode gui.readsegment (übrigens: das gehört nicht in gui, TODO) beschrieben. Diese Funktion liest grundsätzlich vom adressierten File auf das Zielarray (Größe DATABLOCKSIZE) in jenem Datenformat ein, das im wavheader des gerade geladenen Files drinsteht. Aktuell:

If BPS == 16:

wFormatTag == 3 🡪 int16 🡪 float16

wFormatTag == 1 🡪 int16 🡪 float32/self.scl self.scl sollte durch einen Übergabeparameter ersetzt werden.

Aktuell in play\_tstarter und cb\_resampler und FileOpen implementiert:

Ist derzeit:if format[0] == 1:

            self.scl = int(2\*\*int(format[2]-1))-1

        else:

            self.scl = 1

Verwendet aber nur als av in showRFdata und in readsegment. Das ist ein CHAOS !!!

If BPS == 32:

wFormatTag == 3 🡪 int32 🡪 float32

wFormatTag == 1 🡪 int32 🡪 float32/ 2147483648 .

If BPS == 24:

wFormatTag == 3 🡪 int32 🡪 format siehe \*) unteres Zitat

wFormatTag == 1 🡪 int32 🡪 float32/ 8388608 .

\*) unteres Zitat:

dataraw = unpack('<%ul' % 1 ,d + (b'\x00' if d[2] < 128 else b'\xff'))

Damit wird als Basis für das Einlese-Format die Info in wavheader[‚nBitsPerSample‘] und wavheader[‚wFormatTag‘] benutzt und das Zielformat ist meist float32 bis auf den Fall int16/3.

Um eine universelle Variante zu haben, müsste man also das Zielformat immer recasten, aber richtig. Bei 16 🡪 32 kein Problem, aber bei 32 🡪 16 muss man die highest significant bits nehmen:

n=(array\_int32>>16).astype(np.int16)

TODO: (1) statt wavheader lessen, sollte man SBPS und TBPS vorsehen und übergeben (2) richtiges re-casting auf TBPS (3) Aufrufe der Funktion überall nachziehen.

Vereinfachungspotenzial: wFormatTag\_TYPE muss man nicht extra zu wFormattag auch noch verwenden. Geht aus von cb\_resample und wird durchgereicht und eigentlich nur für soxstring-Erzeugung in ‚resample‘ verwendet

Abfangen: Wenn ein Filepfad länge als 260 Zeichen wird, und das Programm versucht, dort ein File zu schreiben, crasht es. Abfangen: Pathlengths überprüfen oder Schreibmöglichkeit prüfen.

Instanz: write\_yaml\_header

In

        with open(self.cohiradia\_yamlheader\_filename, 'w', encoding='utf-8') as f:

Wurde mal primär vor Absturz durch Abfrage der Pfadlänge und return False/Error message/reset\_GUI() bewahrt, aber das File bleibt noch geladen. Besser: Alles schließen und grundsätzlich bereits or dem Schreiben irgendwelcher Files den Pfad prüfen und ab bestimmter Länge Warnings ausschreiben.

Im Scheduler: alle automatisch generierbaren Felder der schedule beim nächten cnt automatisch generieren. Durchputzen scheduler et al: Automatisieren von formelbasierten Ausdrücken bei der Definition der schedule im GUI

04-12-2023: Hierzu wurde im LOshifter bereits umgestellt: Bei centershift < 1e-5 wird kein Sonderfall mehr abgefahren, sondern der Schritt LOshifter wird immer ausgeführt. Nur wird bei Abwesenheit einer centershift im LOshifter-worker einfach Eingangs- auf Ausgangsfile kopiert und eben keine Frequenzshift berechnet. Damit wird eine einheitliche Schedule von sch0 weg generiert

**Ausbau auf Filelisten und Auto-Begrenzung auf 2GB:**

**Schnell und elegant, aber sehr komplex:**

Modif LOShifter\_worker: muss getter für FilesizeLimit erhalten und seine Schreibloop beenden, wenn er das Limit erreicht hat.

