CANalyzat0r Documentation

Release 1.0

pschmied (SCHUTZWERK GmbH)

Jul 21, 2017

CONTENTS:

CHAPTER

ONE

REQUIREMENTS

To install all requirements, please execute the following commands:

- sudo apt-get install python3.5 python3-pip python3-pyside can-utils ffmpeg
- sudo pip3 install pyvit sphinx_rtd_theme

Note: You can just execute `install.requirements.sh`.

1.1 Docker

There's also a docker container available. Check the docker folder.

CHAPTER

TWO

CANALYZAT0R MANUAL

2.1 Introduction

You can use **CANAlyzat0r** to quickly analyze the CAN bus in many ways. It’s great.

This documentation will guide you through the usage of the application. Also, you can find the code documentation in this document if you want to extend and/or contribute to this project.

2.2 Usage: Tab by tab

2.2.1 Main Tab

Welcome to CANalyzat0rs main tab! Here you can change interface settings and creat/remove virtual CAN interfaces. Don't worry, the kernel modules should aready be loaded for you.

2.2.1.1 Where's my interface?!?!1!

If you can't find your attached CAN interface in the ComboBox, please check the output of `ifconfig -a`. In order to use your interface with CANAlyzat0r, a SocketCAN device must be present. Maybe you have to load another kernel module/driver?

2.2.1.2 Creating and selecting projects

On a fresh startup, you should encounter a message saying that a new project should be created. You can still use this application without a selected project. However, one can't save dumps or known packets. To create a project, please refer to the manager tab. After you have created a project there, you can set it as active project in the main tab.

2.2.1.3 Log levels

You can set the minimum log level for which messages will be printed to the log box in this tab.

2.2.1.4 Where's my data being saved to?!!?

By default, CANalyzat0r creates a SQLite database called "database.db" in the data folder. Please take care of this file as everything you discover is saved here.

2.2.1.5 But what if i want to export my data?

Please check the manager tab and learn on how to export projects and dumps.

2.2.2 General

2.2.2.1 Why can I change interface settings in every tab and why is there a global interface?

You can set a global interface in order to set the selected interface for every inactive tab. On top of that, you can override this setting for every tab individually using the button. This allows you to e.g. fuzz on `can0` and sniff on `vcan1` at the same time.

2.2.2.2 Well, how can I manage packets in between tabs?

You can:

- Select rows and copy them to another tab (if allowed)
- Delete rows by selecting rows and pressing `Del` on your keyboard

2.2.2.3 Ok Ok, but how can I import my SocketCAN dumps?

Just copy and paste them into the GUI tables <:

2.2.2.4 What are known packets?

Once you discovered that packet XY does Action ZZ on your car or setup, you can add this knowledge to the database using the manager tab. This adds the discovered information globally for a specific project. Using this, the "Description" column in the GUI tables in filled with data, so you can recognize a re-occuring packet.

2.2.2.5 I've discovered a bug, pls fix!

Pleae report bugs using GitHub issues, Thanks.

2.2.3 Sniffer Tab

2.2.3.1 How to sniff?

It's simple. For every discoverd interface you can find a sniffer tab here. If no tab for your desired interface is displayed, please re-check all available interfaces using the button on the main tab. Once you find your desired interface, just click on start.

2.2.3.2 Ignoring packets

You can add tab specific packets to the ignore list. Hint: Use a blank data field if you want to exclude whole IDs.

Another Hint: You can use the fitering buttons to remove background noise. It's great.

2.2.3.3 I get errors/no data when I try to sniff!!1!

- Maybe your SocketCAN interface disappeared?
- Maybe you have selected a wrong bitrate?
- Have you tried turning it off and on again?

2.2.3.4 The sniffer tab doesn't list the packets I'm sending

Thats normal. You will only see packets that you are receiving on a specific interface. This prevents the packets that you generate via fuzzing from being in your sniffed dump. If you really need all packets in one dump, you can use `candump`.

2.2.4 Sender Tab

2.2.4.1 What do the sub tabs do?

You can create multiple sender tabs to handle sending various packet sets.

2.2.4.2 Why does `Loop` do?!?!

One can either send packets once or in a loop with a specific packet gap. Imagine you have to send a packet every X seconds to keep a CAN device online: It's handy.

2.2.5 Fuzzer Tab

2.2.5.1 What does this thing to?

Using this tab you can send random packets into the CAN bus to discover things. You can tune settings that control random packet generation using the GUI elements.

2.2.5.2 What are masks?

You can write static values into the masks or put an X if that character should be randomized. Using this, you can freely control the payload of generated packets. Hint: You can change masks and lengths **while fuzzing**.

2.2.5.3 Other fuzzers are much faster!!1!

This is a python based fuzzer which also displays the packets on the GUI. This convenience costs performance. If you want the best performance you can use `cangen` of the `can-utils` package and import the created packets later.

2.2.5.4 What are the modes?

- User specified: You can freely specify ID and data masks
- 11 bit IDs / 29 bit IDs: Only short/extended IDs will be used

2.2.6 Comparer Tab

2.2.6.1 What can I compare?

You can compare two sets of packets. You will get all packets they have in common.

2.2.7 Searcher Tab

2.2.7.1 What can I search for?

Using this tab you can perform a binary packet search for a specific packet or a whole packet set that cause an effect. Let's suppose you've fuzzed and got a large packet dump that, when replayed, causes an effect on your CAN device / car. You now want to extract the relevant packet(s) out of that dump. Searcher tab to the rescue – Load the whole packet dump and let the analyzer routine guide you. Note: This first tries to search for **1** packet that causes an action. It this fails, the searcher tries to continously minimize the packet set.

2.2.7.2 It doesn't work!!1!

Don't give up too fast, try the following things: - Set the packet gap to a lower value, you can even try 0 - Just try again and hope for better shuffling - Use another dump/fuzz again, . . . - Wait a few seconds after each chunk

2.2.7.3 It still doesn't work :(

CAN devices can be extremely tricky, for example spedometers. Depending on your dump, you may have to try it multiple times with the same dump because of packet timings and/or bad luck. If you replay your whole dump and see the desired action, you will be able to find it using the searchter tab.

2.2.7.4 I want to do it manually, how can this tool help me?

Create a new sender, add the dump to it and send them in a loop. Minimize the packet set from the bottom using your "CTRL+C" and "DEL" and try again. If it didn't perform the desired action, paste the packets again and delete other packets.

2.2.8 Manager Tab

2.2.8.1 What can I do using the dumps tab?

You can save a set of packets that you want to keep (e.g. for further analysis) and save it to the database. This allows you to load the dump again at a later point. Hint: You can edit the values in the GUI table and update the values in the database using the update button.

2.2.8.2 I know packet XY has effect ZZ, do I create a dump or a known packet?

Just create a dump with one packet entry and the application will handle the rest for you.

2.2.8.3 Importing/Exporting projects

If you want to import/export projects, use the manager tab. It exports all saved data of a project to a editable textfile in JSON format. Go ahead and edit values if you want, but be careful and don't mess with the data integrity <:

2.2.9 Filter Tab

2.2.9.1 What can I filter?

Let's suppose you want to find (a) specific packet(s) that get generated when you (for example) press a button, accelerate or lock the cars doors. The captured CAN traffic contains so much data that you can't seem to find the packets easily. Let's use the filter tab:

- You can collect background noise containing CAN packets that are sent on the bus without any user interaction
- After that, a variable amount of samples get captured. You have to perform the desired action **in every sample** - e.g. lock the doors in every sample.
- As soon as all data has been captured, the filter tab begins to analyze it. It filters background noise out of each sample and tries to find packets that occur in every sample. These are most likely the packets your are looking for.

2.2.9.2 How can I try this without a car or CAN device?

Use ICSim!

CHAPTER

THREE

CONTRIBUTING

Here's some useful info if you want to contribute.

3.1 Guidelines

- Each tab has its own Class. If possible, inherit from `AbstractTab`.
    - To provide comatibility:
    - The displayed data should also be in a raw data list called `rawData` which is *always* up to date
    - `prepareUI` initializes all GUI elements
    - `active` manages the status of a tab
    - Tab specific CANData instances are called `CANData`
- Please log useful information using an own logger instance
- Use existing Toolbox methods if possible
- Use batch database operations using raw lists (not objects) for better performance
- Use docstrings
- Keep the `.ui` files clean: Always name new GUI elements properly according to existing ones
- Put new strings in the Strings file and reference it

3.2 I want to add a new tab, what do I have to do?

- Create a new tab on the GUI and stick to the already existing naming conventions
- Add a QTableView to display your data and other GUI elements
- Update *mainWindow.py* using *pyside-uic mainWindow.ui > mainWindow.py*.
- Add a new File and a new class which inherits from `AbstractTab`
- Call the parents constructor in your `__init__`
- Add the GUI elements from the `.ui` file to your code. You can refer to the other tabs to see how it's done. Also, add the click handlers here.
- Call `prepareUI` as last action in `__init__`
- If your tab needs an interface or displays interface values: Add your tab class or instance to `updateInterfaceLabels()` and/or `updateCANDataInstances()`.

- If your tab uses an instance: Add an instance to *Globals.py* and create one at startup (see *CANalyzat0r.py*).
- If your tab uses a static class: Call *prepareUI* at startup (see *CANalyzat0r.py*).

CHAPTER

FOUR

SRC PACKAGE

4.1 Subpackages

4.1.1 src.ui package

4.1.1.1 Submodules

4.1.1.2 CANalyzat0r.ui.mainWindow module

class `src.ui.mainWindow.`**`Ui_CANalyzatorMainWindow`**

Bases: `object`

`retranslateUi`(*CANalyzatorMainWindow*)

`setupUi`(*CANalyzatorMainWindow*)

4.1.1.3 Module contents

4.2 Submodules

4.3 CANalyzat0r.AboutTab module

Created on Jun 26, 2017

@author: pschmied

class `src.AboutTab.`**`AboutTab`**

This class handles the logic of the about tab.

static **`browseGitHub`**(*event*)

Opens the SCHUTZWERK website.

Parameters **`event`** – Dummy, not used.

static **`browseSW`**(*event*)

Opens the SCHUTZWERK website.

Parameters **`event`** – Dummy, not used.

static **`prepareUI`**()

This just sets up the GUI elements.

