Implementation Guide for TraceSniffer: Frontend

Berti Martens
Hochschule Darmstadt

October 21, 2018

Contents

1 Introduction

This guide enables users to create new tabs, access data provider by the 'Tracer'-application running on a microcontroller. The basic concepts of the Frontend are listed, as well as an example Tab. Other examples can be found in the **Docks**-Folder in the Source-Code.

2 Available Modules

TraceSniffer provides a set of Modules in order to facilitate implementing new tabs. Enabling them **requires** inheriting from TraceDocks.

- **SnifferConfig:** This module allows setting variables to a value that persist between program restarts.
  How to use: During the program, set any variables that you want to be saved like so: *self.snifferConfig.configMyFoo = myBar*. In the constructor of your Tab, call *self.snifferConfig.parseConfigFromFile()*. The variable will now be accessible again, with the old value intact.
  You can find the saved variable in *./SnifferConfig/myTab.jcfg*

- **SnifferFileParser:** This module allows you to parse a file in the format specified in [CJM] to a Global payloadList and vice-versa.
  How to use: Call *self.snifferParser.saveToFile('/path/to/sniff')* to save a file and *self.snifferParser.getFromFile('/path/to/sniff')*.
  In the future, the list will also be returned locally from the function, and a local list can be saved to a sniff-file via second parameter.

- **SnifferLogger:** This module allows you to log events and save them to a logfile automatically with a timestamp. The logging can be disabled or enabled in the configuration-tab.
  How to use: Call *self.logger.logEvent('foo')* at any point in your tab.

- **DisplayException:** This function allows you to display a messagebox to the user, informing him about an Error and the type of Error that occured. It takes the error as a string as the parameter. **How to use:** Call self.displayException('mySampleExceptionText') from any Tab inheriting from TraceDocks.

- **SnifferFilter:** This module allows you to filter your payloadList by IDs or other criteria. Since it is a QWidget, it needs a little bit more work to use.
  How to use: In order to show the Filter, call *self.snifferFilter.show()* in your Tag.
  When the user is finished filtering the list, there are two ways to handle the filtered-list:

  1. Implement the following callback-function:

  ```
  # Declaration of function must be exactly like this
    def filterUpdated(self, filteredIDs, filteredPayloads):
        print('we arrive from SnifferFilter')
        self.doFoo(filteredPayloads)
  ```

  2. Find the filtered list in *self.snifferFilter.filteredIdList* anywhere in the Tab.

- **SnifferStats:** This module allows you to show statistics about the current measurement. **How to use:** In order to show the stats, call *self.snifferStats.show()* in your Tag.

3 Available Globals

TraceSniffer provides a set of Objects in order to facilitate accessing data of the different tabs and measurements. They can be accessed via the Globals-class and should be used as **read-only**.

- **PayloadList:** This object stores a list of all payloads the latest measurement (or .sniff file) provided.
  Example access: accessing *Globals.payloadList[0]* yields the first Payload per [CJM] - format
- **ObjectDict:** This object stores a dictionary about the tracer-Objects like tasks or mutexes. It is provided in the following format:
  (ObjectType: (ObjectNo:cleartext-name,...),...)
  Example access: *Globals.objectDict['QUEUE'][3]* yields a string to your 3rd queue ('my3rdQueue')
- **DockDict:** This object stores a dictionary about all existing tabs. If you follow the sample implementation, your tab will be accessible via the DockDict aswell. Key: 'tabName' Value : tabInstance.
  Example access: If you want to access a configuration item from the configuration Tab, you can retrieve the configTab-instance like so: *myConfigInstance = Globals.dockDict['dockConfig']* and then access a specific value *selectedCOMPort = myConfigInstance.snifferConfig.configCom*

There are a number of different objects available globally. Take a peek at the source code to learn more!

4 Sample Tab

Here, you will find a sample implementation of a tab with comments.
First: Creating your TabClass - MyTab.py

```
# This tab creates a Pushbutton, which display the name of the
                                  first payload in the payloadlist
# in a LineWidget
class MyTab(TraceDocks): # inheriting from TraceDocks to access the
                                   modules
    def __init__(self,parent):
      # giving a name to the tab
        super(MyTab, self).__init__(parent,'My Tab')
        self.tabName = self.name # copying the name
        self.parent = parent
        # first use of Logger-module
        self.logger.logEvent('Creating Tab now'+ self.tabName)

        self.snifferConfig.configState = 'default' # default value
        # override the value from configFile
        self.snifferConfig.parseConfigFromFile()

   def setMyTabLayout(self):
        # Create My Tab --------------------###
        # Create Layouts
        self.Vlayout = QVBoxLayout()

        self.myPb = QPushButton('Click for payload')
        self.myPb.clicked.connect(self.displayObj)

        self.myLineWidget = QLineEdit()
        self.Vlayout.addWidget(self.myLineWidget)

        # Set the container contents to our layout
        self.dockContents.setLayout(self.Vlayout)

   def displayObj():
       # Get the first object from the payload
       payload = Globals.payloadList[0]
       # Get the numerical ID
       payloadID = payload.payloadHead.informationID
       # Convert to cleartext-name
       payloadName = Globals.tspDict[payloadID]
       # Display the name
       self.myLineWidget.setText(payloadName)
       self.snifferConfig.configState = 'displayed something'
```

Second: Instantiating the tab and adding it to the dockDict - TracesnifferFrontend.py

```
        self.dockMine = MyTab(self)
        self.dockMine.setMyTabLayout()
        Globals.dockInstanceList.append(self.dockMine)
        Globals.dockDict['dockMine'] = self.dockMine
```

The created class inherits from TraceDocks, which makes all the mentioned modules in section 3 available to use. Use the self.snifferConfig instance so access persistent variables that need to be saved. Your tab needs a layout in order to display your data to the user. The SnifferAPI provides functions in order to interact with the payloadData. The *displayObj()*-function in the listing above demonstrates how you can access the informationID via the tspDict, which converts numeric IDs to human-readable string format as defined per [CJM]
After creating the tab-Class with your desired contents, you need to instantiace the tab in the TraceSnifferFrontend.py file. First, create the instance and set the Layout. Then, append it to the various lists / dictionaries, so your tab can be accessed from other tabs and its configuration items can be saved by the configParser.

5 Guideline for your tabs

You can create any tab you need with your desired functions, in any way you see fit. However, we recommend following the following requirements in order for the tab to be well integrated into the Sniffer-Framework.

(...) stands for 'The tab'.

- (...) SHOULD be implemented in its own separate File (example: ConfigTab.py → class ConfigTab)
- (...) MUST inherit from TraceDocks
- (...) MUST set a name to the attribute *self.tabName*
- (...) SHOULD use the the provided Logger via *self.logger.logEvent(stringOfEvent*
- (...) MUST use a local copy of the global PayloadList if the PayloadList needs to be modified
- (...) MAY use the global PayloadList provided by *Global.payloadList* for **Read-only** purposes.
- (...) MUST be instantiated by the TraceSnifferMain class and added to the *Globals.dockDict* instance with Key: Tab-Name Value: Tab-instance
- (...) SHOULD **either** use provided configuration values by dockConfig **or** create its own configuration via *self.snifferConfig* instance.

References

[CJM] *TraceSniffer Protocol V1.0*, Carl-Jonas Mair