Modif LOShifter: muss set worker\_FilesizeLimit aus system\_state[„filesizelimit“] erhalten

Modif LOShifter: wenn centershift < 1e-5, nur umschreiben, keine exp-Funktion.

**Langsamer , aber einfach und übersichtlich:**

**Fileliste aus GUI einlesen und in Loop nacheinander abarbeiten: sollte bereits auf cb\_resample-Ebene passieren. Strategie:** Keine verschachtelte Pipeline, die unterschiedlich lange Sourcefiles richtig behandelt, sondern Einzelprozesse mit je einem schedule-Durchlauf pro Sourcefile und Kompilation der finalen, auf 2GB begrenzten targetfiles hinterher mit dem neuen*cat\_replacement\_resize\_files\_postprocessing.py:*

**Ausprobieren Files aus Liste in eine Serie von Files mit 2GB zusammenfügen (mit freien führenden 216 Bit für den Header). Das wäre ein Ersatz für den ‚cat‘ Befehl in LINUX. Checken, wie schnell das geht: Bereits realisiert:** *cat\_replacement\_resize\_files\_postprocessing.py*

**Basis:**



bitte schreib mir ein python programm, das beliebig lange, binäre Eingabe-Files aus einer Liste von Filenamen zu einer Reihe von Ausgabefiles zusammenfügt, die genau 2 GB lang sind

ChatGPT

Hier ist ein Beispiel eines Python-Skripts, das Dateien aus einer Liste von Dateinamen liest und sie in Ausgabedateien mit einer maximalen Größe von 2 GB zusammenfügt:

Bitte modifiziere das Programm so, dass die ersten 216 Bytes der Ausgabedatei nicht nicht

mit Daten sondern Mit Nullen beschrieben werden.

python

import os

def merge\_files(input\_file\_list, output\_file\_prefix, max\_file\_size=2 \* 1024\*\*3): # 2 GB in Bytes

current\_output\_file\_index = 1

current\_output\_file\_size = 0

current\_output\_file\_path = f"{output\_file\_prefix}\_{current\_output\_file\_index}.dat"

with open(current\_output\_file\_path, 'wb') as current\_output\_file:

# Schreibe die ersten 216 Bytes mit Nullen

current\_output\_file.write(b'\x00' \* 216)

for input\_file in input\_file\_list:

with open(input\_file, 'rb') as input\_file:

while True:

# Lese 1 MB Daten aus der Eingabedatei

data\_chunk = input\_file.read(1024\*\*2) # 1 MB in Bytes

# Überprüfe, ob die Eingabedatei vollständig gelesen wurde

if not data\_chunk:

break

# Überprüfe, ob die Ausgabedatei die maximale Größe überschreiten würde

if current\_output\_file\_size + len(data\_chunk) > max\_file\_size:

current\_output\_file\_index += 1

current\_output\_file\_path = f"{output\_file\_prefix}\_{current\_output\_file\_index}.dat"

current\_output\_file\_size = 0

current\_output\_file.close()

current\_output\_file = open(current\_output\_file\_path, 'wb')

current\_output\_file.write(b'\x00' \* 216) # Schreibe die ersten 216 Bytes mit Nullen

# Schreibe Daten in die Ausgabedatei

current\_output\_file.write(data\_chunk)

current\_output\_file\_size += len(data\_chunk)

if \_\_name\_\_ == "\_\_main\_\_":

# Beispiel: Liste von Dateinamen und Präfix für Ausgabedateien

file\_list = ["file1.dat", "file2.dat", "file3.dat"]

output\_file\_prefix = "output\_merged"

# Funktion aufrufen

merge\_files(file\_list, output\_file\_prefix)

**10 min.** Letzte 2 temp files (zu accomplish und terminate) werden nicht mehr gelöscht (sind 0 bytes)

Das Array wird durch Lesen via gui.readsegment (übrigens: das gehört nicht in gui, TODO) beschrieben. Diese Funktion liest grundsätzlich vom adressierten File auf das Zielarray (Größe DATABLOCKSIZE) in jenem Datenformat ein, das im wavheader des gerade geladenen Files drinsteht.