4.4 CANalyzat0r.CANData module

Created on May 17, 2017

@author: pschmied

class `src.CANData.`**`CANData`**`(`*ifaceName*, *bitrate=500000*`)`

`CANDataInstances` **= {}**
  This dictionary stores all currently available CANData instances. The key of the dictionary is the interface name

`__init__``(`*ifaceName*, *bitrate=500000*`)`
  This method initializes the pyvit CAN interface using the passed parameters and the start() method. Please note the active-flag which protects the object from being deleted while being in use.

  Parameters

  - **`ifaceName`** – Name of the interface as displayed by `ifconfig -a`
  - **`bitrate`** – Desired bitrate of the interface

`checkVCAN``()`
  Checks if the SocketCAN device is phyisical or virtual using a `ls` call to `/sys/devices/virtual/net`.

  Returns A boolean value indicating if the device is virtual (True) or not (False)

static **`createCANDataInstance`**`(`*ifaceName*, *bitrate=500000*, *returnObject=False*`)`
  Creates a CANData instance with the desired data and either returns the object or adds it to the CANDataInstances dictionary.

  Parameters

  - **`ifaceName`** – The desired interface name
  - **`bitrate`** – The bitrate
  - **`returnObject`** – Boolean value indicating whether the created object will be returned or appended to the dictionary (XOR)

  Returns The created CANData object if returnObject is True

static **`deleteCANDataInstance`**`(`*ifaceName*`)`
  Removes a CANData object from the CANDataInstances dictionary. Note: CANData objects only will be deleted if the active flag is set to False to prevent running operations from failing.

  Parameters **`ifaceName`** – The name of the interface that will be deleted

  Returns A Boolean value indicating the success of the delete operation

static **`getGlobalOrFirstInstance`**`()`
  Tries to return an available CANData instance (e.g. for startup of the application).

  Returns

  - The global CANData instance if available.
  - Else: The first element of all available instances.
  - Else: None of no interface is present

`logger` **= <logging.Logger object>**
  Class specific logger instance

static `parseSocketCANLines`(*lines*)

Parses a list of SocketCAN lines and generates a list of SocketCANPackets. Note: The expected line format is e.g.:

```
(1493280437.565631) can0 1FD#0000000000000000
```

Parameters **`lines`** – List of lines in SocketCAN format

Returns List of SocketCANPacket objects

static `readCANFile`(*filePath*)

Reads a file in SocketCAN format (as generated by candump from can-utils) and returns a list of SocketCANPacket objects (see *`SocketCANPacket`*).

Parameters **`filePath`** – The path of the dump file that has to be read

Returns A list of SocketCANPackets

`readPacket`()

Read a packet from the queue using the SocketCAN interface. Note: This blocks as long as no packet is being received. You can use *`readPacketAsync()`* to read packets with a timeout.

Returns A packet as pyvit frame

`readPacketAsync`()

Read a packet from the queue using the SocketCAN interface **and a timeout**.

Returns A packet as pyvit frame or None of no packet was received

static `rebuildCANDataInstances`(*CANIfaceNameList*)

Refreshes the CANDataInstances dictionary with up-to-date values using the parameter CANIfaceNameList. Old objects will be kept, new ones will be created and missing ones will be deleted.

Parameters **`CANIfaceNameList`** – Names of interfaces that must be present in the dictionary after this method

Returns A list of removed interface names to handle the consequences of deleting an object.

`sendPacket`(*packet*)

Sends a packet using the SocketCAN interface.

Parameters **`packet`** – A packet as pyvit frame (see *`tryBuildPacket()`*)

`timeout` = None

1 second read timeout for async reads (see *`readPacketAsync()`*)

`toString`()

Return a string to display the currently used settings and interface name on the GUI

Returns A string which consists of either: The interface name (for virtual CAN devices where no bitrate is available) or the interface name along with the currently used bitrate in kBit/s

static `tryBuildPacket`(*CANID*, *data*)

Builds a pyvit frame using the passed parameters. This method automatically uses the extended CAN format if needed.

Parameters

- **`CANID`** – The CAN ID as hex string
- **`data`** – The desired packet data (length must be even)

Returns A packet as pyvit frame containing the passed data or None of no frame can be created

`updateBitrate`(*bitrate*)

Updates the bitrate of the SocketCAN interface (if possible).

Parameters **`bitrate`** – The desired bitrate in bit/s

Returns A boolean value indicating success of updating the bitrate value

static **`writeCANFile`**(*filePath*, *packets*)

Writes/Exports SocketCANPacket objects to a textfile.

Parameters

- **`filePath`** – Path of the file to be saved to
- **`packets`** – List of SocketCANPacket objects

class `src.CANData.`**`SocketCANPacket`**(*timestamp*, *iface*, *id*, *data*)

This class is used to manage data from/to SocketCAN format in a nice manner <:

`__init__`(*timestamp*, *iface*, *id*, *data*)

This just sets data.

Parameters

- **`timestamp`** – Timestamp of the packet
- **`iface`** – Interface the packet was captured from
- **`id`** – CAN ID
- **`data`** – Data payload

`toString`()

Returns the string representation of the current object. ID lengths will be padded:

- length <= 3 –> length = 3
- length > 3 –> length = 8

Returns A string with the data of the current object

4.5 CANalyzat0r.CANalyzat0r module

Created on May 17, 2017

@author: pschmied

class `src.CANalyzat0r.`**`MainWindow`**

Bases: `PySide.QtGui.QMainWindow`, *`src.ui.mainWindow.Ui_CANalyzatorMainWindow`*

`__init__`()

This method has to take care of the following things:

0. Lazy import tab modules
1. Initialize the main UI
2. Setup the database connection and ensure all tables are present
3. Detect all currently attached CAN devices
4. Call the prepareUI()-method of every tab
5. {En, Dis}able GUI elements based on the presence of a CAN device
6. Setup logging
7. Install a globlal exception hook that will catch all “remaining” exceptions to log it to the GUI

8. Load the CAN kernel modules
9. Check if superuser privileges are present - exit if not present
10. Install event handlers for GUI elements (assignWidgets())

assignWidgets()

This method connects all GUI elements of the static tabs to their event handlers. For all other tabs (those that inherit from *AbstractTab*, this is done in the constructor

checkSU()

This method gets the effective UID and returns a boolean value indicating if root privileges are available.

Returns: A boolean value indicating the superuser status

static cleanup()

This gets called when exiting. This cleans up everything <:

staticMetaObject = <PySide.QtCore.QMetaObject object>

src.CANalyzat0r.**globalLoggingHandler**(*type*, *value*, *tb*)

This is the handler method for the global exception hook which parses the message in the traceback (tb). Using this it is possible to log the exception and the last executed line of code.

Parameters

- **type** – Exception class
- **value** – Exception value (the object)
- **tb** – Traceback object containing the previously exectuted lines of code

src.CANalyzat0r.**uncaughtExceptionLogger = <logging.Logger object>**

Logger instance to log uncaught exceptions using *globalLoggingHandler()*

4.6 CANalyzat0r.ComparerTab module

Created on Jun 30, 2017

@author: pschmied

class src.ComparerTab.**ComparerTab**(*tabWidget*)

Bases: *src.AbstractTab.AbstractTab*

This handles the logic of the comparer tab.

__init__(*tabWidget*)

This just sets data and adds click handlers.

compare()

Compares rawPacketSet1 and rawPacketSet2 and display the packet they have in common on the GUI.

getPacketSet(*rawPacketList*)

Opens a *PacketsDialog* to load selected packets into the passed raw packet list.

Parameters **rawPacketList** – The raw packet list

setPacketSet1()

Opens a *PacketsDialog* to load packet set 1 into rawPacketSet1.

setPacketSet2()

Opens a *PacketsDialog* to load packet set 2 into rawPacketSet2.

4.7 CANalyzat0r.Database module

The database model is as follows:

Note: The **bold** columns are `NOT NULL` Created on May 17, 2017

@author: pschmied

class `src.Database.`**`Database`**

This class handles the database connection and is responsible for creating, deleting and updating values

`__init__``()`

This method does the following things:

1. Setup logging
2. Create a DB connection and check the integrity
3. Create tables if necessary

`checkDB``()`

Checks if all the table count of the SQLite database matches the needed table count. If the check does pass the user will be notified to create a project if no project is exisiting yet. If the check does not pass the user will be prompted for an action:

- Truncate the database and create an empty one
- Keep the database and exit

Returns A boolean value indicating the database integrity status (True = good)

`connect``()`

Connect to the hard coded SQLite database path (see Settings).

Note: If a DatabaseError is encountered, the application will close with error code 1

Returns A SQLite3 connection object

`createTables``()`

Creates all needed tables. Check *`DatabaseStatements`* for the SQL statements.

`deleteFromTableByID``(`*tableName*, *id*`)`

Delete a row from a table with a specific ID.

Parameters

- **`tableName`** – The table to delete from
- **`id`** – ID of the record to delete

`deleteFromTableByValue``(`*tableName*, *column*, *value*`)`

Delete rows from a table with a specific value.

Parameters

- **`tableName`** – The table to delete from
- **`column`** – The column to check
- **`value`** – The value to search for

`deleteKnownPacket``(`*knownPacket*`)`

Delete a KnownPacket object

Parameters **`knownPacket`** – The KnownPacket object to delete

Returns

deletePacketSet (*packetSet*)
Delete a PacketSet along with the associated packets

Parameters **packetSet** – The PacketSet object to delete

Returns

deleteProjectAndData (*project*)
Delete a project and all associated data.

Parameters **project** – The Project object to delete

getKnownPackets (*project=None*)
Get all known packets of a Project as objects. Uses the global project if no project is given.

Parameters **project** – Optional parameter to specify the project to use

Returns A list of all known packets as KnownPacket objects

getOverallTableCount (*tableName*)
Returns the count(*) of a table.

Parameters **tableName** – The table to count the rows of

Returns The number of rows as integer

getPacketSets (*project=None*)
Get all packet sets of a Project as objects. Uses the global project if no project is given.

Parameters **project** – Optional parameter to specify the project to use

Returns A list of all known packets as PacketSet objects

getPacketsOfPacketSet (*packetSet*, *raw=False*)
Get all packets of a specific packet set. Note: Use raw=True for better performance.

Parameters

- **packetSet** – All returned packets will belong to this packet set
- **raw** – Boolean value to indicate if the packets will be returned as raw data list (True) or as list of objects (False)

Returns Depending on the value of raw: - True: List of value lists (raw data) - False: List of Packet objects

getProjects ()
Get all available projects as Project objects.

Returns A list of all projects as Project objects

saveKnownPacket (*knownPacket*)
Save a known packt to the database.