Verbessern:

* read\_segment(,##,DATABLOCKSIZE) aus gui in eine andere Klasse verfrachten
* temp\_wavheader aus dem body in die Parameterliste verlagern.
* Konsequenz: der wavheader müsste beim schedule-Durchlaufen ggf öfter überschrieben werden. Überlegen, ob das eine gute Option ist.

**Bereits getan: Quarantäne:**

**Aufräumen:**

Aktuell in play\_tstarter und cb\_resampler und FileOpen implementiert:

Ist derzeit:if format[0] == 1:

            self.scl = int(2\*\*int(format[2]-1))-1

        else:

            self.scl = 1

Verwendet aber nur als av in showRFdata und in readsegment. Das ist ein CHAOS !!!

**Self.scl erzeugt in FileOpen aus wavheader aber dort nicht verwendet**

self.scl = int(2\*\*int(self.wavheader['nBitsPerSample']-1))-1

**Self.scl erzeugt in play\_tstarter aus format: format wird für Übergabe an playrec\_tworker gebraucht**

self.scl = int(2\*\*int(format[2]-1))-1

**Self.scl erzeugt in cb\_resample, aber dort nicht verwendet**

self.scl = int(2\*\*int(self.wavheader['nBitsPerSample']-1))-1

**Self.scl wird initialisiert**

**Verwendet in showRFdata, wo es aber aus wavheader erzeugt werden könnte**

**Verwendet in readsegment, wo es aber aus wavheader erzeugt werden könnte**

**Fazit: self.scl Total eliminieren und in den beiden Verwendungsinstanzen direkt aus dem wavheader mit der Formel erzeugen**

**Cb\_resample:**

Konfiguriere schedule

Lies ersten Fileheader und ersten filesize

Lies Targetparameter aus der schedule

Swavheader bleibt durchwegs konstant, das sollte kein Problem machen CHECK

Definiere Filesizelimit (2GB), kann Teil der schedule sein 🡪 system\_state[„filesizelimit“]

Generiere:

expected filesize

size\_scalor: Faktor zwischen expected filesize und sourcefilesize

Anfangszeit, Endzeit

Nächsten Targetfilenamen-Prefix

Nächsten Targetfilenamen -> nextfilename

nächsten targetwavheader

Loop 1: sourcefile aus sourcefileliste

Curr\_sourcefilename = sourcefilename 🡪 system\_state[„sourcefn“]

Generiere:

expected filesize

size\_scalor: Faktor zwischen expected filesize und sourcefilesize

while TRUE

Fall 1: sourcefile kürzer als min (intMultiple von Blockalign (size\_scalor \* system\_state[„filesizelimit“]), system\_state[„remainder“]

System\_state[„schedule\_wait“] = True: veranlasst alle activitie\_methods, kein SigIncrSchedule abzusetzen. Folge: *Scheduler bleibt nach einem Step stehen (LOshifter; kein resampler, accomlisher etc)*

SigIncrScheduler.emit

System\_state[„schedule\_int\_step\_terminated“] = False

While System\_state[„schedule\_int\_step\_terminated“] == False

Wait Loop (time.sleep)

*Warte auf Beendigung der aktuellen schedule: CHECK, was man da abfragen könnte. System\_state[„schedule\_int\_step\_terminated“] == True: müsste von allen activitie\_methods gesetzt werden, wenn worker finished?*

Berechne system\_state[„remainder“] = wieviel auf intMultiple von Blockalign (size\_scalor \* system\_state[„filesizelimit“]) – source\_filesize

Setze Scheduler Zähler um 1 zurück, damit der LOShifter beim nächsten sourcefile nochmals aufgerufen wird um das Targetfile fortzusetzen. (system\_state[„targetfn“] bleibt gleich)

Folge: Schließe Konvertierung nicht ab, das temp0File bleibt offen (CHECK im scheduler, ob das so ist !)

Return/break while loop 🡪 Gehe zu loop 1 (targetfile bleibt gleich)

Fall 2: else:

Setze system\_state[„filesizelimit“] = system\_state[„remainder“]

System\_state[„schedule\_wait“] = False

*Scheduler wird voll durchgefahren und erzeugt ein fertiges wav-Targetfile am Ende. Das temp0-File wird gelöscht.*

System\_state[„schedule\_int\_step\_terminated“] = False

While System\_state[„schedule\_int\_step\_terminated“] == False

Wait Loop (time.sleep)

*Warte auf Beendigung der aktuellen schedule: CHECK, was man da abfragen könnte. System\_state[„schedule\_int\_step\_terminated“] == True: müsste von allen activitie\_methods gesetzt werden, wenn worker finished?*

Set Read\_offset in LOshifteractivity to *: ######## where the process stopped in the last call of LOshifter: =* current LOSHIFT\_Filesizelimit (+ something ?)

*(set\_LOshifter\_worker should start next reading sourcefile there…) Sourcefile ist noch aktiv und noch nicht ausgetauscht.*

Generiere:

expected filesize ?

neue Anfangszeit, Endzeit

Nächsten Targetfilenamen-Prefix

Nächsten Targetfilenamen -> nextfilename

nächsten targetwavheader

source\_filesize = source\_filesize - system\_state[„remainder“] ####check condition

*gaukle einen reduzierten filesize vor, um wieder in die erste if-Abfrage zu gelangen*

system\_state[„remainder“] = 2GB

system\_state[„filesizelimit“] = 2GB

*bleibe in while-Loop (sourcefile bleibt gleich)*

Checken, ob alle templatewavheader-Instanzen noch nötig sind (eher nicht) und ggf aufräumen: Wahrscheinlich nein. Falls es beim Testen jemals mal gebraucht würde, poppt eine Errormessage auf. Das entsprechende File wird wohl ab v26 nie mehr gebraucht

**LOShifter mit Threads implementieren:**

* **LOshifter\_worker als Methode der soxwriter-Klasse als gerüst eingebaut**
* **LOshifter Worker slots einführen (sourcefilename)**
* **LOshifterworker Arbeitsloop schreiben**
* **LOshifter actionmethod umbauen: Thread konfigurieren und starten**

**Testen: im Main GUI: Das ist ggf bei erfolglosem Test wieder auf LOshifter statt LOshifter\_new zurückzustellen:**

* #after change 26-11-2023: schedule\_objdict["connect"]["LOshift"] = resamp.LOshifter
* schedule\_objdict["connect"]["LOshift"] = resamp.LOshifter\_new

TESTEN:

Immer wieder kommt es zwischendurch zu Zugriffsverletzungen. Da muss beim Timing was nicht stimmen !

Lösung:

Die expected filesize simmt oft nicht mit der von sox tatsächlich erzeugten Filesize zusammen. Daher manchmal zu rascher Zugriff auf das temp file, bevor sox geendet hat.

sox erzeugt immer wieder bei Resampeln etwas kürzere Files als erwartet, was zu vorzeitiger

Nun wurde in der sox-Statusabfrageschleife des resampler-workers eine komplizierte Abfrage eingebaut: wenn expected filesize fast erreicht ODER keine Änderung der Zielfilesize mehr innerhalb 20 0.5-s-Abfragezyklen (also innerhalb 10s) erfolgt , dann verlasse die sox-Schleife. Damit ist bei den letzten tests alles stabiler gelaufen.

BUG: wenn einmal ein Loshift und resample gelaufen ist, werden bei einem weiteren Mal beim Starten des schedulers beide Threads (LOshifter und soxwriter) gleichzeitig gestartet und führen am Ende zu Absturz. ????