Parameters **knownPacket** – The KnownPacket object to save

Returns The database ID of the saved known packet

savePacket (*packet=None*, *packetSetID=None*, *CANID=None*, *data=None*, *timestamp=”*, *iface=”*, *commit=True*)
Save a packet to the database: Either by object Oo by list-values –> Faster for many values If the value in packet is not None: The passed object will be used Else: The seperate values will be used

Parameters

- **packet** – Optional parameter: Packet object to save

- **packetSetID** – PacketSet ID of the saved packet
- **CANID** – CAN ID
- **data** – Payload data
- **timestamp** – Timestamp
- **iface** – The interface the packet was captured from
- **commit** – If the operation will be commite to the database or not. Batch operations use commit=False

Returns The database ID of the saved packet if commit is True. Else -1

savePacketSet (*packetSet*)
Save a PacketSet to the database. If there's a name collision, the user will be prompted for a new and unique name.

Parameters **packetSet** – The PacketSet to save

savePacketSetWithData (*packetSetName*, *rawPackets=None*, *project=None*, *packets=None*)
Save a packet set in the database with the given data. If no project is given, the global project will be used. You must specifiy `rawPackets` or `packets`.

Parameters

- **packetSetName** – The desired name of the PacketSet
- **rawPackets** – Optional: Raw Packet data (List of lists). You can also use `packets`.
- **project** – Optional parameter to specify a project. If this is not specified, the selected project will be used
- **packets** – Optional; List of packet objects to save to the packet set. If this is specified, `rawPackets` will be ignored

Returns

savePacketsBatch (*packetSetID*, *rawPackets=None*, *packets=None*)
Save many packets as a batch to the database. Use this for improved speed: No objects, only 1 commit.

Parameters

- **packetSetID** – The PacketSet ID the packets belong to
- **rawPackets** – Optional: Packet data as raw data list (List of lists)
- **packets** – Optional: List of packet objects to save. If this is not None, this will be used instead of `rawPackets`

saveProject (*project*)
Save a Project objet to the database

Parameters **project** – The project to save

Returns The assigned database ID

updateKnownPacket (*knownPacket*)
Update a known packet object in the database

Parameters **knownPacket** – The KnownPacket object which holds the updated values

Returns

updatePackets (*rowList*, *packetSet*, *packetIDsToRemove*)
Update the packets of a specific packet set

Parameters

- **`rowList`** – A list containing raw packet data e.g.: `[[("CANID", "232"), ("DATA", "BEEF"), packetID], [("DATA", "C0FFEE"), ("CANID", "137"), packetID]]`
- **`packetSet`** – The PacketSet object the data in rowList belongs to
- **`packetIDsToRemove`** – Database IDs of packets to remove

`updateProject`(*project*)
Update a Project object in the database.

Parameters **`project`** – The Project object which holds the updated values

class `src.Database.`**`DatabaseStatements`**
This class is used to store and generate database statements.

`checkTablesPresentStatement` = **"SELECT name FROM sqlite_master WHERE type='table'"**
Statement which gets the names of all currently available tables in the database to check the integrity

`createKnownPacketTableStatement` = **'CREATE TABLE 'KnownPacket' (\n\t'ID'\tINTEGER PRIMARY KEY,\n**

`createPacketSetTableStatement` = **'CREATE TABLE 'PacketSet' (\n\t'ID'\tINTEGER PRIMARY KEY,\n\t'Proj**
Note: Unique index for the combination of Project ID and Name –> A PacketSets name is unique for a project

`createPacketTableStatement` = **'CREATE TABLE 'Packet' (\n\t'ID'\tINTEGER PRIMARY KEY,\n\t'PacketSetII**

`createProjectTableStatement` = **'CREATE TABLE 'Project' (\n\t'ID'\tINTEGER PRIMARY KEY,\n\t'Name'\tT**
Project names are unique

`createTableStatementsList` = **['CREATE TABLE 'Project' (\n\t'ID'\tINTEGER PRIMARY KEY,\n\t'Name'\tTE**
Holds all needed create table statements

static **`getDeleteByIDStatement`**(*tableName*, *id*)
Returns an SQL delete statement using an ID where clause using *`getDeleteByValueStatement()`*

Parameters

- **`tableName`** – The table name to delete from
- **`id`** – Desired ID value for the where clause

Returns

The resulting SQL statement

e.g.: `DELETE FROM TABLE1 WHERE ID = 1337`

static **`getDeleteByValueStatement`**(*tableName*, *column*, *value*)
Returns an SQL delete statement using a where clause using *`getDeleteByValueStatement()`*

Parameters

- **`tableName`** – The table name to delete from
- **`column`** – Desired column for the where clause
- **`value`** – Desired column value for the where clause

Returns

The resulting SQL statement

e.g.: `DELETE FROM TABLE1 WHERE NAME = 'BANANA'`

static **`getInsertKnownPacketStatement`** (*projectID*, *CANID*, *data*, *description*)
Returns an SQL insert statement for a KnownPacket using *`getInsertStatement()`*.

Parameters

- **`projectID`** – The project this KnownPacket belongs to
- **`CANID`** – CAN ID
- **`data`** – Payload data
- **`description`** – What this specific Packet does

Returns The SQL insert statement with all KnownPacket specific values set

static **`getInsertPacketSetStatement`** (*projectID*, *name*, *date*)
Returns an SQL insert statement for a PacketSet using *`getInsertStatement()`*.

Parameters

- **`projectID`** – The project ID the record belongs to
- **`name`** – The name of the PacketSet
- **`date`** – Date

Returns The SQL insert statement with all PacketSet specific values set

static **`getInsertPacketStatement`** (*packetSetID*, *CANID*, *data*, *timestamp*, *iface*)
Returns an SQL insert statement for a Packet using *`getInsertStatement()`*.

Parameters

- **`packetSetID`** – The PacketSet this Packet belongs to
- **`CANID`** – CAN ID
- **`data`** – Payload data
- **`timestamp`** – Timestamp of the packet
- **`iface`** – Interface name from which this packet was captured from

Returns The SQL insert statement with all Packet specific values set

static **`getInsertProjectStatement`** (*name*, *desription*, *date*)
Returns an SQL insert statement for a project using *`getInsertStatement()`*.

Parameters

- **`name`** – Projectname
- **`desription`** – Project description
- **`date`** – Project date

Returns The SQL insert statement with all project specific values set

static **`getInsertStatement`** (*tableName*, *columnList*, *valuesList*)
Builds a SQL insert statement using the passed parameters

Parameters

- **`tableName`** – The table name to insert into
- **`columnList`** – List of column names that will be affected
- **`valuesList`** – List of values to put into the columns

Returns

An SQL insert statement with the desired values mapped to the columns

e.g. `INSERT INTO TABLE1 (col1, col2) VALUES (1, 2)`

static `getOverallCountStatement`(*tableName*)

Returns an SQL select count statement for the desired table using all rows

Parameters **`tableName`** – The table name to get the rowcount from

Returns

The resulting SQL statement

e.g.: `SELECT COUNT(*) FROM TABLE1`

static `getSelectAllStatement`(*tableName*)

Returns an SQL select statement to get all data from a table.

Parameters **`tableName`** – The table name from which data will be selected

Returns

The resulting SQL select statement

e.g.: `SELECT * FROM TABLE1`

static `getSelectAllStatementWhereEquals`(*tableName*, *column*, *value*)

Returns an SQL select statement to gather all data from a table using a where clause

Parameters

- **`tableName`** – The table name from which data will be selected
- **`column`** – The column which the where clause affects
- **`value`** – The desired value of the column

Returns

The resulting SQL statement with where clause

e.g.: `SELECT * FROM TABLE1 WHERE ID = 1337`

static `getUpdateByIDStatement`(*tableName*, *colValuePairs*, *ID*)

Builds a SQL update statement using the passed parameters **and an ID**

Parameters

- **`tableName`** – The table name to update
- **`colValuePairs`** – List of tuples: (column, desired value)
- **`ID`** – The ID of the record to update

Returns

An SQL update statement with the desired values mapped to the columns using the ID

e.g. `UPDATE TABLE1 SET col1 = 1, col2 = 2 WHERE ID = 1337`

`knownPacketTableCANIDColName` = 'CANID'

`knownPacketTableDataColName` = 'Data'

`knownPacketTableDescriptionColName` = 'Description'

`knownPacketTableName` = 'KnownPacket'

`packetSetTableName` = **'PacketSet'**

`packetTableCANIDColName` = **'CANID'**

`packetTableDataColName` = **'Data'**

`packetTableName` = **'Packet'**

`projectTableDescriptionColName` = **'Description'**

`projectTableName` = **'Project'**

`projectTableNameColName` = **'Name'**

`tableCount` = **4**
The Amount of tables that must be present

4.8 CANalyzat0r.FilterTab module

Created on Jun 02, 2017

@author: pschmied

class `src.FilterTab.`**`DataAdderThread`**(*snifferReceivePipe*, *sharedEnabledFlag*, *curSampleIndex*)
Bases: `PySide.QtCore.QThread`

This thread receives data from the sniffer process and emits a signal which causes the main thread to add the packets.

`__init__`(*snifferReceivePipe*, *sharedEnabledFlag*, *curSampleIndex*)

`frameToList`(*frame*)
Converts a received pyvit frame to raw list data. After that, the data is emitted using `signalSniffedPacket`

Parameters **`frame`** – pyvit CAN frame

`run`()
As long as `sharedEnabledFlag` is not set to `0` data will be received using the pipe and processed using *`frameToList()`*.

`signalSniffedPacket` = **<PySide.QtCore.Signal object>**
Emit a signal to the main thread when items are ready to be added This emits the packet and the current sample index

`staticMetaObject` = **<PySide.QtCore.QMetaObject object>**

class `src.FilterTab.`**`FilterTab`**(*tabWidget*)
Bases: *`src.AbstractTab.AbstractTab`*

This class handles the logic of the filter tab

`__init__`(*tabWidget*)

`addSniffedNoise`(*dummyIndex*, *packet*)
Adds the passed packet data to `noiseData`. This method gets called by a DataAdderThread.

Parameters

- **`dummyIndex`** – Not used, only exists to match the signature defined in the signal of the DataAdderThread
- **`packet`** – The packet object to extract and add data from

addSniffedPacketToSample (*curSampleIndex*, *packet*)
Adds a sniffed packet to the sample defined by `curSampleIndex`. Gets called by a DataAdderThread.