Verfrachte GUI\_Labelementchange Befehle in den scheduler , soweit machbar/nötig

Bisher erfolgreicher Versuch des Fixings: scheduler disconnect vom SigIncrScheduler jedesmal wenn er aufgerufen wird. Ein reconnect muss von den einzelnen action\_Mezhoden durchgeführt werden.

BUG dass tabs nach 24bit warning nicht wiedereingeschaltete werden entfernen

Zusätzl. Temp files immer bei nächstem scheduler run (Erzeugung des nächsten temp-filenamens) neu kreieren (fopen(‚w‘), fclose).

**10 min**. Alte resampler-Klasse von Festplatte zurückspielen:

Undefiniert verändertes Modul umbenannt in resampler\_module\_undef\_changed\_before\_v2.py

Version resampler\_module vom 04-08-2023 zurückeongespielt von Platte COHIRADIA DISK2

**10min** temp wavheader aus allen Files entfernen

**597: LOshifter tofile Befehl erzeugt offenbar 32 bit Datensegmente; mögl deshalb die 32-resample Option im cb\_resample** Das ist offenbar der nun festgelegte Standard

**Konsequenz: übergieb immer einen 32bit- sox befehl, wenn LOshifter vorher gelaufen ist.**

Aufzulösen: system\_state[“sox\_worker”] muss existieren, der wird aber im resampler.resample() festgelegt. Hier existiert eine unklare Situation. Man muss im GUI wissen, dass es einen solchen system state gibt, der woanders (im controller) definiert wird.

        if system\_flags["progress\_source"].find('normal') > -1:  #TODO: solve double function in better datacommunication structure

            progress = system\_flags["progress"]

        elif system\_flags["progress\_source"].find('sox') > -1:

            progress = system\_state["sox\_worker"].get\_progress() #TODO: dazu muss aber system\_state["sox\_worker"] erst einmal existieren

        else:

            print("error, progress source system flag invalid")

            return False

Besser wäre: einen allgemeingültigen State auszulesen, in dem eine Referenz auf sox\_worker existiert, die dann in der GUI-Methode ausgelesen wird, sofern sie existiert, egal wie sie heisst. Sie wird zwar im Controller definiert, aber

Andererseits wird die GUI Methode durch ein Signal ausgelöst, das eh von der entsprechenden Controller-Funktion generiert wird. Daher sollte diese Controllerfunktion die referenz erzeugen und im State eintragen. Aktuell wird die workerfunktion zugewiesen. Man könnet statt ‚system\_state[„sox\_worker“] allgemeiner system\_state[„calling\_worker“] verwenden.

**Lösung: LOshift muss generelle targetBPS können, die aktuellen BPS werden immer weitergereicht über scheduler: check**

**Check ob das mit dem Timing ausgeht (beendet sox bevor scheduler wieder aktiv wird ?)**

**Tgt\_wavheader Bitspers wird falsch berechnet im resampler\_modul L 313, im finalen header ist aber alles richtig**

Für LOShifter volle scheduler-Steuerung

Überfl. Prints #### entfernen

Files nach LOShift hatten oft falsche Längen

Lösung: die temp files müssen nach Gebrauch gelöscht werden, da beim Schreiben immer ein append stattfindet. Das kann der scheduler übernehmen.

PROBLEM: accomplish löst das scheduler Signal nicht aus, wenn es normal läuft, wenn ich debugge und händisch auslöse, geht es. Hier liegt ein Timing problem vor.

Lösung: falsches Signal aufgerufen (self-Version statt gui-Version) : self-Version entfernen ?

Resample erzeugt einen neuen target-wavheader aus dem target\_fn, der typ. Temp0 ist. Dieser Header wird bis zu accomplish durchgereicht. Dabei ist zu diesem Zeitpunkt dieses File vom resampler\_worker noch nicht fertiggeschrieben, also stimmen die stats sicher nicht. Der targetfileheader sollte erst am Ende erstellt werden. Dort alles entfernen und nur im accomplish machen

Die im Scheduler vorkonfigurierte automatische

Filenamensgebung checken: von Schritt zu Schritt kann man ja Indizieren (filenam\_temp\_1, 2, 3…) und erst das endgültige File benennen.

tLO und fshift durch system\_state[„tLO“] und system\_state[„fshift“] ersetzt

Test o.k.

Vorbereitet Codelines (noch kommentiert) im LOshifter-Teil von cb\_resample, Erweiterung der schedule durch schß

Noch zu tun: append richtig stellen

**Versuchen, ob man nach Erweitern der Schedule um sch0 gleich vom scheduler aus LOshifter starten kann**. Natürlich endet das danach im Nirvana, weil ich nicht weiss, ob der scheduler dann wartet.

Ja: scheduler, wird ja eventgesteuert erst beim ‚resample‘ wieder aufgerufen, da ich nach der Rückkehr aus LOshifter wieder im Hauptcode von cb\_resampler fortfahre

**Klären, wie cb\_resampler nach Emission des self.SigLOshift-Signals und damit Starten des** LO\_shifters auf dessen Beendigung wartet:

Ergebnis: LOShift wird abgearbeitet, bevor das Signal emittiert wird !

Offenbar wartet cb\_resampler bis LOshifter fertiggelaufen ist. Ist ja auch kein Thread worker

Prüfen, wo tatsächlich die Signale konfiguriert werden. Sofortige Aktivierung aus der Metastruktur !

Ergebnis: im GUI-Modul, eh im cb\_resample. Der schduler liest dann nur mehr aus diesem dict die abbonements aus, das ist also bereits realisiert.

#TODO: Umstellen auf schedule\_objdict["connect"]["resample"] = resamp.resample\_new

        #schedule\_objdict["connect"]["resample"] = resamp.resample\_new #FFFFFFFFFFFFFFFSSSSSSSSSSSSSSSSSSS  UMSTELLUNG

        schedule\_objdict["connect"]["accomplish"] = resamp.accomplish\_resampling

        #TODO: Umstellen auf schedule\_objdict["connect"]["resample"] = accomplish\_resampling\_new

        schedule\_objdict["connect"]["LOshift"] = resamp.LOshifter

        #TODO: Umstellen auf schedule\_objdict["connect"]["resample"] = resamp.LOshifter\_new

**Problem mit: QThread: Destroyed while thread is still running**

Lösung: der Thread muss beim Definieren lauten:

Statt:

self.soxthread = QThread()

Muss man schreiben:

self.soxthread = QThread(parent = self)

Quelle: https://stackoverflow.com/questions/43647719/pyqt5-qthread-destroyed-while-thread-is-still-running