Parameters

- **curSampleIndex** – The sample index to get a list from `rawData[curSampleIndex]`
- **packet** – Packet data to add

Returns

analyze (*removeNoiseWithIDAndData=True*)

Analyze captured data:

1. Remove sorted noise data (if collected): For each sample:
   - Sort the sample to increase filtering speed
   - Remove noise
2. **Find relevant packets:**
   - Sort each sample to increase filtering speed
   - Assume that all packets of the first sample occurred in every other sample
   - Take every other sample and compare

Depending on `removeNoiseWithIDAndData` noise will be filtered by ID and data (default) or ID only

Parameters **removeNoiseWithIDAndData** – Optional value: Filter data by ID and Data or ID only

clear (*returnOldPackets=False*)
Clear the currently displayed data on the GUI and in the lists.

collectNoise (*seconds*)
Collect noise data and update `noiseData`. Uses the processes/threads started in *`startSnifferAndAdder()`*.

Parameters **seconds** – Amount of seconds to capture noise

Returns True if noise was captured. False if the user pressed "Cancel"

getSampleData (*sampleAmount*, *curSampleIndex*)
Collect sample data and add the sniffed data to `rawData[curSampleIndex]`. Uses the processes/threads started using *`startSnifferAndAdder()`*.

Parameters

- **sampleAmount** – Amount of samples to collect
- **curSampleIndex** – Index of the currently captured sample in `rawData`

noiseData = None
Noise that will be substracted from the collected data

outputRemainingPackets (*remainingPackets*)
Output all remaining packets after filtering to the table view. Note: This also clears previous data

Parameters **remainingPackets** – Raw packet list to display

sharedDataAdderEnabledFlag = None
Shared process independent flag to terminate the data adder

sharedSnifferEnabledFlag = None
Shared process independent flag to terminate the sniffer process

startFilter()

Handle the filtering process:

1. Collect noise
2. Record sample data
3. Analyze captured data

startSnifferAndAdder(*adderMethod*, *curSampleIndex=-1*)
Start a DataAdderThread and a SnifferProcess to collect data. They will communicate using a multiprocess pipe to collect data without interrupting the GUI thread.

Parameters

- **adderMethod** – The DataAdderThread will call this method to handle the received data
- **curSampleIndex** – The index of the currently captured sample (-1 as default)

stopSnifferAndAdder()
Stop the DataAdderThread and SnifferProcess using the shared integer variable.

toggleGUIElements(*state*)
{En, Dis}able all GUI elements that are used to change filter settings

Parameters **state** – Boolean value to indicate whether to enable or disable elements

updateNoiseCollectProgress(*progressDialog*, *value*)
Update the text and progressbar value displayed while collecting noise.

Parameters

- **progressDialog** – The QProgressDialog to update
- **value** – The value to set the progressbar to

4.9 CANalyzat0r.FuzzerTab module

Created on May 31, 2017

@author: pschmied

class src.FuzzerTab.**FuzzerTab**(*tabWidget*)
Bases: *src.AbstractTab.AbstractTab*

This class handles the logic of the fuzzer tab

IDMask = None
The ID is 8 chars max. - initialize it with only X chars

IDMaskChanged()
This allows changing the ID mask values on the fly because a new value will only be set if the new value is valid.

IDMaxValue = None
Default: allow the max value of extended frames

__init__(*tabWidget*)

addPacket (*valueList*, *addAtFront=True*, *append=True*, *emit=True*, *addToRawDataOnly=False*)
Override the parents class method to add packets at front and to update the counter label

clear (*returnOldPackets=False*)
Clear the currently displayed data on the GUI and in the rawData list

Parameters **returnOldPackets** – If this is true, then the previously displayed packets will be returned as raw data list

Returns Previously displayed packets as raw data list (if returnOldPackets is True), else an empty list

dataMask = None
The data is 16 chars max.

dataMaskChanged ()
This allows changing the data mask values on the fly because a new value will only be set if the new value is valid.

dataMaxLength = None
This length corresponds the length when interpreted as bytes

fuzzSenderThread = None
Sending takes place in a loop in a separate thread

fuzzingModeChanged ()
This gets called if the ComboBox gets changed to update the active fuzzing mode. The other GUI elements will be set and enabled depending on the selected mode.

fuzzingModeComboBoxValuePairs = None
These values will be available in the fuzzing mode ComboBox

generateRandomPacket ()
This generates a random pyvit Frame using `tryBuildPacket()`

Returns Pyvit frame with random data (random ID, data length and data)

itemAdderThread = None
Adding items also takes place in a separate thread to avoid blocking the GUI thread

packetBuildErrorCount = None
Used to avoid spamming the log box when the user specified wrong parameters while sending

prepareUI ()

sliderChanged ()
This method gets called if one of the two length sliders (min. and max. value) are changed. `dataMinLength` and `dataMaxLength` will be directly updated and available to a running FuzzerThread.

toggleFuzzing ()

This starts and stops fuzzing.

- Starting: - Input values are read and validated - ItemAdderThread and FuzzSenderThread (see *FuzzSenderThread*) are started - Some GUI elements will be disabled
- Stopping: - The threads will be disabled - Disabled GUI elements will be enabled again

toggleGUIElements (*state*)
{En, Dis}able all GUI elements that are used to change fuzzer settings

Parameters **state** – Boolean value to indicate whether to enable or disable elements

toggleLoopActive()

If there is a FuzzerThread sending then the tab title will be red.

validateDataMaskInput()

Validates the user specified data mask:

- The length must be <= 16
- It must be a valid hex string
- Value will be padded to 16 chars (8 bytes)

Returns A validated data mask or None if it's not possible to validate the input

validateIDMaskInput()

Validates the user specified ID mask:

- The length must be either 3 or 8
- It must be a valid hex string
- Has to be < 0x1FFFFFFF which is the max. value for extended frames

Returns A validated ID mask or None if it's not possible to validate the input

4.10 CANalyzat0r.Globals module

Created on May 17, 2017

@author: pschmied

src.Globals.**CANData** = **None**

Instance to interact with the bus

src.Globals.**db** = **None**

Object to handle db connections

src.Globals.**knownPackets** = **{}**

Stores all known packets for the current project Key: CAN ID and data concatenated and separated with a "#" Value: Description

src.Globals.**project** = **None**

Manage the currently selected project

src.Globals.**textBrowserLogs** = **None**

Display logs in the GUI

src.Globals.**ui** = **None**

The general UI

4.11 CANalyzat0r.ItemAdderThread module

Created on May 18, 2017

@author: pschmied

class `src.ItemAdderThread.`**`ItemAdderThread`**(*receivePipe*, *tableModel*, *rawData*, *useTimestamp=True*)

Bases: `PySide.QtCore.QThread`

This thread receives data from a process and emits a signal which causes the main thread to add the packets to the table.

`__init__`(*receivePipe*, *tableModel*, *rawData*, *useTimestamp=True*)

`appendRow` **= <PySide.QtCore.Signal object>**

Emit a signal to the main thread when items are ready to be added Parameters: valueList

`disable`()

This sets the enabled flag to False which causes the infinite loop in `run()` to exit.

`frameToRow`(*frame*)

Converts a pyvit frame to a raw value list. This list will be emitted using the signal `appendRow` along with the table data and rawData list.

Parameters **`frame`** – Pyvit CAN frame

`run`()

As long as the thread is enabled: Receive a frame from the pipe and pass it to `frameToRow()`.

`staticMetaObject` **= <PySide.QtCore.QMetaObject object>**

4.12 CANalyzat0r.KnownPacket module

Created on May 22, 2017

@author: pschmied

class `src.KnownPacket.`**`KnownPacket`**(*id*, *projectID*, *CANID*, *data*, *description*)

This class is being used to handle known packet data. It's more comfortable to use a object to pass data than to use lists and list indexes. Please note that this costs much more performance, so please use lists if you have to deal with much data.

`__init__`(*id*, *projectID*, *CANID*, *data*, *description*)

static **`fromJSON`**(*importJSON*)

This class method creates a KnownPacket object using a JSON string.

Parameters **`importJSON`** – The JSON string containing the object data

Returns A KnownPacket object with the values set accordingly

`toComboBoxString`()

Calculate a string that will be displayed in a ComboBox

Returns String representation of a KnownPacket object

`toJSON`()

To export a KnownPacket, all data is being formatted as JSON. The internal attribute __dict__ is being used to gather the data. This data will then be formatted.

Returns The formatted JSON data of the object

4.13 CANalyzat0r.Logger module

Created on May 17, 2017

@author: pschmied

class `src.Logger.`**`LogHandler`**

Bases: `logging.Handler`

To manage different log levels and custom logging to the log box/text browser, this class is needed.

`__init__``()`

`emit``(`*record*`)`

This checks to loglevel and also logs to log box/text browser.

Parameters **`record`** – The record to log

class `src.Logger.`**`Logger`**`(`*className*`)`

Bases: `object`

This class implements a simple logger with a formatter.

`__init__``(`*className*`)`

Along with set statements, a formatter is being applied here.

Parameters **`className`** – The tag for the logger

`getLogger``()`

`minLogLevel` **= 20**

4.14 CANalyzat0r.MainTab module

Created on May 22, 2017

@author: pschmied

class `src.MainTab.`**`MainTab`**

This class handles the logic of the main tab

static **`VCANCheckboxChanged`**`()`

Clickhandler for the VCAN CheckBox which causes the SpinBox to be toggled.

static **`addApplicationStatus`**`(`*status*`)`

Add a new status to the status bar by name (ordered). If no status is present, it will display `Strings.statusBarReady`

Parameters **`status`** – The new status to add

static **`addVCANInterface`**`()`

Manually add a virtual CAN interface. This uses a syscall to `ip link`. If this call succeeds, a new CANDataInstance will be created using `createCANDataInstance()`. The detected CAN interfaces will be refreshed, too.

static **`applyGlobalInterfaceSettings`**`()`

Set the currently selected interface as the global interface. Also, the bitrate will be updated and GUI elements will be toggled. The CANData instances of all **inactive** tabs will also be set to the global interface.

static **`applyLogLevelSetting`**`()`

Set the minimum logging level to display messages for.

static **`detectCANInterfaces`**`(`*updateLabels=True*`)`

Detect CAN and VCAN interfaces available in the system. A syscall to `/sys/class/net` is being used for this. For every detected interface a new CANData instance will be created using `createCANDataInstance()`.

Also, interface labels and the global interface ComboBox will be updated.

Parameters **`updateLabels`** – Whether to update the interface labels or not

static `easterEgg`(*event*)
Nothing to see here :return: fun

static `loadKernelModules`()
Load kernel modules to interact with CAN networks (`can` and `vcan`).