**Test ob auch dat files richtig LOshifted werden**

### Liste keys System state

**System\_state, Stand 06-12-2023:**

|  |  |
| --- | --- |
| Key | Funktion |
| previous\_RF\_data | Beim vorigen Update des Status beim Übertragen von RF-Daten über die Playloop (worker) ausgelesenes Datensegment; zum Zweck von Korrelationsanalysen |
| Rates | Dict von Indices zu zulässigen Samplingraten f Stemlab |
| Irate | Integer Wert der Samplingrate |
| Irates | Liste der zulässigen Samplingraten f Stemlab |
| gui\_reference | Referenz auf das gui-Objekt |
| Actionlabel |  |
| LO\_offset | Mittenfrequenz-Offset |
| Readoffset | Readoffset für Beginn des Datensegments in wav Header; entw 216 oder 86 |
| schedule\_objdict | Referenzdictionary für scheduler-Signale und Funktionen |
| r\_sch\_counter | Aktuelle Scheduler-Position |
| tLO | Target LO |
| Fshift | Frequenzverschiebung für LOshifter |
| tSR | Target SR |
| s\_wavheader | Wav header der beim Resampling bearbeiteten Source-Datei |
| source\_fn | Name der beim Resampling bearbeiteten Source-Datei |
| target\_fn | beim Resampling zu erzeugenden Ziel-Datei |
| wFormatTag\_TYPE | Typ des Formats: signed-integer, float,… |
| res\_schedule | Schedule als Liste von dicts, deren jedes die Parameter eines Tasks (resample, LOshift, terminate, Accomplish) beinhaltet |
| new\_name | Name der beim Resampling erzeugten Zieldatei |
| res\_blinkstate | Flag, ob das Aktionslabel im Resampler blinken soll oder nicht |
| sSR | SR des source files beim resampeln |
| sBPS | Bytes per Sample beim Sourcefile (resampler) |
| tBPS | Bytes per Sample beim Targetfile (resampler) |
| sfilesize | Filesize des Sourcefiles |
| wFormatTag | FormatTag des Sourcefiles (resampler) |
| t\_filesize | Targetfilesize expected (resampler) |
| accomp\_label |  |
| Playlength | Spielzeit in s |
| Playlist\_ix |  |
| HostAddress | IP Adresse des Stemlab |
| centerfreq |  |

**GUI Elemente**

**Bessere Darstellung mit umlet.exe** , Projektfile = [C:\Users\scharfetter\_admin\Documents\MW\_Aufzeichnungen\COHIRADIA\Softwareentwicklung\SDR\_Postprocessor\methodenablauf.uxf](methodenablauf.uxf)

label\_Filename2

pushButton\_Stop

**Playrec\_tworker()**

Stoploop()

Sig Finished.Emit

**cb\_Butt\_STOP ()**

if self.playthreadactive:

playrec\_tworker.stoploop

stemlabcontrol.sdrserverstop

reactivate all inactive TABs

reset Playbutton

self. playthreadactive = False

else:

return False

self. playthreadactive

**Stemlabcontrol()**

sdrserverstart()

set\_play()

config\_socket()

sdrserverstop()

**EOF\_manager()**

If nextfile or endlessloop:

Read new WAV-header

playthread\_start()

updatecurtime(0) #reset playtime

else:

cb\_Butt\_STOP()

**updatecurtime(+/- self.CURTIMEINCREMENT)**

If self.CUTIMEINCREMENT == 0:

Reset lineEditCurTime

If self.modality == play and !self.pausestate

Increment/decrement self.curtime

Display time in lineEditCurTime

Calculate new fileposition

If |self.CURTIMRINCREMENT| = 1

If self.fileopened

seek (filehandle, new fileposition )

update ScrollBar\_playtime

pushButton\_FF

pushButton\_REW

**Playrec\_tworker()**

Slot 4

**cb\_Butt\_toggleplay ()**

if play:

change Butt to ||

GUIClickPlay()

else:

change Butt to Δ

**Play\_manager()**

if self.playthreadactive

Return

else:

updatecurtime(0) #reset playtime

stemlabcontrol

sdrserverstart()

set\_play()

config\_socket()

if success:

playthread\_start()

else:

return False

return True

ButtonPlay

**play\_tstarter()**

starte playthread, playrec (worker)

SigFinished.connect(EOF\_manager)

SigIncrementCurtime.connect(updatetimer(0))

if running:

self.playthreadactive = True

inactivate all other GUI Tabs

return True

else:

errormessage

return False

**Stemlabcontrol()**

sdrserverstart()

set\_play()

config\_socket()

sdrserverstop()

**Playrec\_tworker()**

Schreib Daten zu STEMLAB

SigFinished.emit

stoploop()

**cb\_Butt\_STOP ()**

if self.playthreadactive

Return

Else:

Self.playrec.stoploop()

Stemlabcontrol.sdrserverstop()

**EOF\_manager()**

If nextfile or endlessloop:

Read new WAV-header

playthread\_start()

updatecurtime(0) #reset playtime

else:

cb\_Butt\_STOP()