`logger` **= <logging.Logger object>**
The tab specific logger

static `populateProjects`(*keepCurrentIndex=False*)
This populates the project ComboBox in the main tab.

Parameters **`keepCurrentIndex`** – If this is set to True, the previously selected index will be re-selected in the end

static `prepareUI`()

1. Setup the status bar
2. Detect CAN interfaces and preselect the VCAN CheckBox
3. Populate project ComboBoxes
4. Add the logo

static `preselectUseBitrateCheckBox`()
Preselect the VCAN CheckBox state because we can't use the bitrate along with VCAN interfaces.

static `removeApplicationStatus`(*status*)
Remove a status from the status bar. For statuses with mutiple possible values (e.g. `Sending (X Threads)` the search will be done using a substring search

Parameters **`status`** – The status to remove

Returns

static `removeVCANInterface`()
This removes the currently selected VCAN interface. This uses a syscall to `ip link`. If the removed interface was the current global interface, the global interface will become None. :return:

static `setGlobalInterfaceStatus`(*red=False*)
Sets the text of the global interface status in the status bar. If the global CANData instance is None then the text will read "None".

Parameters **`red`** – Optional; If this is set to True, the text will appear red. Else black.

static `setProject`(*wasDeleted=False*, *setNone=False*)
This sets the current project to the currently selected project in the corresponding ComboBox. Also, the status bar and project specific ComboBoxes and GUI Elements will be updated.

Parameters

- **`wasDeleted`** – This is set to True if the current selected project was deleted. This causes `Globals.project` to become None, too.
- **`wasNull`** – This is set to True, if the project has to be set to None. Default: False

static `setProjectStatus`(*projectName*, *red=False*)
Sets the text of the project status in the status bar.

Parameters

- **projectName** – The text to put as the new project name
- **red** – Optional; If this is set to True, the text will appear red. Else black.

static setupStatusBar()
Add labels to the status bar and prepare it.

statusBarActiveStatuses = []
These text will appear in the status bar

statusBarApplicationStatus = None
Statusbar labels

statusBarInterface = None

statusBarProject = None

static updateVCANButtons()
Update the text of the buttons to add and remove VCAN interfaces.

4.15 CANalyzat0r.ManagerTab module

Created on May 22, 2017

@author: pschmied

class `src.ManagerTab.`**ManagerTab**(*tabWidget*)
Bases: *src.AbstractTab.AbstractTab*

This class handles the logic of the manager tab

__init__(*tabWidget*)

addKnownPacket(*CANID=None*, *data=None*, *description=None*)
Save a known packet to the current project (and database) Default: Get values from the GUI elements. But you can also specify the values using the optional parameters.

Parameters

- **CANID** – Optional: CAN ID
- **data** – Optional: Payload data
- **description** – Optional: Description of the known packet

Returns

clear(*returnOldPackets=False*)
Clear the GUI table displaying PacketSets along with data lists.

createDump(*rawPackets=None*)
Save a new dump to the database. This creates a new PacketSet along with associated Packet objects. If only one packet is saved, the user will be asked if he wants to create a known packet entry for the just saved packet.

Parameters **rawPackets** – Optional: If this is not None, the values from `rawPackets` will be used instead of the data that is currently being displayed in the GUI table.

createProject()
Create a new project and save it to the database. This also updates the project ComboBoxes.

deleteDump()
Delete the currently selected PacketSet from the database. This also re-populates the table with the data of another dump (if existing)

deleteProject()
Delete a project along with associated data. This also updates the project ComboBoxes.

dumpsRowIDs = None
Kepps track between the association of table row <-> database id of the packet e.g. row 2 - database ID 5

editKnownPacket()
Update a known packet with new specified values.

editProject()
Update a project with new specified values.

exportProject()
Export a project as JSON string to a textfile. The `toJSON()` method is called for every object to be exported.

getDump()
Display the data of the selected PacketSet in the GUI table. This also updates `rawData`

getKnownPacketsForCurrentProject()
(Re-)Populate the dictionary `Globals.knownPackets` with up-to-date data. If no project is set, the dictionary will be cleared only.

handleCopy()
Pass the copy event to the Toolbox, but only if no data is being loaded

handlePaste()
Pass the paste event to the Toolbox, but only if no data is being loaded

importProject()
Import a project from a JSON file. The `fromJSON()` method of every class is called to re-create objects.

loadingData = None
Disallow copying while loading data

populateKnownPacketEditLineEdits()
Sets the GUI elements concerning editing known packets to the current selected known packet data.

populateKnownPackets(*keepCurrentIndex=False*)
Populate the known packet ComboBoxex (delete and edit).

Parameters **keepCurrentIndex** – Optional: Reselect a specific KnownPacket in the ComboBox

populatePacketSets(*IDtoChoose=None*)
Populate the dumps ComboBox in the dumps tab. Reloading dump data is being handled by the triggered event

Parameters **IDtoChoose** – Optional: Preselect a specific PacketSet in the ComboBox

populateProjectEditLineEdits()
Sets the GUI elements concerning editing projects to the current selected project data.

populateProjects(*keepCurrentIndex=False*)
Populate the project ComboBoxes (delete, Edit, export project).

Parameters **keepCurrentIndex** – If this is set to True, the previously selected index will be reselected

prepareUI()
Prepare the tab specific GUI elements, add keyboard shortcuts and set a CANData instance

removeKnownPacket()
Remove a known packet from the current project (and the database)

removeSelectedPackets()
Pass the remove requested event to the super class. After that, add the **database** IDs of the deleted packets to `dumpsDeletedPacketIDs`. The deleted rows will be removed from `dumpsRowIDs`, too.

saveToFile()
Save the packets in the GUI table to a file in SocketCAN format.

updateDump()
Users can change the data displayed in the GUI table. This method allows the changed data to be saved to the database.

4.16 CANalyzat0r.Packet module

Created on May 19, 2017

@author: pschmied

class `src.Packet.`**Packet**(*packetSetID*, *CANID*, *data*, *timestamp=''*, *iface=''*, *length=None*, *id=None*)
This class is being used to handle packet data. It's more comfortable to use a object to pass data than to use lists and list indexes. Please note that this costs much more performance, so please use lists if you have to deal with much data.

__init__(*packetSetID*, *CANID*, *data*, *timestamp=''*, *iface=''*, *length=None*, *id=None*)
The parameters `CANID` and `data` must be valid hex strings. If length is not specified, it will be calculated automatically.

static fromJSON(*importJSON*)
This class method creates a Packet object using a JSON string.

Parameters **importJSON** – The JSON string containing the object data

Returns A packet object with the values set accordingly

static getDisplayDataLength(*CANID*, *hexData*)
This makes sure that the displayed length is correct. If CANID is empty, the length will also be empty. If the length if hexData is odd, the length will read "INVALID" which prevents saving to the database. Else the length will be the amount of chars / 2

Parameters

- **CANID** – CAN ID
- **hexData** – Payload data as hex string

Returns The correct length as string

lengthStringToInt(*string*)
This makes sure that the specified length is an int :return: The length as integer (if possible) - else None by exception) :raises ValueError if the length isn't an integer

toJSON()
To export a Packet, all data is being formatted as JSON. The internal attribute __dict__ is being used to gather the data. This data will then be formatted.

Returns The formatted JSON data of the object

4.17 CANalyzat0r.PacketsDialog module

Created on Jun 23, 2017

@author: pschmied

class `src.PacketsDialog.`**`PacketsDialog`**(*packets=None*, *rawPacketList=None*, *returnPacketsAsRawList=True*)

Bases: *`src.AbstractTab.AbstractTab`*

This class handles the logic of the "manage packets" dialog. For example, this can be found in the *`SnifferTabElement`*.

`__init__`(*packets=None*, *rawPacketList=None*, *returnPacketsAsRawList=True*)

This basically just sets data and reads the widget from the `.ui` file.

Parameters

- **`packets`** – Optional: List that contains the elements that will be pre loaded into the GUI table in the following format: `<CAN ID>#<Data>`. This is used for the *`SnifferTabElement`*
- **`rawPacketList`** – Optional: Raw packet list that contains the elements that will be pre loaded into the GUI table. If this is specified, `packets` will be ignored.
- **`returnPacketsAsRawList`** – Boolean value indicating whether the displayed packets will be returned as raw packet list. If this is False, the values will be returned as list in the following format: `<CAN ID>#<Data>`.

`displayUniquePackets`(*IDOnly=False*)

Filter the currently displayed data for unique packets and display them on the table. :param IDOnly: If this is True, only the ID will be matched to compare data. This allows wildcard ignores.

`getUniqueIDs`()

Filters all unique IDs out of `rawData` and displays them on the GUI table. Unique ID means that the data column will be ignored and left blank for wildcard ignores. This uses *`displayUniquePackets()`*.

`getUniquePackets`()

Filters all unique packets out of `rawData` and displays them on the GUI table. This uses *`displayUniquePackets()`*.

static **`getUniqueRawPackets`**(*rawPacketList*)

Helper method to extract unique raw packets out of a given raw packet list which has been cleaned before (Only necessary data fields are present, all others are set to an empty string)

Parameters **`rawPacketList`** – The cleaned list of raw packets

Returns A list of unique raw packet lists

`open`()

Show the widget, extract data and return it

Returns Raw packet list if the user didn't press Cancel and if `returnPacketsAsRawList` is True. Else: list of values of the following form: `<CAN ID>#<Data>` if the user didn't press Cancel. Else None.

`prepareUI`()

Prepare the GUI elements and add keyboard shortcuts. Also, pre populate the table

4.18 CANalyzat0r.PacketSet module

Created on May 19, 2017

@author: pschmied

class `src.PacketSet.`**`PacketSet`**(*id*, *projectID*, *name*, *date=None*)

This class is being used to handle packet set data. It's more comfortable to use a object to pass data than to use lists and list indexes. Please note that this costs much more performance, so please use lists if you have to deal with much data.

`__init__`(*id*, *projectID*, *name*, *date=None*)

The date of a PacketSet will be automatically set to the current date as string

static **`fromJSON`**(*importJSON*)

This class method creates a PacketSet object using a JSON string.

Parameters **`importJSON`** – The JSON string containing the object data

Returns A PacketSet object with the values set accordingly

`toComboBoxString`()

Calculate a string that will be displayed in a ComboBox

Returns String representation of a PacketSet object

`toJSON`()

To export a PacketSet, all data is being formatted as JSON. The internal attribute __dict__ is being used to gather the data. This data will then be formatted.

Returns The formatted JSON data of the object

4.19 CANalyzat0r.PacketTableModel module

Created on May 31, 2017

@author: pschmied

class `src.PacketTableModel.`**`PacketTableModel`**(*parent*, *dataList*, *header*, *readOnlyCols*, *IDColIndex=1*, *dataColIndex=2*, *lengthColIndex=3*, *timestampColIndex=4*, *descriptionColIndex=5*, **args*)

Bases: `PySide.QtCore.QAbstractTableModel`, `PySide.QtCore.QObject`

A custom TableModel is needed to allow efficient handling of **many** values.

`__init__`(*parent*, *dataList*, *header*, *readOnlyCols*, *IDColIndex=1*, *dataColIndex=2*, *lengthColIndex=3*, *timestampColIndex=4*, *descriptionColIndex=5*, **args*)

`appendRow`(*dataList=[]*, *addAtFront=False*, *emit=True*, *resolveDescription=False*)

Inserts the `dataList` list into `self.dataList` to add a whole row with values at once.

Parameters

- **`dataList`** – The list containing data. The length must be equal to *`rowCount()`*.
- **`addAtFront`** – Values will be added to the front of `self.dataList` if this is True. Else: They will be appended at the end

- **`emit`** – Optional: If the GUI will be notified of the data change or not. This is used for batch imports where the GUI isn't notified after each row to increase speed Default: True (Emit everytime)
- **`resolveDescription`** – If this is set to True, the description of the potential known packet will be resolved. Default: False

Returns The description of the known packet. If `resolveDescription` is False, an empty string is returned. Else None.

This also emits the `dataChanged` and `layoutChanged` signals to let the GUI know that the data/layout has been changed.

`appendRows`(*rowList*, *addAtFront=False*, *resolveDescriptions=True*)

This allows appending a whole set of rows at once using the best possible speed

Parameters

- **`rowList`** – List of raw data lists to append
- **`addAtFront`** – Values will be added to the front of `self.dataList` if this is True. Else: They will be appended at the end
- **`resolveDescriptions`** – If this is set to true, the description for every packet will be resolved. Default: True

Returns If `resolveDescriptions` is True, a list of known packet descriptions will be returned. If no description for a particular packet can be resolved, an empty string will be inserted in the list to keep indexes. Else None will be returned

`cellChanged` **= <PySide.QtCore.Signal object>**

Emits rowIndex and columnIndex of the changed cell

`clear`()

Clears all managed data from the `dataList`. This is a shortcut to *`setRowCount()`* with parameter `0`.

`columnCount`(*parent=None*)

Returns the current column count by returning the length of the header list.

Parameters **`parent`** – Dummy parameter to keep the needed signature

Returns The column count as integer

`data`(*index*, *role*)

Return managed data depending on the `role` parameter.

Parameters

- **`index`** – Index object containing row and column index
- **`role`** – The display role that requests data

Returns

- If the index is invalid: None
- `AlignCenter` if `role = TextAlignmentRole`
- Column data if `role = DisplayRole` or `EditRole`

`flags`(*index*)

Return the flags for cell at a given index.

Parameters **`index`** – Index object containing row and column index

Returns A flags object containing whether an object is editable, selectable or enabled

getValue(*rowIndex*, *colIndex*)
Get the data from the table at the given indexes.

Parameters

- **rowIndex** – Row index
- **colIndex** – Column index

Returns The data at the specified index (if possible); Else None

headerData(*headerIndex*, *orientation*, *role*)
Returns the header data to properly display the managed data on the GUI.

Parameters

- **headerIndex** – Which column of the data is requested
- **orientation** – This can be either `Horizontal` or `Vertical`:
  - `Horizontal`: Return a value from `self.header`
  - `Vertical`: Return the `headerIndex`
- **role** – This is always expected to be `DisplayRole`

Returns See `orientation`. None is returned if `orientation` or `role` do not match

insertRow(*dataList=[]*)
This is just an alias to `appendRow()` for compatibility.

Parameters **dataList** – A list that stores the data that will be added

removeRow(*rowIndex*)
Removes the specified row from the table model. This also emits the `layoutChanged` signal to let the GUI know that the layout has been changed.

Parameters **rowIndex** – The row index to delete

removeRows(*rowIndexes*)
Remove multiple rows at once.

Parameters **rowIndexes** – The rows that will be deleted

rowCount(*parent=None*)
Returns the current row count by returning the length of the data list.

Parameters **parent** – Dummy parameter to keep the needed signature

Returns The row count as integer

setData(*index*, *value*, *role=PySide.QtCore.Qt.ItemDataRole.EditRole*)
This gets called to change the element on the GUI at the given indexes This also emits the `layoutChanged` signal to let the GUI know that the layout has been changed.

Parameters

- **index** – Index object containing row and column index
- **value** – The new value
- **role** – Optional: The role calling this method. Default: EditRole

Returns True if the operation succeeded

setRowCount(*count*)
Sets the row count by removing lines / adding empty lines. This also emits the `layoutChanged` signal to let the GUI know that the layout has been changed.

Parameters **`count`** – The desired amount of rows

`setText` (*rowIndex*, *colIndex*, *data*)

Sets the text of at the given indexes. If `data` is None the text will be an empty string. This method also emits the `dataChanged` and `layoutChanged` signals to let the GUI know that the data/layout has been changed.

Parameters

- **`rowIndex`** – Row index
- **`colIndex`** – Column index
- **`data`** – New data for the column

`sort` (*colIndex*, *order*)

Sort the data by given column number. This also emits the `layoutChanged` signal to let the GUI know that the layout has been changed.

Parameters

- **`colIndex`** – The column index to sort for
- **`order`** – Either `DescendingOrder` or `AscendingOrder`

`staticMetaObject` **= <PySide.QtCore.QMetaObject object>**

4.20 CANalyzat0r.Project module

Created on May 19, 2017

@author: pschmied

class `src.Project.`**`Project`** (*id*, *name*, *description*)

This class is being used to handle project data. It's more comfortable to use a object to pass data than to use lists and list indexes. Please note that this costs much more performance, so please use lists if you have to deal with much data.

`__init__` (*id*, *name*, *description*)

This just sets the attributes to the passed parameters.

static **`fromJSON`** (*importJSON*)

This class method creates a project object using a JSON string.

Parameters **`importJSON`** – The JSON string containing the object data

Returns A project object with the values set accordingly

`toComboBoxString` ()

Calculate a string which will be displayed in the ComboBoxes.

Returns String representation of the object

`toJSON` ()

To export a Project, all data is being formatted as JSON. The internal attribute __dict__ is being used to gather the data. This data will then be formatted.

Returns The formatted JSON data of the object

4.21 CANalyzat0r.SearcherTab module

Created on May 17, 2017

@author: pschmied

class `src.SearcherTab.`**`SearcherTab`**`(`*tabWidget*`)`
Bases: *`src.AbstractTab.AbstractTab`*

This class handles the logic of the filter tab

`__init__``(`*tabWidget*`)`

`askActionPerformed``()`
Ask the user if the action has been performed using a MessageBox

Returns True if the user pressed yes, else False

`askWhichAction``()`

Ask the user what to do if no chunk worked:

- Try again
- Re-test the current last working chunk
- Cancel

Returns An integer value indicating the pressed button: - 0 if the user wants to try again - 1 if the user wants to re-test - 2 if the user wants to cancel

`beep``()`
To play a sound after sending has been finished.

`clear``(`*returnOldPackets=False*`)`
Clear the GUI table and all associated data lists

`downwardsSearch` **= None**
We first search downwards in the binary search tree. If this doesn't succeed, we use randomization and begint to search upwards.

`enterWhenReady``()`
Block the GUI thread until the user pressed the button on the MessageBox.

`lastWorkingChunk` **= None**
The currently smallest known set of packets that cause a specific action. Values are lists with raw data

`outputRemainingPacket``(`*packet*`)`
Show the passed packet on the GUI table.

Parameters **`packet`** – The raw packet to display

`outputRemainingPackets``(`*rawPackets*`)`
Show all passed packets on the GUI table. Note: This also sets `rawData` to the passed set of packets.

Parameters **`packet`** – List of raw packets to display

`searchPackets``()`

This starts the whole searching routine and sets up things first:

1. Set a CANData instance
2. Get user input values

3. Walk down the binary search tree: Try to find a specific packet for an action
4. If 1 packet has been found: output the packet
5. If not: Get the last working chunk of packets that worked. Use shuffling and new values for the chunk amount to find a minimal set of packets

sendAndSearch(*chunkAmount=2*)

Use the remaining data to search for relevant packets:

1. Setup a progress bar
2. Split the raw packet list in the desired amount of chunks
3. Test each chunk (Newest packets first) and ask the user if it worked
4. If it worked: Set `lastWorkingChunk` to the last tested chunk and return True
5. Else: Return False if all other chunks failed too.

Parameters **chunkAmount** – The amount of chunks to generate from the given data list

Returns True if a specific chunk worked, else False

splitLists(*lst*, *chunkAmount=2*)
Split a list into a specific amount of chunks

Parameters

- **lst** – The list to split
- **chunkAmount** – Desired amount of chunks

Returns List of chunks (List of lists in this case)

toggleGUIElements(*state*)
{En, Dis}able all GUI elements that are used to change searcher settings

Parameters **state** – Boolean value to indicate whether to enable or disable elements

4.22 CANalyzat0r.SenderTab module

Created on May 22, 2017

@author: pschmied

class `src.SenderTab.`**SenderTab**
This class handles the logic of the sender tab. Subtabs are being handled in *SenderTabElement*.

CANData = None
The tab specific CANData instance

active = False

static addSender(*senderTabName=None*)
Appends a new sender tab to the sub tab bar.

Parameters **senderTabName** – Optional; The displayed name of the tab. If this is None, the user is requested to enter a name

static addSenderWithData (*listOfRawPackets=None*, *listOfPackets=None*)
: Uses `addSender()` to add a new sender tab with data already filled in into the GUI table. You must specify `listOfRawPackets` **or** `listOfPackets`. If both are specified, `listOfRawPackets` will be used.

  Parameters

  - **`listOfRawPackets`** – Optional; List of raw packets to add to the table.
  - **`listOfPackets`** – Optional; List of packet objects to add to the table.

  Returns

`currentlySendingTabs` = 0
: Used to handle the font color of the sender tab

classmethod handleInterfaceSettingsDialog ()
: This invokes `handleInterfaceSettingsDialog()` for the class

`indexInMainTabBar` = 2
: The index of the sender tab in the main tab bar

`labelInterfaceValue` = None

`logger` = <logging.Logger object>
: The tab specific logger

static prepareUI ()
: Prepare the tab specific GUI elements, add sender tab and keyboard shortcuts. Also set a CANData instance.

static removeSender (*senderTabElement*)
: Remove a sender from the sub tab bar. This method gets called from an instance of `SenderTabElement` by `removeSender()`.

  Parameters **`senderTabElement`** – The `SenderTabElement` instance to remove

static sendSinglePacket ()
: Sends a single packet using the specified interface. All packet values are read from the GUI elements.

`senderTabs` = []
: Consinsts of all SenderTabElements

classmethod setInitialCANData ()
: This invokes `setInitialCANData()` for the class

static toggleActive ()
: If there is at least one tab sending then the tab bar title will be red.

static toggleGUIElements (*state*)
: {En, Dis}able all GUI elements that are used to change filter settings

  Parameters **`state`** – Boolean value to indicate whether to enable or disable elements

classmethod updateCANDataInstance (*CANDataInstance*, *delegate=False*)
: This invokes `updateCANDataInstance()` for the class

  Parameters

  - **`CANDataInstance`** – The new CANData instance
  - **`delegate`** – Boolean indicating if all sender sub tabs will be updated too. Default: False

classmethod updateInterfaceLabel ()
: This invokes `updateInterfaceLabel()` for the class

4.23 CANalyzat0r.SenderTabElement module

Created on May 23, 2017

@author: pschmied

class `src.SenderTabElement.`**`SenderTabElement`**(*tabWidget*, *tabName*)

Bases: *`src.AbstractTab.AbstractTab`*

This class handles the logic of the sender sub tab. The main tab is being handled in *`SenderTab`*.

`__init__`(*tabWidget*, *tabName*)

Set all passed data. Also, add the own send button to `SenderTab.sendButtonList` to allow managing it globally.

Parameters **`tabWidget`** – The element in the tab bar. **Not** the table widget.

`amountThreadsRunning` **= 0**

Amount of sending threads running to display in the status bar

`getTabIndex`()

Get the **current** tab index of the sub tab element

Returns The tab index of the sender tab

`loopSenderThread` **= None**

The thread that runs when sending takes place in a loop

`prepareUI`()

Prepare the tab specific GUI elements, add keyboard shortcuts and set a CANData instance

`removeSender`()

This gets called when the remove sender button is pressed on the sub tab. This stops the sender thread and calls the parents method (*`removeSender()`*) to remove the sender form the tab bar.

`sendAll`()

Send all packets in the GUI table. By default, this just sends the packet once using simple calls. If the user requests to send the packets in a loop, an instance of *`LoopSenderThread`* is being used to send the packets. Also, GUI elements like the status bar are being updated.

`sendButtonList` **= []**

Static attribute of send buttons to manage {en, dis}abled states

`setSendButtonState`(*state*)

This sets the enabled state of the send button.

Parameters **`state`** – The desired enabled state as boolean value

`stopSending`()

This stops the currently running instance of *`LoopSenderThread`* from sending. Also, GUI elements like the status bar are being updated.

`toggleGUIElements`(*state*)

{En, Dis}able all GUI elements that are used to change filter settings

Parameters **`state`** – Boolean value to indicate whether to enable or disable elements

`toggleLoopActive`()

Toggles the current sub tab to (in)active. This also calls *`toggleActive()`* to manage the color of the main tab (parent tab).

`updateStatusBar`()

Updates the status bar label to display the correct amount of sending tabs (if any)

4.24 CANalyzat0r.SenderThread module

Created on Jun 08, 2017

@author: pschmied

class `src.SenderThread.`**`FuzzSenderThread`**(*sleepTime*, *fuzzerSendPipe*, *CANData*, *threadName*)

Bases: `PySide.QtCore.QThread`

Spawns a new thread that will send random data in a loop.

`__init__`(*sleepTime*, *fuzzerSendPipe*, *CANData*, *threadName*)

`disable`()

This sets the `enabled` flag to False which causes the main loop to terminate.

`run`()

Send the packets in a loop and wait accordingly until the thread is disabled.

`staticMetaObject` = <PySide.QtCore.QMetaObject object>

class `src.SenderThread.`**`LoopSenderThread`**(*packets*, *sleepTime*, *CANData*, *threadName*)

Bases: `PySide.QtCore.QThread`

Spawns a new thread that will send the passed packets in a loop.

`__init__`(*packets*, *sleepTime*, *CANData*, *threadName*)

`disable`()

This sets the `enabled` flag to False which causes the main loop to terminate.

`run`()

Send the packets in a loop and wait accordingly until the thread is disabled.

`staticMetaObject` = <PySide.QtCore.QMetaObject object>

4.25 CANalyzat0r.Settings module

Created on May 17, 2017

@author: pschmied

`src.Settings.`**`APP_NAME`** **= ‘CANalyzat0r’**

The application name

`src.Settings.`**`APP_VERSION`** **= ‘1.0’**

The application version

`src.Settings.`**`DB_NAME`** **= ‘../data/database.db’**

The relative path of the SQLite database file

`src.Settings.`**`DB_PATH`** **= ‘../data/database.db’**

Optionally we can specify a different database path

`src.Settings.`**`FORKME_PATH`** **= ‘./ui/icon/forkme.png’**

Where to find the “Fork Me” icon

`src.Settings.`**`GITHUB_URL`** **= ‘https://github.com/SCHUTZWERK-CANalyzat0r/CANalyzat0r’**

Where to find this project in GitHub

`src.Settings.`**`ICON_PATH`** **= ‘./ui/icon/icon.png’**

Where to find the app icon

`src.Settings.`**`LOGO_PATH`** **= './ui/icon/swlogo_small.png'**
Where to find the company logo

4.26 CANalyzat0r.SnifferProcess module

Created on May 18, 2017

@author: pschmied

class `src.SnifferProcess.`**`SnifferProcess`**(*snifferSendPipe*, *sharedEnabledFlag*, *snifferName*, *CANData=None*)
Bases: `multiprocessing.process.Process`

Spawn a new process that will sniff packets from the specified CANData instance. Captured data will be transmitted via the `snifferSendPipe`.

`__init__`(*snifferSendPipe*, *sharedEnabledFlag*, *snifferName*, *CANData=None*)
Set the passed parameters.

Parameters

- **`snifferSendPipe`** – The multiprocessing pipe to send received data to
- **`sharedEnabledFlag`** – The multiprocessing value to handle disabling
- **`snifferName`** – The name of the sniffer process, used for logging
- **`CANData`** – Optional: The CANData instance to query for data. If this is not specified, the global interface is being used

`run`()
As long as the process hasn't been disabled: Read a frame using *`readPacketAsync()`* and transmit the received pyvit frame via the pipe.

4.27 CANalyzat0r.SnifferTab module

Created on May 18, 2017

@author: pschmied

class `src.SnifferTab.`**`SnifferTab`**
This class handles the logic of the sniffer tab. Subtabs are being handled in *`SnifferTabElement`*.

static **`addSniffer`**(*snifferTabName*)
Appends a new sniffer tab to the sub tab bar.

Parameters **`snifferTabName`** – The displayed name of the tab. Normally, this corresponds to the CAN interface the tab is managing

static **`clearAndAddPlaceholder`**()
Add a placeholder where normally sniffer tabs are displayed

`indexInMainTabBar` **= 1**
The index of the sniffer tab in the main tab bar

`logger` **= <logging.Logger object>**
The tab specific logger

static **`prepareUI`**()
This adds a placeholder of no instance of *`SnifferTabElement`* was created previously.

static removeSniffer(*snifferTabElement=None*, *snifferTabName=None*)

Remove a sniffer from the sub tab bar. This method gets called from an instance of `SnifferTabElement` by `removeSender()`. One can either specify `snifferTabElement` *or* `snifferTabName` to delete a tab. If both are used, the object parameter is used.

Parameters

- **senderTabElement** – Optional: The `SnifferTabElement` instance to remove
- **snifferTabName** – Optional: The name of the `SenderTabElement` instance to remove

snifferTabs = {}

Consinsts of all SnifferTabElements, interface names as key

static toggleActive()

If there is at least one tab sniffing then the tab bar title will be red.

classmethod updateInterfaceLabel()

This invokes `updateInterfaceLabel()` every sniffer tab

4.28 CANalyzat0r.SnifferTabElement module

Created on Jun 16, 2017

@author: pschmied

class `src.SnifferTabElement.`**SnifferTabElement**(*tabWidget*, *tabName*, *ifaceName=None*)

Bases: `src.AbstractTab.AbstractTab`

This class handles the logic of the sniffer sub tab. The main tab is being handled in `SnifferTab`.

__init__(*tabWidget*, *tabName*, *ifaceName=None*)

Set parameters and initialize the CANData instance

Parameters

- **tabWidget** – The element in the tab bar. **Not** the table widget.
- **tabName** – The name of the tab
- **ifaceName** – Optional: The interface name. If this is None, then the `tabName` will be used

addPacket(*valueList*, *addAtFront=True*, *append=True*, *emit=True*, *addToRawDataOnly=False*, *ignorePacketRate=False*)

Override the parents method to add packets at front and to update the counter label. If too much data is received, the data will be added after sniffing to prevent freezes. Also, only add packets if the data isn't present in `self.ignoredPackets`

Parameters **ignorePacketRate** – Additional optional parameter: Boolean value indicating whether the rate of packets per second will be ignored or not. This is set to False by Default. We need to set it to True to process `self.valueList` after sniffing if too much data was received.

amountThreadsRunning = 0

Amount of sniffing threads running to display in the status bar

clear(*returnOldPackets=False*)

Clear the currently displayed data on the GUI and in the lists.

Parameters **`returnOldPackets`** – Optional: If this is True then the previously displayed data will be returned as raw data list. Default is False

`getPacketCount``()`
This uses a call to `/sys/class/net/<ifaceName>/statistics/rx_packets` to return the number of total received packets of the current interface

Returns Received packet count of the current interface as itneger

`handleInterfaceSettingsDialog``(`*allowOnlyOwnInterface=True*`)`
Override the parents method to only allow the currently set CAN interface

`handleManageIgnoredPacketsDialog``()`
Open a dialog to manage ignored packets when sniffing

`removeSniffer``()`
This gets called when associated interface disappears after a re-check. This stops the sniffer thread and calls the parents method (`removeSender()`) to remove the sniffer form the tab bar.

`tabIndex``()`
Get the **current** tab index of the sub tab element

Returns The tab index of the sniffer tab

`terminateThreads``()`
This stops the processes/threads called `snifferProcess` and `itemAdderThread`. Also, the CAN-Data instance will be set to inactive and GUI elements will be toggled.

`toggleActive``()`
Toggles the current sub tab to (in)active. This also calls *`toggleActive()`* to manage the color of the main tab (parent tab).

`toggleSniffing``()`
This starts and stops sniffing for a specific sniffer tab. Instances of *`ItemAdderThread`* and *`SnifferProcess`* are used to gather and display data.

`updateStatusBar``()`
Updates the status bar label to display the correct amount of sniffer tabs (if any)

4.29 CANalyzat0r.Strings module

Created on May 17, 2017

@author: pschmied

4.30 CANalyzat0r.Toolbox module

Created on May 22, 2017

@author: pschmied

class `src.Toolbox.`**`Toolbox`**
This calls offers helpful static methods that every tab can use to unify program logic and simplify code.

static **`askUserConfirmAction`**`()`
Spawns a MessageBox that asks the user to confirm an action

Returns True if the user has clicked on Yes, else False

static **`checkProjectIsNone`**(*project=-1*)

Checks if a project is `None` and displays a MessageBox if True.

Parameters **`project`** – Optional. The project to check. Default: -1 wich causes the global project to be checked

Returns Boolean value indicating whether the checked project is `None`

static **`getKnownPacketDescription`**(*CANID*, *data*)

Get a description for a known packet. This will use the dictionary defined in `Globals` to find data

Parameters

- **`CANID`** – CAN ID
- **`data`** – Data

Returns The description if one can be found, else an empty string

static **`getPacketDictIndex`**(*CANID*, *data*)

Calculates the index of a packet with a specific CAN ID and data in a dictionary.

Parameters

- **`CANID`** – CAN ID
- **`data`** – Data

Returns The index of a packet in a dictionary

static **`getSaveFileName`**(*dialogTitle*)

Spawns a "save file dialog" to get a file path to save to.

Parameters **`dialogTitle`** – The displayed dialog title

Returns The file path of the selected file

static **`getWorkingDialog`**(*text*)

Generates a working dialog object which blocks the UI.

Parameters **`text`** – Text to display while working

Returns The created working dialog widget

`interfaceDialogWidget` **= None**

The dialog widget to set the interface settings

static **`interfaceSettingsDialog`**(*currentCANData*, *CANDataOverrideValues=None*)

Handles the logic of the interface settings dialog.

Parameters **`CANDataOverrideValues`** – Optional: List of CANData instances that will be selectable instead of all values.

Returns A new CANData instances with the selected values. None if no editable CANData instance exists

static **`interfaceSettingsDialogCheckBoxChanged`**(*state*)

Gets called when the use VCAN CheckBox of the interface dialog gets changed to handle the enabled state of the bitrate SpinBox

Parameters **`state`** – Not used, state is determined by `isChecked`

static **`interfaceSettingsDialogComboBoxChanged`**()

Gets called when the interface ComboBox of the interface dialog gets changed to pre-populate the GUI elements accordingly.

static `isHexString` (*hexString*)
: Checks if a hexString is a valid hex string of base 16

  Parameters **`hexString`** – The hex string

  Returns Boolean value indicating the correctness of the hex string

`logger` **= <logging.Logger object>**
: The toolbox also has its own logger instance

`mp3Processes` **= {}**
: Used to keep track of the processes that play mp3 files. Key = filepath, value: multiprocessing Process object

static `playMP3` (*filePath*)
: Plays an mp3 sound file using ffmpeg

  Parameters **`filePath`** – Path of the mp3 file

static `populateInterfaceComboBox` (*comboBoxWidget*, *reselectCurrentItem=True*, *ignoreActiveInstances=False*)
: Inserts all available interface values into the passed ComboBox widget

  Parameters

  - **`comboBoxWidget`** – The GUI element to fill with items
  - **`reselectCurrentItem`** – Optional: If this is true, the previously selected index will be re-selected Default: True

static `stopMP3` (*filePath*)
: Stops the playback of a given mp3 file :param filePath: Path of the mp3 file

static `tableExtractAllRowData` (*table*)
: Get **all** contents of a GUI table

  Parameters **`table`** – The `QTableView` object to gather data from

  Returns A list of raw row data –> List of lists

static `tableExtractSelectedRowData` (*table*)
: Get the **selected** contents of a GUI table

  Parameters **`table`** – The `QTableView` object to gather data from

  Returns A list of raw row data –> List of lists

static `toggleDisabledProjectGUIElements` ()
: This toggles specific GUI elements that should only be active if a project has been selected

static `toggleDisabledSenderGUIElements` ()
: This toggles specific GUI elements that should only be active if a CANData instance is present

static `updateCANDataInstances` (*CANDataInstance*)
: Calls `updateCANDataInstance` for every tab.

  Parameters **`CANDataInstance`** – The new CANData instance

static `updateInterfaceLabels` ()
: Calls `updateInterfaceLabel` for every tab.

static `widgetFromUIFile` (*filePath*)
: Reads an `.ui` file and creates a new widget object from it.

  Parameters **`filePath`** – Where to find the `.ui` file

  Returns The new created widget

static **`yesNoBox`** (*title*, *text*)

Spawns a MessageBox that asks the user a Yes-No question.

Returns True if the user has clicked on Yes, else False

4.31 CANalyzat0r.mainWindow module

4.32 CANalyzat0r.AbstractTab module

Created on Jun 26, 2017

@author: pschmied

class `src.AbstractTab.`**`AbstractTab`**(*tabWidget*, *loggerName*, *readOnlyCols*, *packetTableViewName*, *labelInterfaceValueName=None*, *CANData=None*, *hideTimestampCol=True*, *sendToSenderContextMenu=True*, *saveAsPacketSetContextMenu=True*, *allowTableCopy=True*, *allowTablePaste=True*, *allowTableDelete=True*)

This is a base class for *most* tabs. If you're using a tab that uses the following things, you can use this class: - Non-static tab - you create instances from it - a QTableView to display data - `rawData` as raw packet list - Copy, paste and/or delete actions by shortcuts and context menus

Just take care of `packetTableViewName` and `labelInterfaceValueName` as they're necessary for this to work.

`CANData` = **None**

The tab specific CANData instance

`__init__`(*tabWidget*, *loggerName*, *readOnlyCols*, *packetTableViewName*, *labelInterfaceValueName=None*, *CANData=None*, *hideTimestampCol=True*, *sendToSenderContextMenu=True*, *saveAsPacketSetContextMenu=True*, *allowTableCopy=True*, *allowTablePaste=True*, *allowTableDelete=True*)

`active` = **None**

Whether the tab is currently active (using `CANData`)

`addPacket`(*valueList*, *addAtFront=False*, *append=True*, *emit=True*, *addToRawDataOnly=False*)

Add a packet to the GUI table.

Parameters

- **`valueList`** – Packet data as raw value list
- **`addAtFront`** – Optional. Indicates whether the packets will be inserted at the start of `rawData`. Default: False
- **`append`** – Optional. Indicates whether data will be added to `self.rawData` or not. Default: True
- **`emit`** – Optional. Indicates whether the GUI will be notified using signals. Default: True
- **`addToRawDataOnly`** – Optional. Indicates whether only `self.rawData` will be updated, ignoring the packet table model. Default: False

`applyNewKnownPackets`()

Apply new known packets which have been saved in the mean time. This reloads the packets into the GUI table.

`clear`(*returnOldPackets=False*)

Clear the currently displayed data on the GUI and in the rawData list

Parameters **`returnOldPackets`** – If this is true, then the previously displayed packets will be returned as raw data list

Returns Previously displayed packets as raw data list (if returnOldPackets is True), else an empty list

`handleCellChanged`(*rowIndex*, *colIndex*)
To update the rawData element and to put the length of the changed data field into the length field.

Parameters

- **`rowIndex`** – The changed row
- **`colIndex`** – The changed column

Returns

`handleCopy`()
Handle copying **selected** rows from a GUI table. This copies the raw data list **to the clipboard**.

`handleInterfaceSettingsDialog`(*allowOnlyOwnInterface=False*)
Open a dialog to change interface settings and set the updated CANData instance.

Parameters **`allowOnlyOwnInterface`** – If this is true, you can only edit the CANData instance that is already selected for the current tab. This is being used for the sniffer tabs.

`handlePaste`()
Handle pasting rows into a GUI table. Data is being gathered from the clipboard and be of the following types:

- Raw data list (list of lists which consist of column data) - Parsing takes place asynchronously
- SocketCAN format (see `SocketCANFormat`)

`handleRightCick`()
This spawns a custom context menu right next to the cursor if a user has right clicked on a table cell.

`loggerName` **= None**
The tab specific logger

`manualAddPacket`()
Manually add an empty packet row to the GUI table. This also updates `rawData`.

`packetTableModel` **= None**
Custom packet model of the GUI table

`prepareUI`()
Prepare the tab specific GUI elements, add keyboard shortcuts and set a CANData instance

`rawData` **= None**
Raw packet data that corresponds to the data displayed in the GUI table

`readOnlyCols` **= None**
Tab specific read only columns as list of indexes

`removeSelectedPackets`()
Remove selected rows from a table and also delete those rows from a `rawData` list. :return: A list of indexes of the removed rows. None if no rows have been selected

`setInitialCANData`()
Try to get initial an initial CANData instance. This method also updates GUI elements.

Returns A boolean value which indicates the success

`tabWidget` **= None**
The specific GUI tab

`toggleGUIElements``(`*state*`)`

`updateCANDataInstance``(`*CANDataInstance*`)`
Updates the tab specific CANData instance to the passed parameter. This only takes place if the tab is not active to prevent errors. This also calls *`updateInterfaceLabel()`* to update the label.

Parameters **`CANDataInstance`** – The new CANData instance

`updateInterfaceLabel``()`
Set the text of the interface label to the updated value, if the label is present. Uses the text “None” if no interface is set.

4.33 Module contents

CHAPTER

FIVE

USED LIBRARIES AND FILES

5.1 Libs

- pyvit by Eric Evenchick
- PySide: Python for Qt
- **‘ffmpeg <https://ffmpeg.org/‘_**
- can-utils
- Sphinx
- Sphinx RTD Theme

5.2 Misc

- Car Icon Flat
- Orange flame 2 icon
- Soylent red flame 2 icon

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX