BRAIN PRODUCTS
Solutions for neurophysiological research

BrainVision PyCorder
User Manual

Software Version 1.0.8*

valid as of September 29, 2014*

BRAINVISION
engineering PyCorder

* Valid until publication of a new version of the manual

Blank page

Manual version 006

Imprint

Blank page

Contents

List of figures

List of tables

Important information

Contact

Brain Products GmbH
Zeppelinstraße 7
D-82205 Gilching (Munich)
Phone: +49 8105 73384 – 0
Fax: +49 8105 73384 – 505
Web site: http://www.brainproducts.com

Support forum

PyCorder is an Open Source software that is provided by Brain Products free of charge. It contains all the basic functionalities to perform EEG recordings with *actiCHamp*. For support and questions about *PyCorder*, please contact your local dealer.

Unlike our licensed software, *PyCorder* is not continuously developed. Software fixes will be made available as updates. With the relevant Python programming skills you can add or adapt functionalities to your needs. If you find any errors in the software, we would like to ask you to report them to us through your dealer or the support forum: http://www.acti-champ.com/forum/.

This platform also allows you to communicate with other users, publish your own program extensions or ask questions about *PyCorder*. Please always contact your local dealer first, since they can usually offer quick solutions.

Besides *PyCorder* we offer our standard software *BrainVision Recorder*, which is constantly developed and updated. Brain Products has installed a free personal support for *Recorder* by email. If *PyCorder* is not the preferred solution for your needs, please contact your local dealer for a trial version of *Recorder*.

License

About this manual

The *PyCorder User Manual* will teach you:

- how to operate *PyCorder*
- how to create your own modules
- how to edit the basic modules.

The manual provides you with an overview of the basic functions of *PyCorder*, its GUI and its tools. In addition, the manual will teach you all that you need to know in order to create your own modules for *PyCorder* quickly and simply. A range of programming examples will help you to find your way round in a simple way. The source code for our examples is available on the supplied *Application Suite* DVD.

Remember that this manual is not a general guide to programming in Python. On the contrary, it assumes that you already have advanced Python programming skills.

The structure of the manual

The *PyCorder* User Manual is divided into four chapters:

- Chapter 1 describes the installation of all the software components that you need to run *PyCorder* and provides you with notes on configuring the Windows® operating system for recording data.
- Chapter 2 deals with the functions of the *PyCorder* and how it is operated.
- In Chapter 3, a number of examples will show you how to program your own modules for *PyCorder*.
- Chapter 4 offers assistance with troubleshooting.

Who is the manual intended for?

The *PyCorder User Manual* is aimed at users from the fields of psychophysiological and neurological research who have advanced Python programming skills.

Conventions used in the manual

The manual uses the following typographical conventions:

Bold	Indicates elements in the user interface (menus, dialog boxes, buttons, options, file and folder names) as well as emphases in the text
Italic	Indicates product names
Underscore	Indicates cross-references and Web addresses
`Monospaced`	Indicates text or characters to be entered at the keyboard
●	The blue dot indicates the end of a chapter.

The manual also uses the following symbols to help you find your way around:

Note: This symbol draws your attention to important information relating to the current topic.

Cross-reference: This symbol indicates a reference to a related chapter, section or document.

Tip: This symbol draws your attention to recommendations on how to use our products.

Stop: This symbol indicates that you should not carry out a particular action.

New: This symbol indicates that the Operating Instructions have been changed or that new material has been added at this point.

●

Introduction

What is PyCorder?

PyCorder is based on the Python programming language and the software is open source. It is freely configurable and you are able to extend it by adding modules you have written yourself.

PyCorder is used together with the actiCHamp EEG amplifier. In combination with active electrodes, sensors or a PC, it permits the recording of ExG signals (e.g. EEG, ECG, EOG and EMG signals) and sensor data. The acquired data is stored on a computer as digitized raw data.

PyCorder provides you with unrestricted control over the processes running in the software. You can intervene in the recording and storage process at any point to configure *PyCorder* to suit your research requirements. You can modify the basic modules supplied as well as integrate your own modules in *PyCorder*.

Intended use: What can PyCorder be used for and what can it not be used for?

PyCorder may be used exclusively for research purposes.

PyCorder is not a medical product and is therefore not subject to the regulations set out in EU Directive 93/42/EEC. It is expressly forbidden to use the product for medical diagnosis or therapy of any kind.

Brain Products GmbH accepts no liability for incorrect use or misuse of *PyCorder* and/or *actiCHamp*. The use of the products beyond the boundaries of pure scientific research is deemed to be incorrect use or misuse. ●

Chapter 1 PyCorder under Windows® 7 and Windows® 8

1.1 Hardware and software requirements

The system has the following hardware and software requirements:

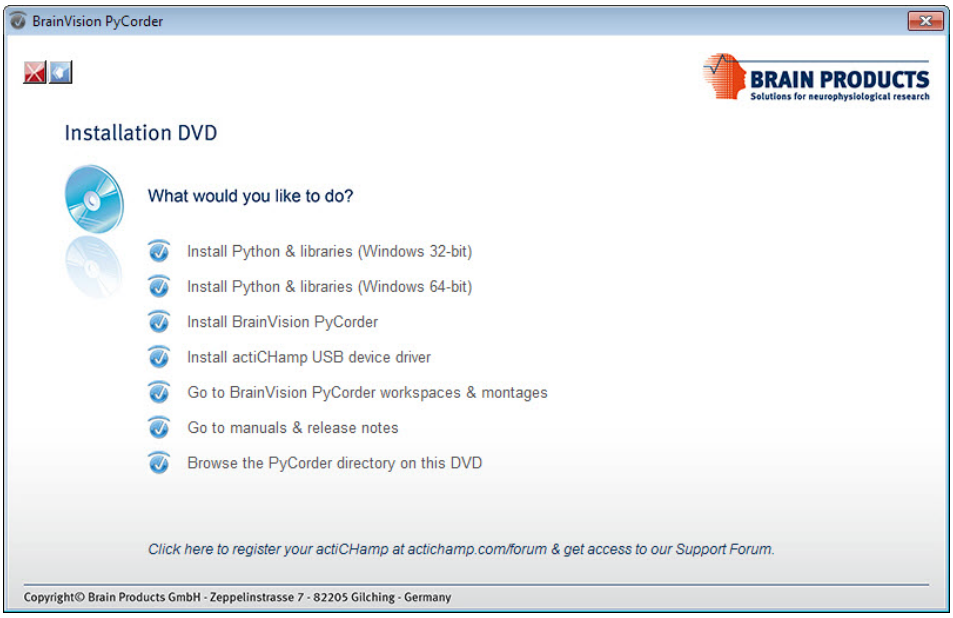
Operating system:	- Windows® 7, 32- or 64-bit - Windows® 8, 32- or 64-bit
Min. computer configuration:	- Intel® Core™ 2 Quad processor, 2.4 GHz or similar - 3 GB of RAM - Graphics adapter with 1280 x 1024 pixel resolution and min. 512 MB internal memory - Windows® Experience Index › 5.0

1.2 Installing PyCorder

You will need a number of different software packages: Python, the six Python libraries and, of course, *PyCorder* itself. The installation procedure is simplified with the installation menus.

Insert the supplied *Application Suite* into the DVD drive of your computer. When the welcome dialog opens, click on **Install BrainVision PyCorder** to open the *PyCorder* installation dialog (Figure 1-1).

Figure 1-1. PyCorder installation dialog



Proceed as follows:

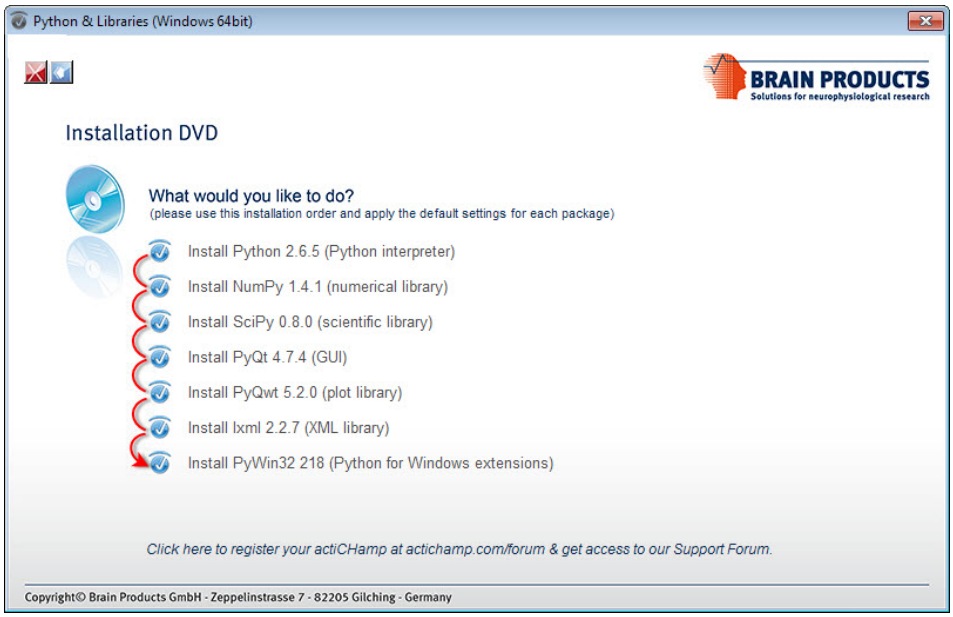
1 Choose **Install Python & libraries** for your operating system version (32-bit or 64-bit).

Note: If you wish to extend *PyCorder* with your own modules or modules of other users, we recommend to **install the 32-bit version**, even if you have a 64-bit operating system. This will facilitate the extension, since some Python libraries can only be installed in their 32-bit version.

2 Then run all the files from the Python & Libraries dialog (Figure 1-2), i.e. first Python (**Python interpreter**) and then the six Python libraries required *in the specified sequence* (**numerical**, **scientific**, **GUI**, **plot**, **XML, PyWin 32**).

Do not change the preset setup settings.

Figure 1-2. Python & Libraries installation dialog for Windows® 64-bit



3 Now return to the previous dialog (see Figure 1-1) and click on **Install BrainVision PyCorder** to install *PyCorder*.

4 The *PyCorder* application is then copied into a *PyCorder* folder in the Windows® program folder.

If you wish to program your own modules or work with the programming examples, simply copy the entire folder to a different folder on your hard disk and then work with this copy.
You will find instructions for programming your own modules in Chapter 3 as of page 43.

It is possible that the Windows® firewall may block a number of the functions of the *PyCorder*. If you receive a message to this effect, make the necessary settings to avoid this in the future.

1.3 Installing the actiCHamp drivers

You must install the necessary drivers in order to be able to operate the *actiCHamp*. You can install the drivers independently of the *PyCorder*, but must do so before you use the *actiCHamp* for the first time.

You will find the drivers on the supplied *Application Suite* DVD. In the PyCorder installation menu of the DVD (see Figure 1-1), simply click **Install actiCHamp USB device driver**. The appropriate drivers for your operating system version (32/64-bit) will be installed.

You will find instructions for connecting the *actiCHamp* to your computer in the Operating Instructions for the amplifier.

1.4 How to configure Windows® 7/8 for recording data

To ensure that data is recorded without errors, you should disable the following Windows® functions before recording:

- Sleep mode
- Windows® Update
- Windows® Defender
- Automatic defragmentation.

All these functions can be accessed from the *Control Panel*.

You can, of course, reactivate any services and functions that have been deactivated once you have finished recording data.

Refer to your Microsoft user documentation for detailed information on configuring your Windows® operating system. ●

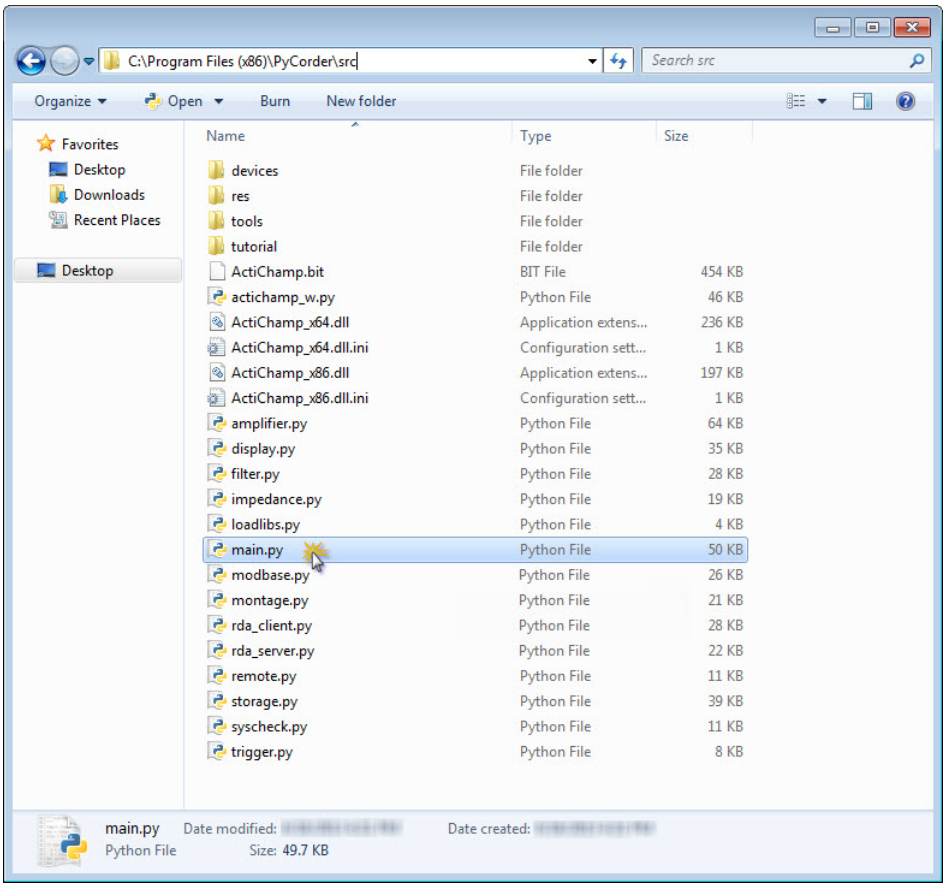
Chapter 2 Getting started and handling the program

2.1 Starting the program and GUI

Start the program by double-clicking the *PyCorder* icon that is placed on your desktop during installation. Alternatively, you can start the program by choosing BrainVision *PyCorder* under **All Programs › BrainVision › BrainVision PyCorder** in the Windows® start menu.

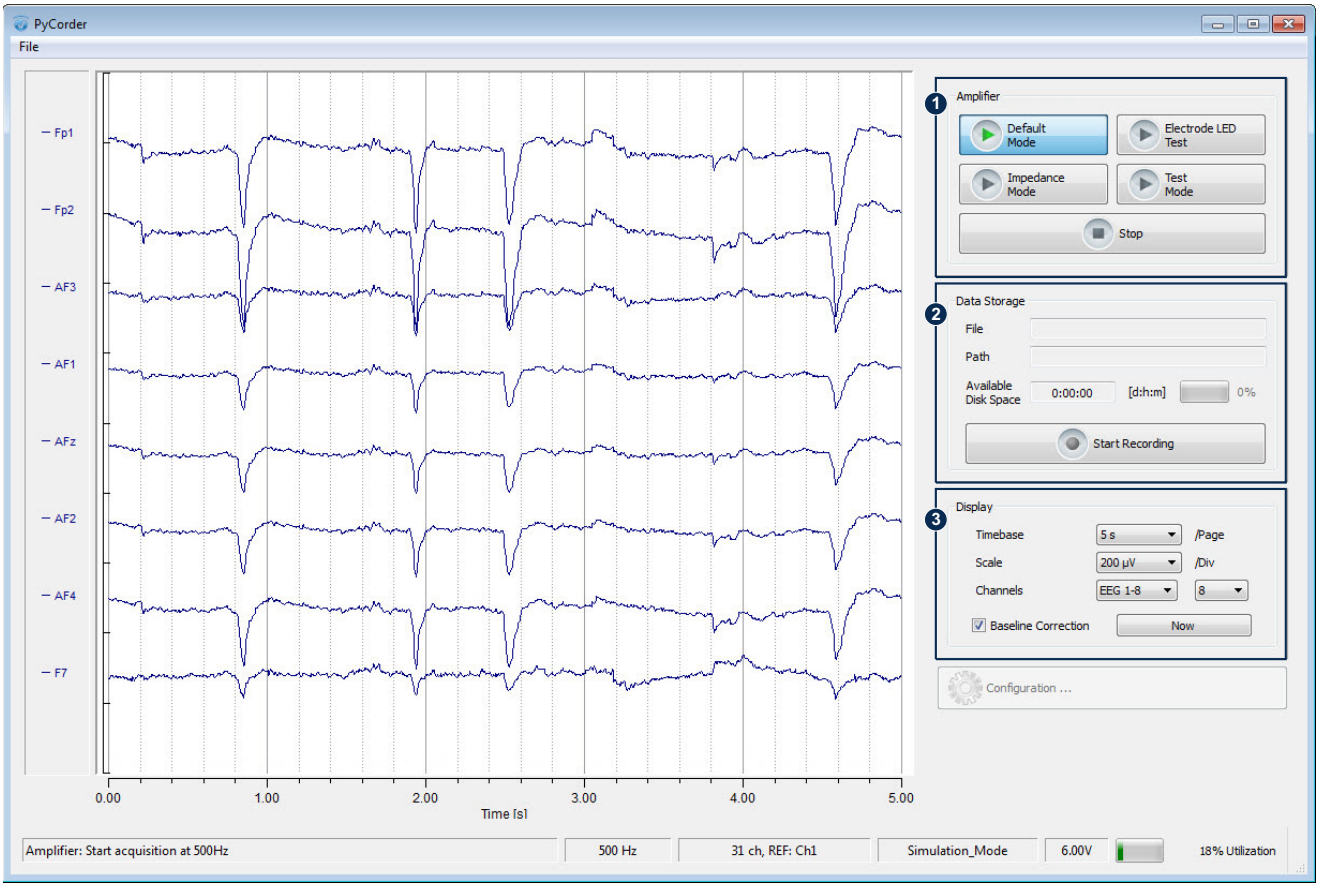
If you have created a copy of the program folder in order to write your own modules or work with the programming examples, start *PyCorder* by double-clicking the main program file "main.py" located in the source code folder (*src*) (Figure 2-1).

Figure 2-1. Starting *PyCorder* from the program folder



Note on the *PyCorder* GUI (see Figure 2-2): The data sent by the amplifier is displayed in the raw signal pane. The controls are located on the right of the window. In the **Amplifier** (1) group you select the mode. The **Data Storage** (2) group allows you to control the recording of data and view the recording status; and the **Display** (3) group allows you to change the representation of the data.

Figure 2-2. PyCorder GUI



Any changes to the states of the signal lines at the trigger input and output of the *actiCHamp* are shown as markers at the bottom of the interface (see Figure 2-3). The eight signal lines of the trigger input are acquired in two groups: The first group (D0 through D3) is designated **S nn** and the second (D4 through D7) is designated **R nn**. The decimal value **nn** (between 1 and 15) corresponds to the states of the signal lines in one of these two groups. All eight signal lines of the trigger output (D0 through D7) are designated **TOnnn**. The decimal value **nn** (between 1 and 255) corresponds to the states of these signal lines.

Figure 2-3. Display of the digital trigger inputs and trigger outputs

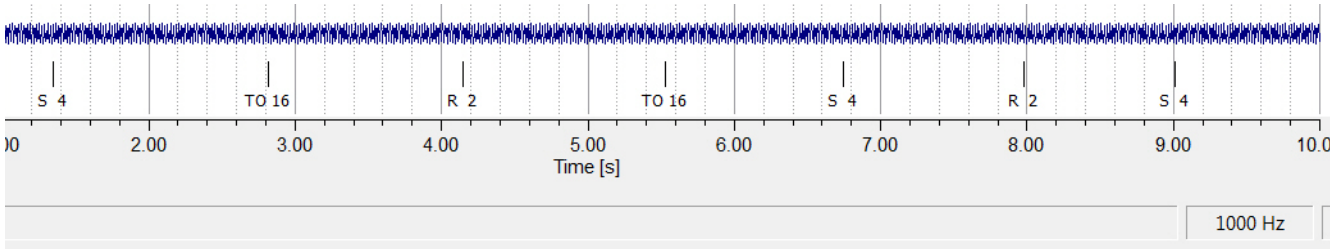


Table 2-1 contains the designations of all the markers used by default in *PyCorder*.

Table 2-1. Overview of marker designations

actiCHamp trigger connection	Bit	Marker type	Designation in PyCorder	Decimal value
Trigger input	0 through 3	“Stimulus”	*S nn*	`nn` from 1 through 15
Trigger input	4 through 7	“Response”	*R nn*	`nn` from 1 through 15
Trigger output	0 through 7	“Comment”	*TOnnn*	`nnn` from 1 through 255

Status bar

The status bar at the bottom of the window (see Figure 2-4) contains the following information:

1 Status field (indicates, for example, program status and error messages)

2 Sampling rate

3 Number of channels and reference channel

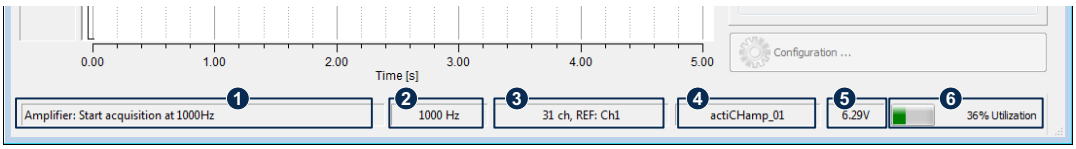
4 Employed configuration file

5 Battery voltage, color-coded:

- ▷ not highlighted (gray): normal range (5.6 V or higher)
- ▷ yellow: critical range (between 5.6 V and 5.3 V). You should charge or change the battery pack as soon as possible
- ▷ red: battery voltage too low (below 5.3 V)

6 Program utilization in percent. This value helps you to assess the performance of a (user-defined) module.

Figure 2-4. Status bar



2.2 Data display: Overview of the PyCorder modes

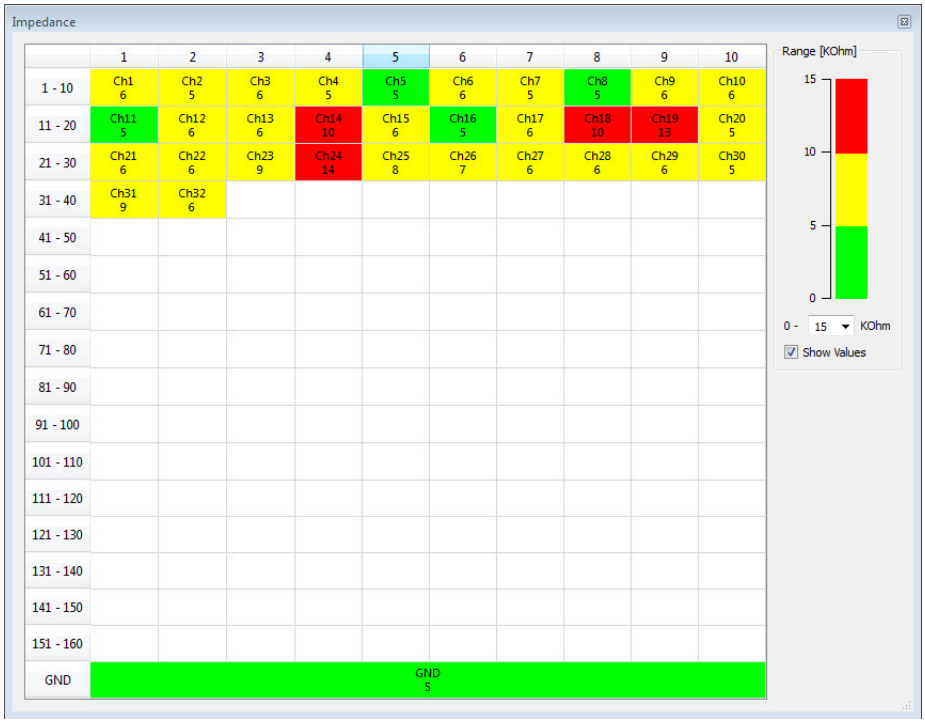
2.2.1 Default mode

To start displaying the data received from the amplifier, click the **Default Mode** button. Alternatively, sine waves are displayed if you are simulating data display (see Section 2.2.5).

2.2.2 Impedance mode

If you wish to measure the impedance values of the active electrodes, click **Impedance Mode**. In the Impedance window, the impedances of all the electrodes are shown color-coded in a grid (see Figure 2-5).

Figure 2-5. Impedance mode



Show Values allows you to show the impedance values of the electrodes. You can adjust the color scale using the drop-down list box.

During acquisition, the impedance values are also displayed by the LEDs which are built into the active electrodes. When using active electrodes, an impedance of 25 kOhm is completely sufficient in order to achieve outstanding data quality.

2.2.3 Electrode LED test

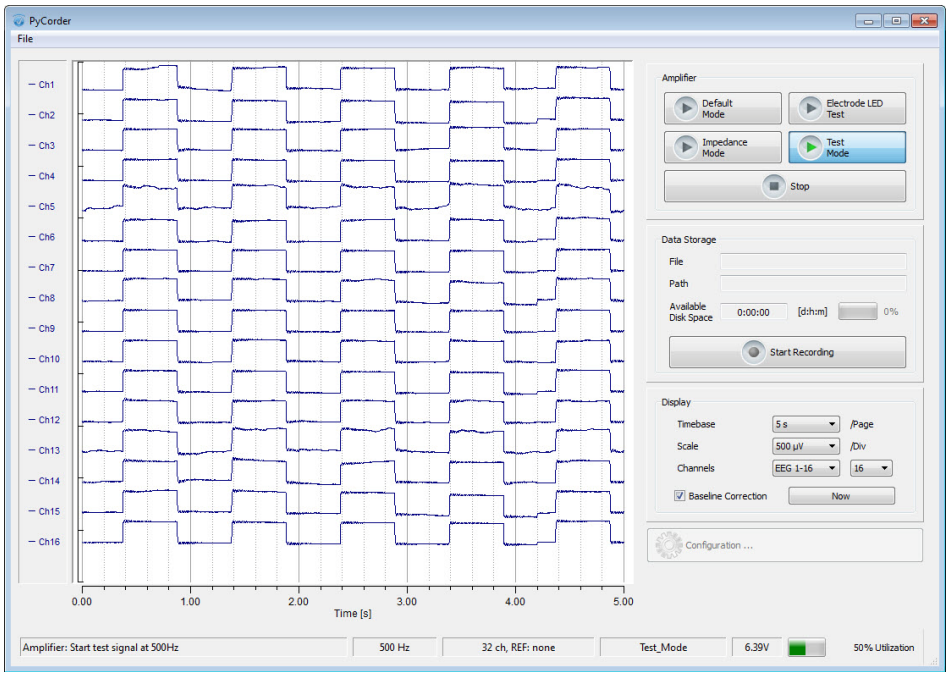
LED test mode can be used to check the LEDs of the active electrodes and *EP-PreAmps*. This makes it possible, for example, to identify defects or malfunctions. During the LED test, *PyCorder* cyclically activates the LEDs of the data electrodes and also, with a time delay, the GND electrode (green, red, off).

2.2.4 Test mode

Test mode allows you to check whether the EEG electrodes are working properly. Place all electrodes including the reference electrode and GND electrode in a saline bath (approx. 3 tablespoons of salt to 1 liter of water in a plastic bowl). Then launch test mode on the *PyCorder*. If the electrodes are operating correctly, you will see a square-wave signal in *PyCorder* (see Figure 2-6). If, during configuration, you have selected one channel as the reference channel then this channel is displayed as a normal EEG channel in test mode.

You will find detailed information on performing tests and checking correct operation of the electrodes in the actiCHamp Operating Instructions.

Figure 2-6. Square-wave signal in test mode

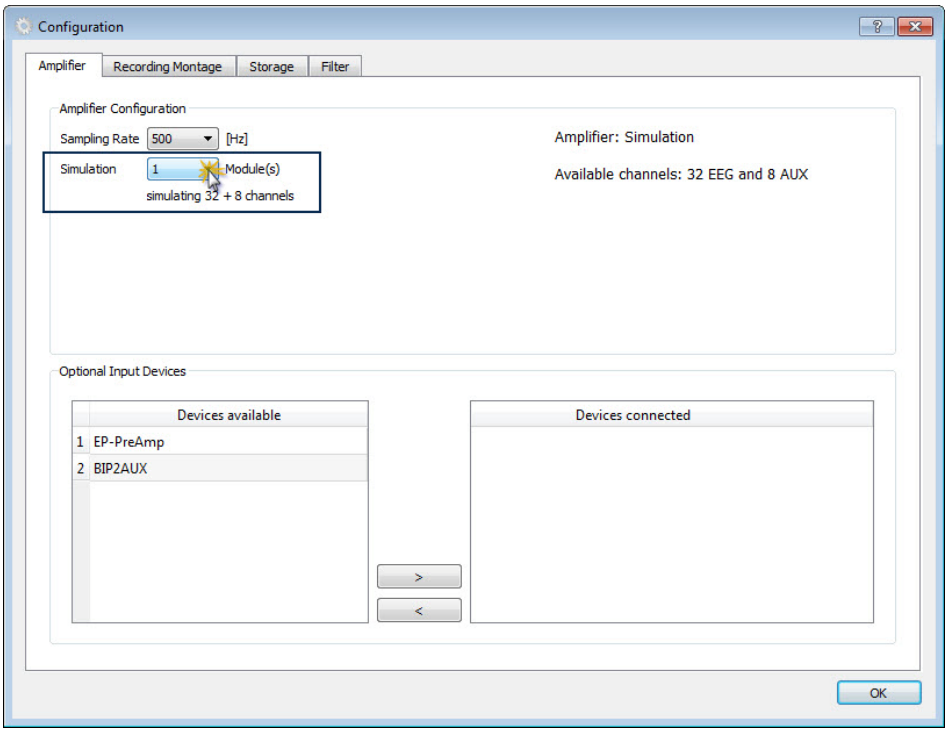


2.2.5 Demo mode: Creating and testing modules without connected hardware

If you wish to create and test your own modules without having an amplifier connected, you can use demo mode.

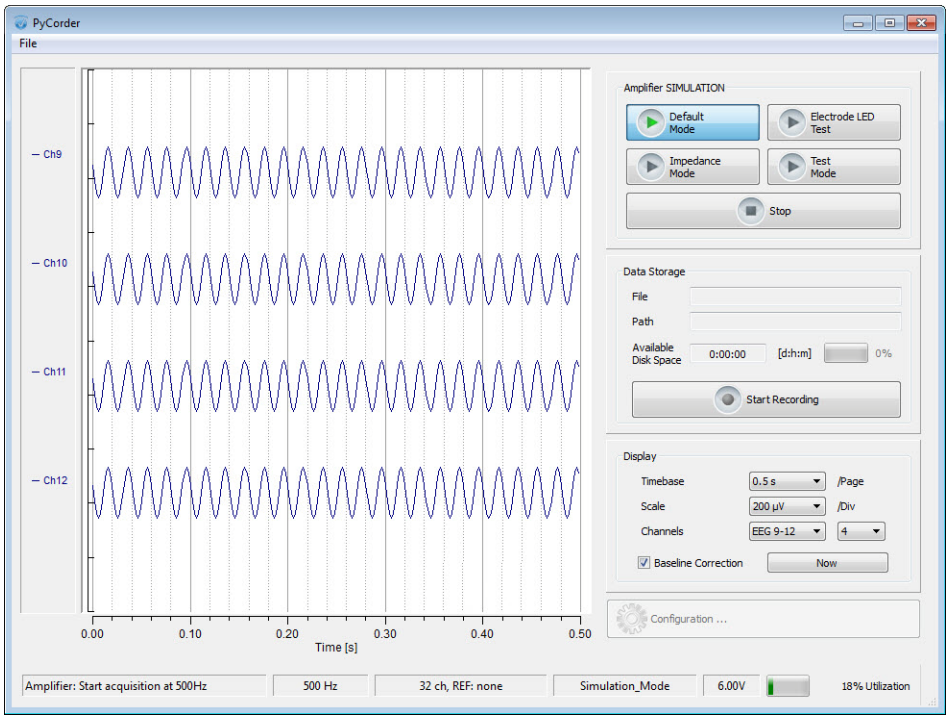
To activate demo mode, choose the number of amplifier modules from the configuration settings (**Configuration...** button) depending on the number of channels you wish to simulate (see Figure 2-7). You can simulate up to five amplifier modules or up to 160 EEG channels.

Figure 2-7. Activating demo mode



If you click **Default Mode** then sine waves are displayed (see Figure 2-8). In **Test Mode**, by contrast, square-wave signals are displayed.

Figure 2-8. Sine waves in demo mode



2.3 Recording and storing data

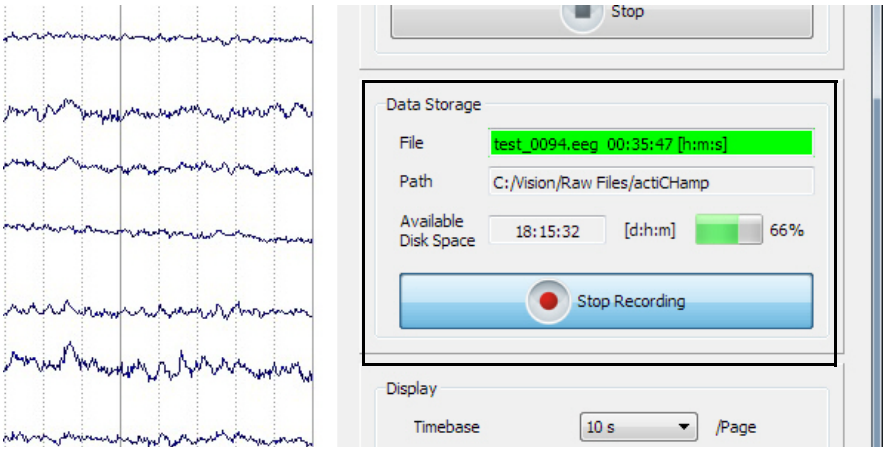
To record data, switch the *PyCorder* to **Default Mode** and then click **Start Recording**.

You can also record the test signal: To do this, switch *PyCorder* to **Test Mode** and start the recording.

In the Save dialog box that opens, you can specify a name for the data set and the location at which it is to be stored.

After you have started recording, the name, length and path for the data set and the remaining hard disk space are displayed under Data Storage (see Figure 2-9).

Figure 2-9. Displaying memory information



2.4 Configuring the data display

You can configure the data display in the Display section:

- You can change the interval shown on the time axis under Timebase (displayed interval from 0.1 through 50 seconds).
- You can adjust the scaling under Scale.
- *Channels* allows you to select how many channels and which channels (EEG, AUX, EPP[1], BIP[2]) or which channel groups (1 through 32, 33 through 64, ...) are displayed.
- Under Baseline, you can perform a baseline correction and display the baseline at any given time (**Baseline Correction Now**).

2.5 Configuring the user settings

Click **Configuration...** to change the user settings. Some modules (Amplifier, Recording Montage, Data Storage, Filter, ...) have a tab in the **Configuration** dialog box that allows you to make the necessary settings.

You can make the basic settings for the amplifier on the **Amplifier** tab (see Figure 2-10):

1. EPP = EP-PreAmp
2. BIP = BIP2AUX

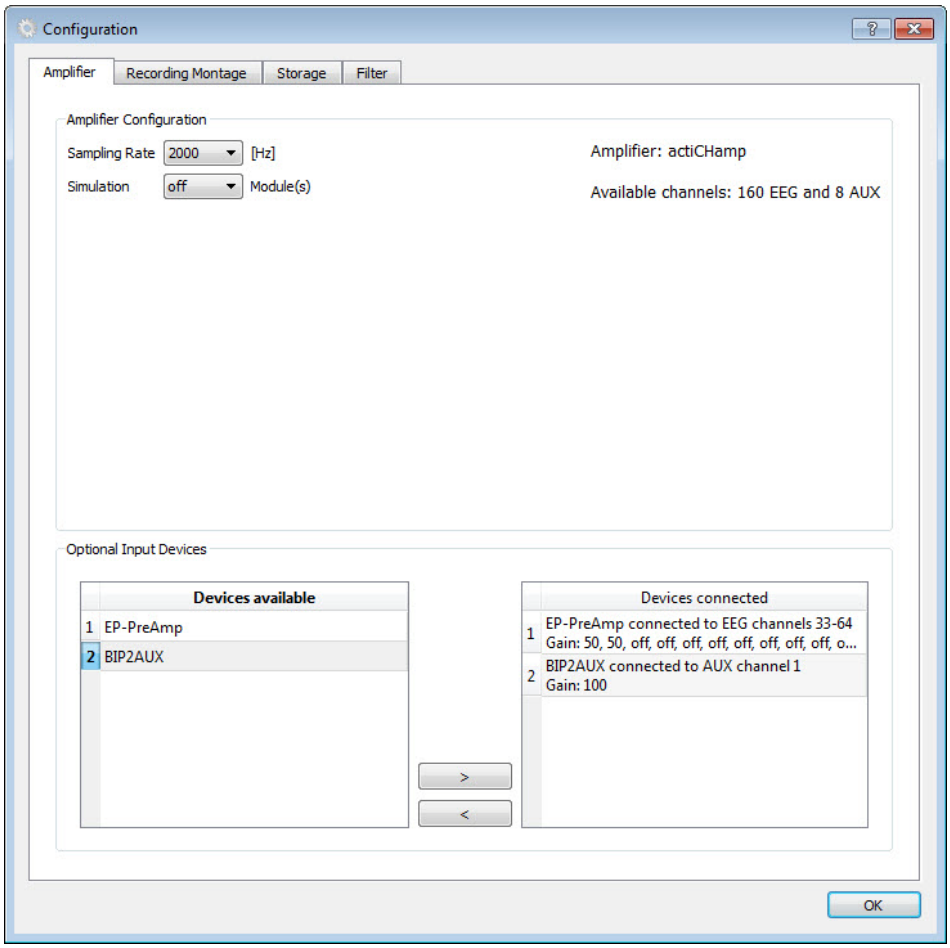
- Sampling Rate: Drop-down list for the sampling rate
- Simulation: For selection of demo mode. Select the number of modules to be simulated from the drop-down list (see also Section 2.2 as of page 24).
- Optional Input Devices: For selection of additional input devices such as, for example, *EP-PreAmp* and *BIP2AUX* Adapter.

  EP-PreAmp: Select the EEG module for the connection of the *EP-PreAmp* and define the individual gains for the *EP-PreAmp* in the drop-down lists.

  BIP2AUX: Specify the AUX channel to which you have connected the *BIP2AUX* Adapter.

For more detailed information on the settings for EP-PreAmp modules, refer to the EP-PreAmp Operating Instructions.

Figure 2-10. Configuring the amplifier settings



On the **Recording Montage** tab, you can define the settings for the EEG and AUX channels.

- You can enable and disable individual channels
- You can specify one or more reference channels for the EEG channels.

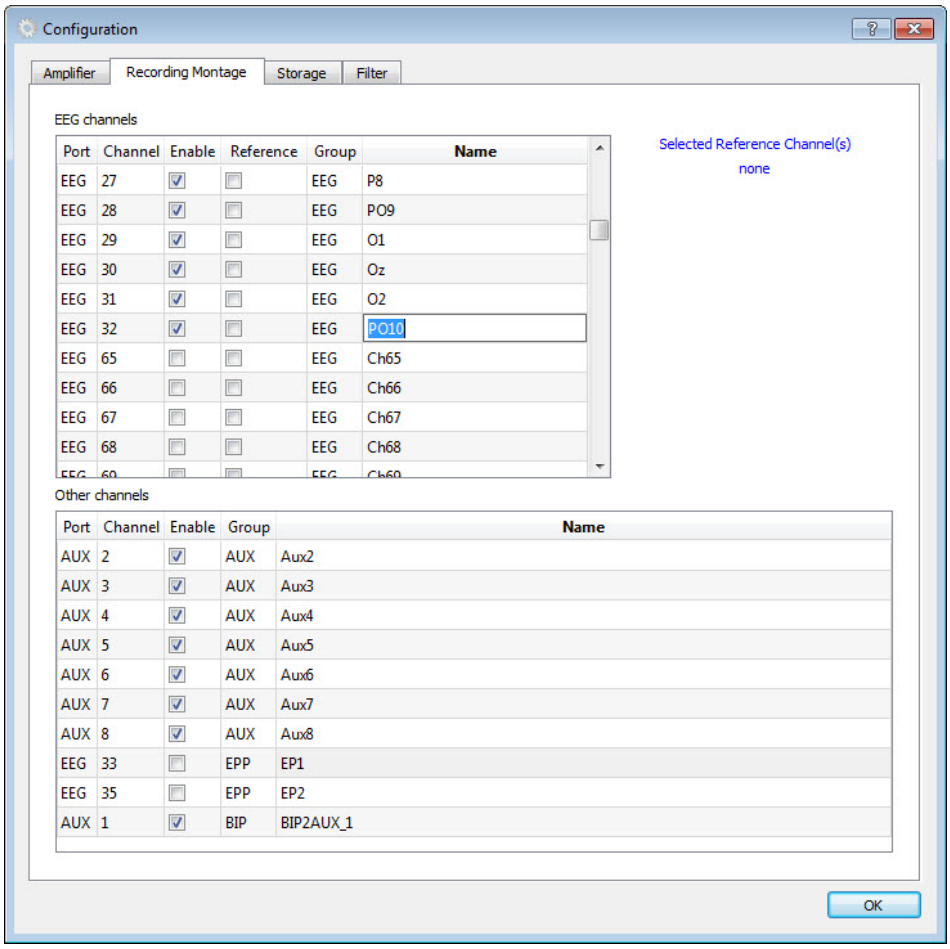
Please note here that the signals are acquired with *actiCHamp* without the use of a physical reference electrode and that the amplifier internally forms a virtual ground point as the reference system for the acquisition (see also the *actiCHamp Operating Instructions*).

If you select a single reference channel in *PyCorder* then, for every data point, the program calculates the difference between one of the other acquired channels and the reference channel. If you select multiple reference channels then the arithmetic mean of these channels is used for the difference calculation.

- You can edit the channel name.
- You can enable/disable and edit the AUX and EP-PreAmp channels.

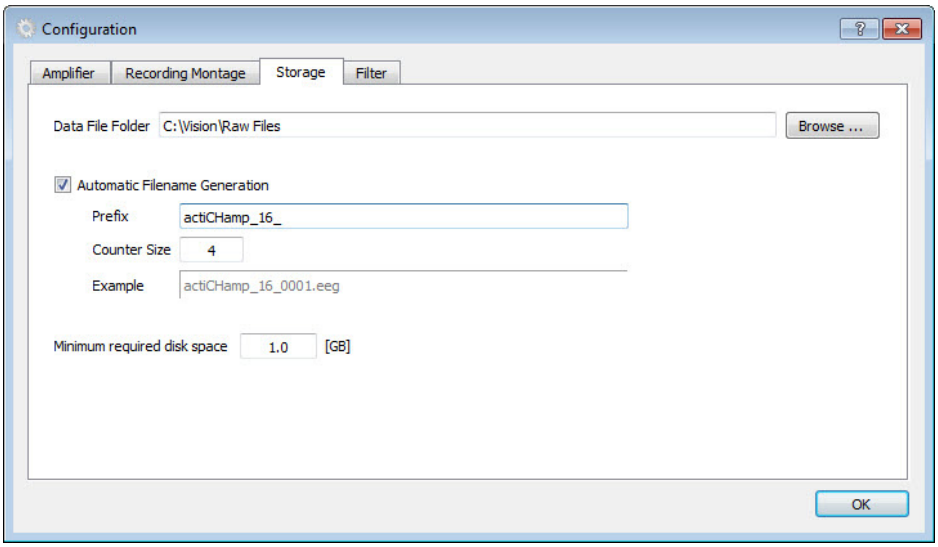
The *Other channels* list contains the AUX channels and the channels for the optional input devices (for instance *EP-PreAmp*). This list only shows the odd-numbered channels, because an *EP-PreAmp* always occupies two channels. The corresponding even-numbered channels are selected automatically. Please note that only *EP-PreAmps* that you have not set to **OFF** are displayed.

Figure 2-11. Configuring channel settings



You can configure the storage options on the **Storage** tab (see Figure 2-12). Here you can specify the folder in which your recording data is to be stored. You can also generate automatic file names and specify a minimum amount of hard disk space required for recording.

Figure 2-12. Configuring storage options

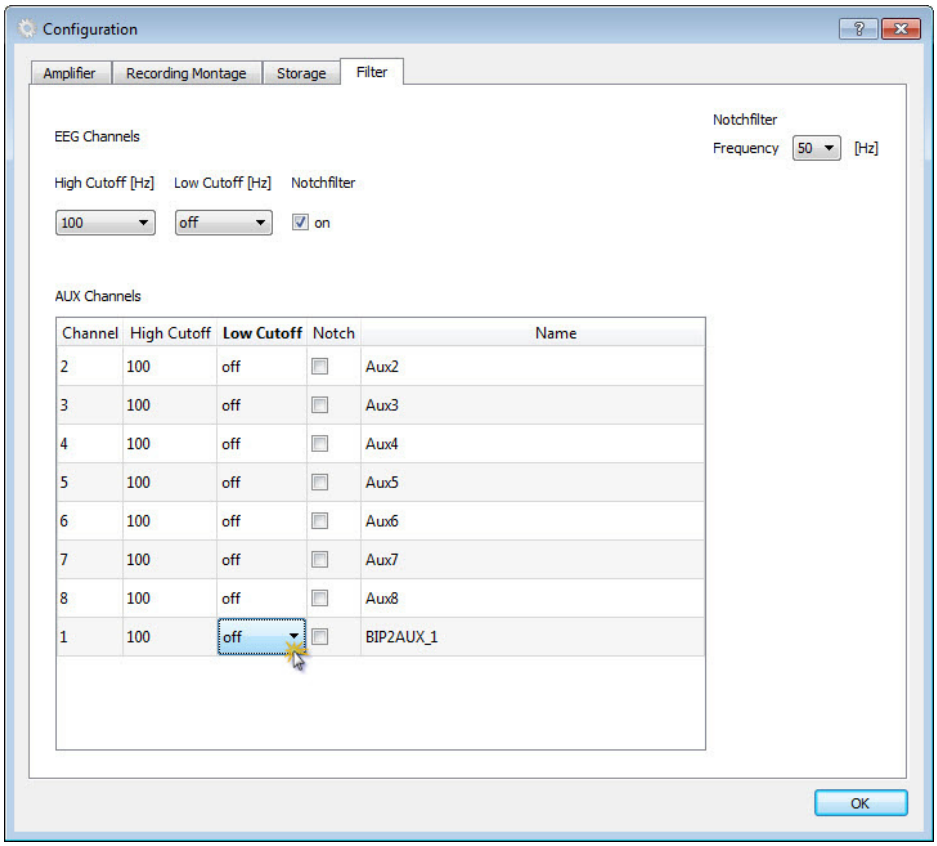


On the basis of the reserved storage space, *PyCorder* calculates the approximate number of minutes of recording time available. This value is displayed in the **Data Storage** area of the raw data pane (see Figure 2-2 on page 22). If the reserved storage space is completely filled during acquisition then recording stops and an error message is displayed in the status bar.

Figure 2-13. Error message displayed when no further reserved storage space is available

You can make filter settings – globally for all EEG channels and separately for each AUX channel – on the **Filter** tab (see Figure 2-14): You can enable or disable the high-cutoff filter, the low-cutoff filter and the notch filter and select the filter frequencies.

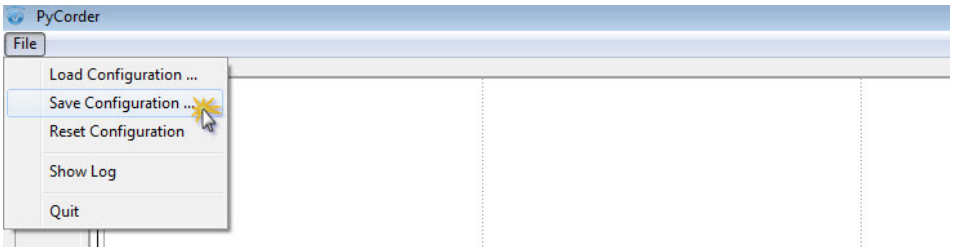
Figure 2-14. Configuring the filter settings



2.6 Simplified workflows using configuration files

To facilitate working with different configurations and switching between these configurations, you can save different user settings in separate files and read them in again subsequently. The relevant functions are available from the **File** menu (see Figure 2-15).

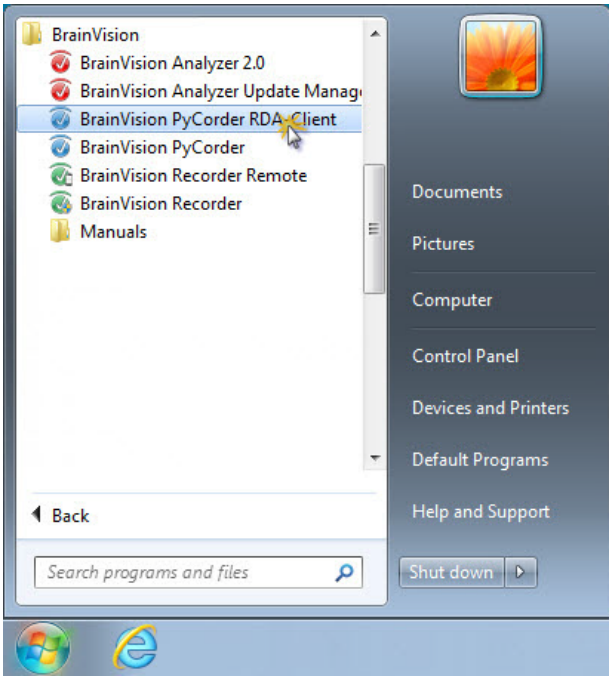
Figure 2-15. Saving and loading configurations



2.7 RDA client

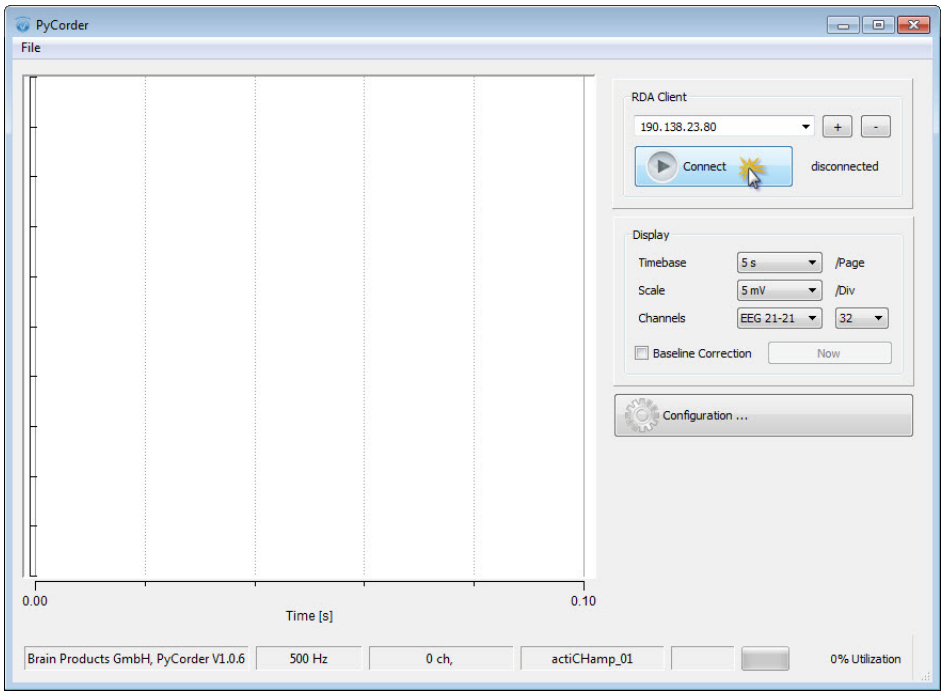
When you install *PyCorder*, the RDA client is installed at the same time. You can start the program by choosing BrainVision *PyCorder* RDA-Client under **All Programs › BrainVision** in the Windows® start menu (see Figure 2-16).

Figure 2-16. Calling the RDA client



The RDA client allows you to *monitor* data acquisition from a remote computer in the network. In this event, the *PyCorder* running on the computer used for recording the data acts as the server. To establish the connection to the server, enter its address (IP address or name) in the text box and click **Connect** (see Figure 2-17).

Figure 2-17. Starting *PyCorder* as an RDA client



2.8 Remote control using stimulus presentation software

Remote control of *PyCorder* is possible using various stimulus presentation programs (e.g. E-Prime® or Presentation®). No additional communications interface is required since *PyCorder* possesses an integrated TCP/IP server which can receive commands via port 6700.

2.8.1 Enabling remote control

The remote control function is disabled by default. To enable this function, it is necessary to enter a command in the command line.

Proceed as follows to enable the remote control function:

1 Open the *PyCorder* program folder.

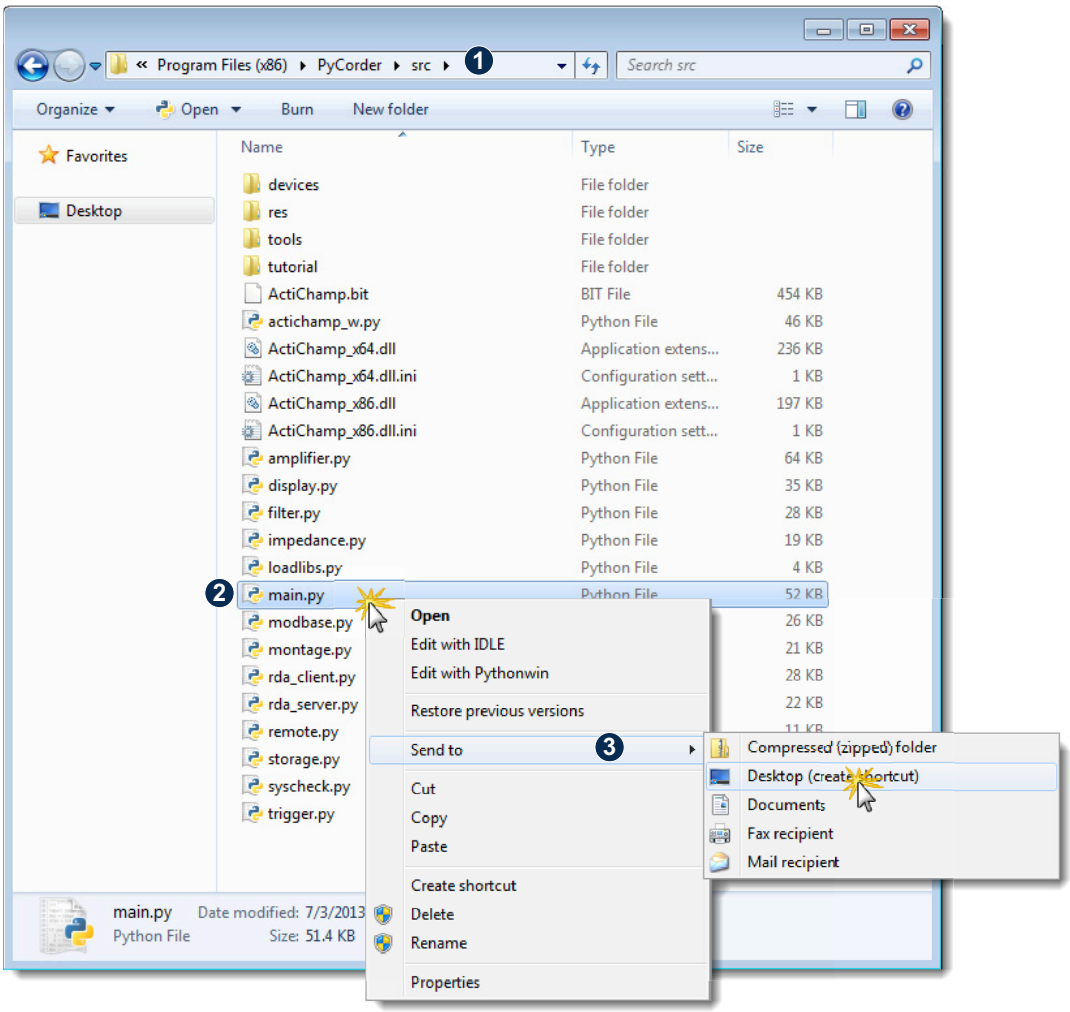
2 Right-click the file **main.py**.

3 Choose **Send To › Desktop (to create a shortcut)**[1].

1. You can move the shortcut to a different location later.

The **main.py** shortcut icon is now present on the desktop.

Figure 2-18. Creating the main.py shortcut on the desktop



4 Right-click this desktop shortcut and choose **Properties**.

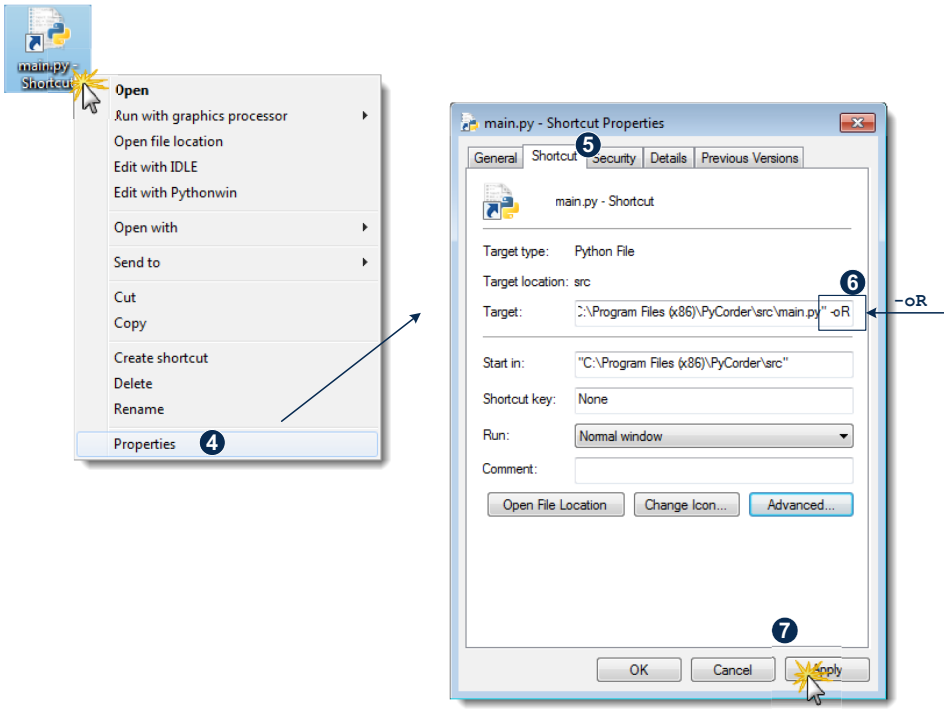
5 Open the **Shortcut** tab and click in the **Target** line.

6 Move the cursor to the end of the line and enter "`-oR`".

Make sure you use the correct case (**o** = lowercase, **R** = uppercase).

7 Click **Apply** and then **OK**.

Figure 2-19. Enter the command “-oR“ in the command line



If you now start *PyCorder* via this desktop shortcut, the remote control function is enabled.

Only use this shortcut to start *PyCorder* if you are also using remote control!

2.8.2 Initialization

You must create a configuration file before you can use the *PyCorder* remote control capability. Both the path and file name of the configuration file are required for initialization.

First of all, you must enter the IP address and port (6700) of the PC used for recording in the stimulus presentation software. If the same PC is used for recording and to run the stimulus presentation software, enter “`localhost`” as IP address. The connection is established via TCP/IP.

- Name: PyCorder
- Server: IP address or localhost
- Port: 6700
- Connection type: TCP (TCP/IP)

Initialization commands

Each of the initialization commands consists of a number and a string (Table 2-20 presents an overview). These commands must be executed in sequence and it is necessary to wait for approximately 1 second between each command in order to ensure that *PyCorder* is initialized correctly. There must be NO SPACES between the command number and the string.

Table 2-2. Initialization commands for remote control

String	Function
`1PathToConfigurationFile`	This command defines the „path to the configuration file.
`2ExperimentNumber`	You can specify a name, a number or a combination of the two. An underscore is inserted in the file name between the experiment number and the subject ID.
`3SubjectID`	You can specify a name, a number or a combination of the two.
`4`	This command loads the configuration file.

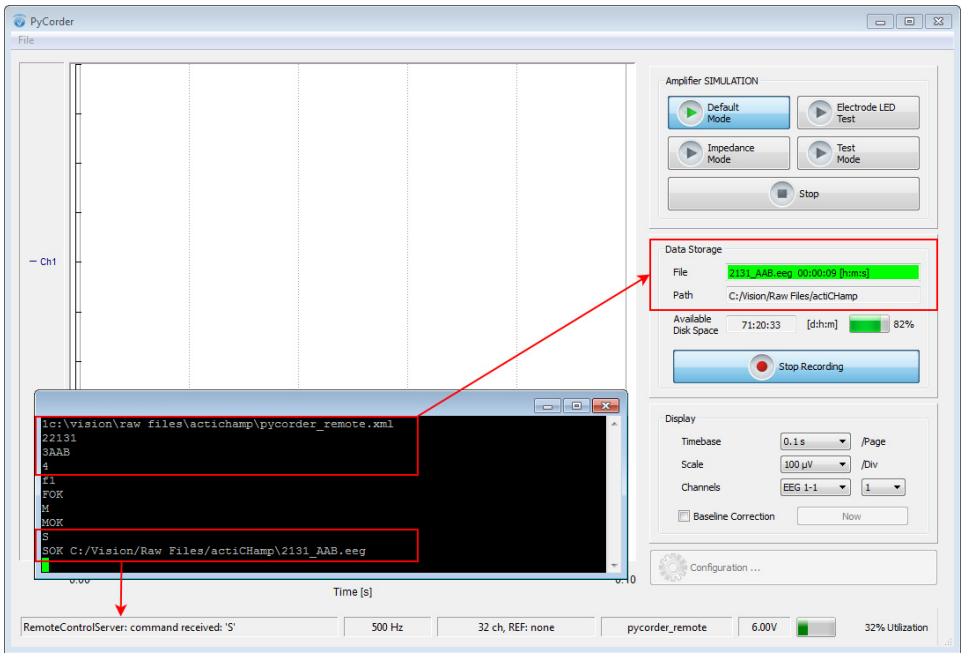
Example

```
1C:\Vision\Raw Files\config.xml

22131

3AAB

4
```

In this example, the resulting EEG file is saved as “`2131_AAB.eeg`” in the folder specified under “Data File Folder” in the configuration. If an EEG file with this name already exists then a sequential number is appended to the specified name.

The *PyCorder*'s status bar indicates that a connection has been established to the client software (e.g. stimulus presentation software) and shows you which commands have been received (Figure 2-20).

Figure 2-20. Example of remote control using client software



2.8.3 Commands

1 Mode commands

Both impedance and default mode are available for the remote control of *PyCorder*. You can start these using the following commands.

String	Function
I	Impedance mode
M	Default mode (monitoring)

2 Recording commands

You can control the recording of the data using the following commands.

String	Function
S	Starts recording.
Q	Stops recording.
X	Stops recording and exits the mode (impedance/default)

3 Feedback

You can use the following commands to enable and disable the feedback function in order to obtain feedback on the transmitted commands:

String	Function
F0	Feedback: Off
F1	Feedback: On

The called command with the suffix **OK** is returned as feedback (e.g. **MOK** or **XOK**, etc.). If a command cannot be executed then **FAILED** is returned (e.g. **XFAILED** etc.).

2.8.4 Useful notes

It is **not necessary** to distinguish between uppercase and lowercase when entering the commands.

Remote control is only possible after you have accepted the disclaimer displayed on program start-up! The commands **I, M, S, X, Q** are not available during a manually started recording.

The initialization commands 1, 2, 3, 4 are not available while recording is in progress.

Chapter 3 Creating and implementing your own modules

The following sections refer to *PyCorder* Version 1.0.0. New features in the development environment or in *PyCorder* are not taken into account here.

3.1 Installing a Python development environment

We recommend that you install a separate Python development environment comprising the following components:

- Java Runtime Environment (JRE) available from http://java.com/
- Eclipse Classic 3.x available from http://www.eclipse.org/downloads/
- PyDev (Python add-on for Eclipse)

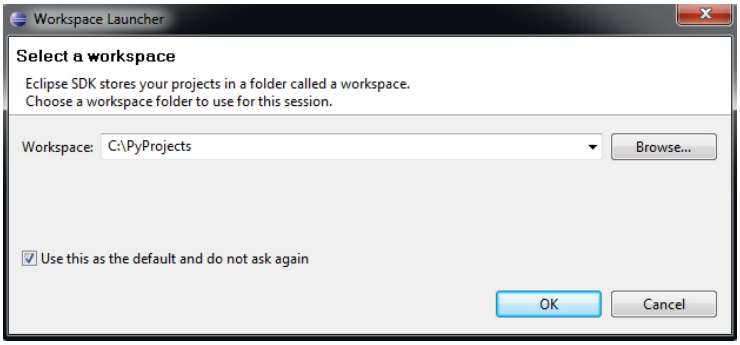
3.1.1 Setting up Eclipse and installing the PyDev add-on

Proceed as follows to set up Eclipse and install the PyDev add-on:

1 When you first start Eclipse, you are prompted to create a folder for your projects (see Figure 3-1). Under Workspace, choose a folder and check the option *Use this as the default (...)*. Close the dialog box by clicking **OK**.

You will find instructions for installing Eclipse under http://wiki.eclipse.org/Eclipse/Installation (English) or http://www.inf.fh-flensburg.de/lang/eclipse/installation.htm (German).

Figure 3-1. Creating a project folder in Eclipse

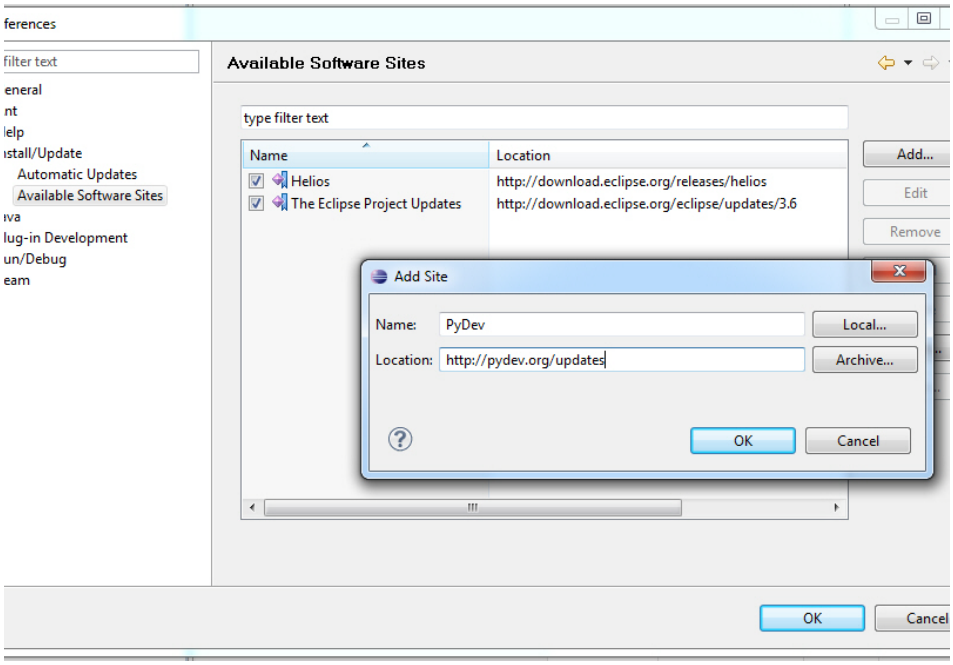


2 At the top of the *Welcome* screen, right-click on **Workbench** to switch to the actual workbench.

3 Choose **Window › Preferences** from the menu to add the PyDev add-on to the list of available software sites (see Figure 3-2). On the left, under *Install/Update*, select *Available*

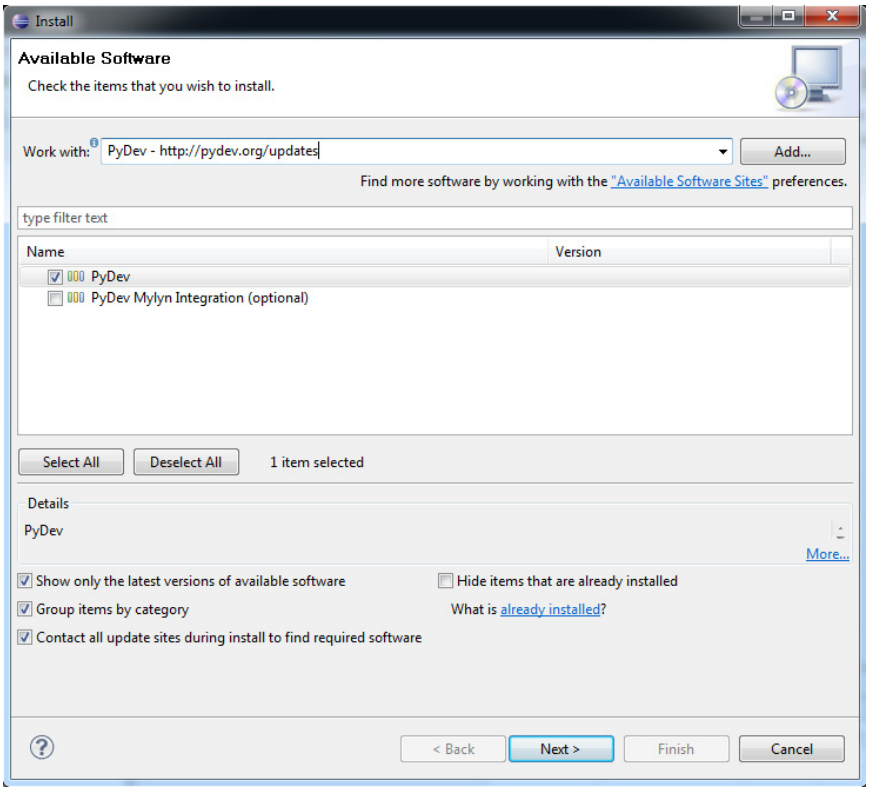
Software Sites and add PyDev as a new site by clicking **Add...**. The address of the site is "`http://pydev.org/updates`".

Figure 3-2. Adding the PyDev site to the list of available sites



4 Then install the add-on by choosing **Help › Install New Software...** from the menu (see Figure 3-3). Confirm all the dialog boxes that follow and restart Eclipse when prompted to do so.

Figure 3-3. Installing the PyDev add-on



3.1.2 Configuring the Python interpreter

Once installation is complete, you must configure the Python interpreter. This is done as follows:

1 Open the configuration dialog box by choosing **Window › Preferences** from the menu.

2 On the left, select **Pydev › Interpreter - Python** (see Figure 3-4 f.) and click **New...** to create this interpreter entry. Confirm all the dialog boxes that follow by clicking **OK**.

You will also find instructions for configuring the interpreter under: http://pydev.org/manual_101_interpreter.html.

Figure 3-4. Selecting the Python interpreter

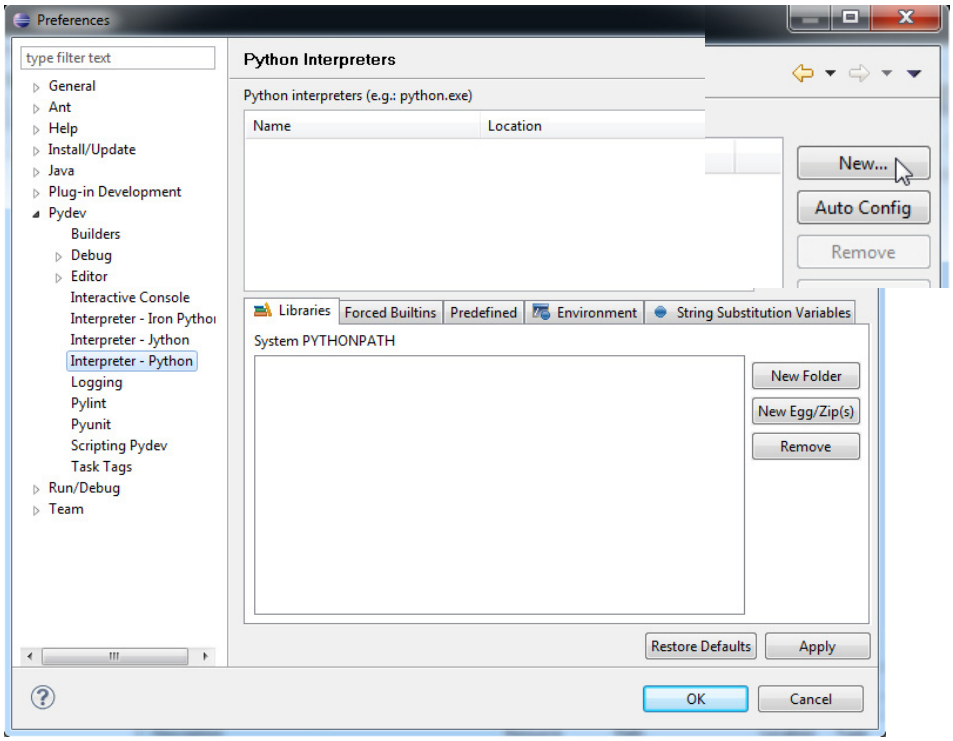
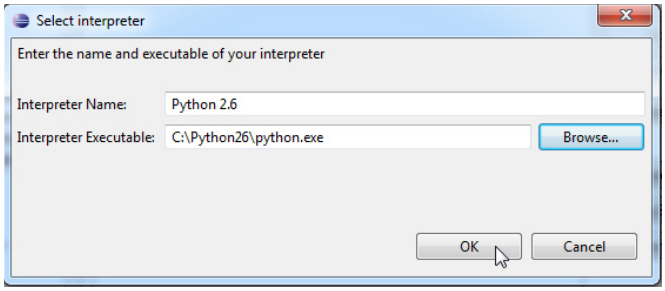
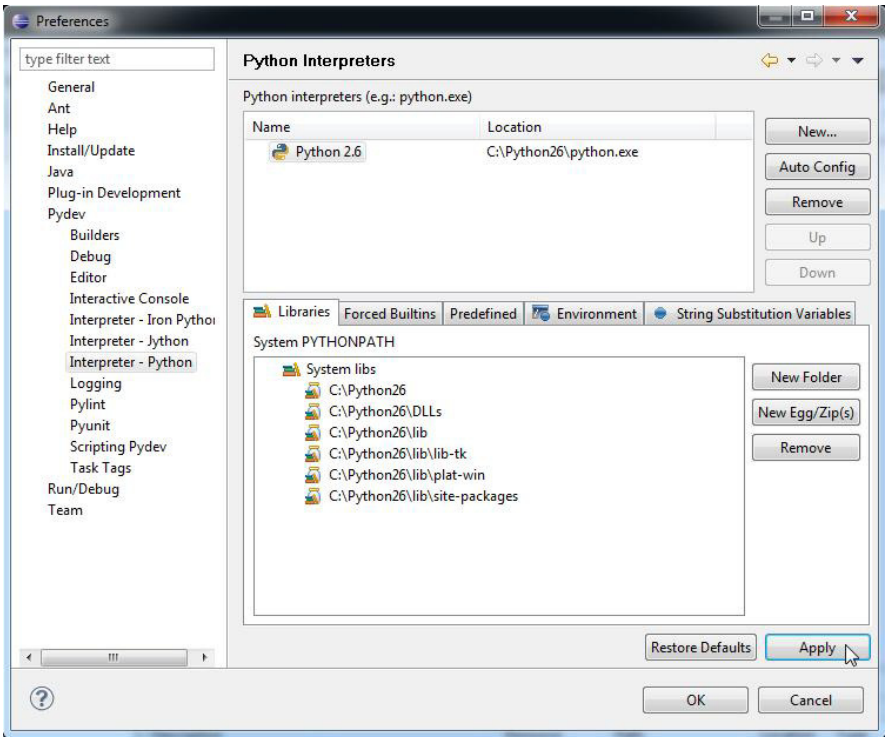


Figure 3-5. Configuring the Python interpreter



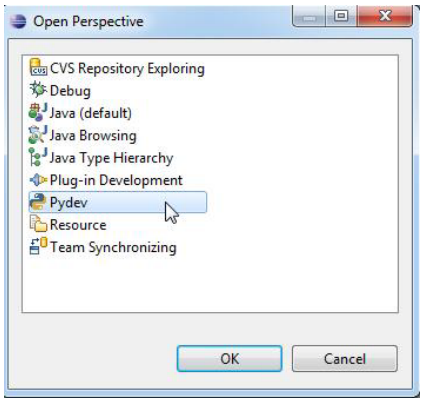
3 Click **Apply** to create the new entries (see Figure 3-6). You can then close the configuration dialog box by clicking **OK**.

Figure 3-6. Creating the interpreter entry



4 To be able to work with the Python projects, you must select the appropriate Eclipse perspective by choosing **Window › Open Perspective › Other...** from the menu. Select *Pydev* and confirm your choice with **OK** (see Figure 3-7).

Figure 3-7. Selecting the PyDev perspective

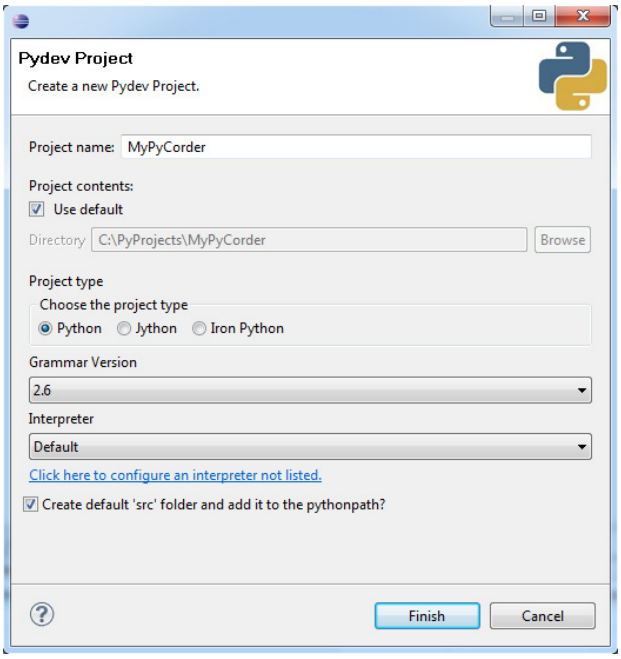


3.1.3 Importing the PyCorder project to the Eclipse environment

After you have configured the Python interpreter, import your *PyCorder* project into the Eclipse workspace as follows:

1 First create a new Python project by choosing **File › New › Pydev Project** from the menu and selecting a project name (e.g. "MyPyCorder"). The following settings should be made for the new project (see Figure 3-8):

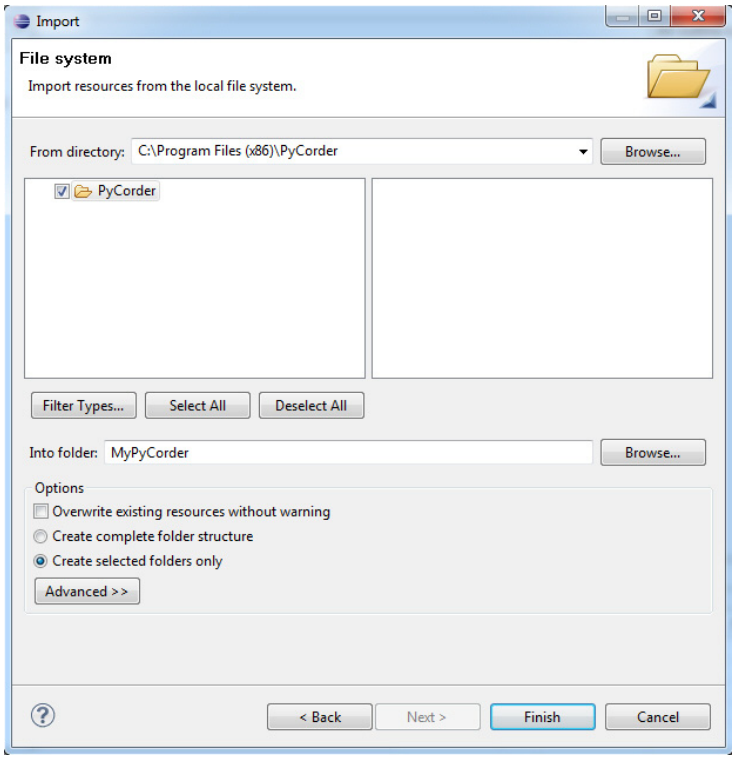
Figure 3-8. Properties of the new Python project



2 Right-click the new project in the Pydev Package Explorer and choose **Import...** from the context menu. Specify **General › File System** as the import source in the dialog box that follows.

3 Now choose the folder in which the *PyCorder* sources are located (on a default installation, this is **C:\Program Files\PyCorder**) and choose the option **Create selected folders only** (see Figure 3-9).

Figure 3-9. Selecting the program folder



4 Click **Finish**. The complete project will now be copied.

5 Once the project has been imported, select the file **main.py** and start the *PyCorder* by choosing **Run › Run As › Python Run** from the menu.

3.2 The basic framework

The basic Python application uses Qt from Trolltech and Nokia as its application framework and graphical user interface. This is linked in with PyQt. The user interface was designed with the Qt Designer and compiled to various Python modules using the PyQt compiler.

The application itself has only a few functions of its own. Instead, it provides a framework for the *PyCorder* modules. This framework is in essence made up of three parts in which the *PyCorder* modules provide signal displays, controls and configuration elements.

3.2.1 The application classes and their most important functions

MainWindow class

```
class MainWindow(Qt.QMainWindow, frmMain.Ui_MainWindow)
```

Main window of the application including the menu, status bar and module management facilities.

`__init__(self)`
: Instantiation and initialization of the GUI objects.
Links for the menu items and controls for the class functions.
Linkage of the PyCorder modules.
Embedding of the signal displays and controls from the modules in the GUI.
Loading of the most recently used configuration.

`configurationClicked(self)`
: Functions for the **Configuration...** button.
Creating and calling the global configuration dialog box in which every PyCorder module can incorporate a page.

`defaultConfiguration(self)`
: **Reset Configuration** menu
Reset all PyCorder module parameters to default values.

`loadConfiguration(self)`
: **Load Configuration...** menu
Load a configuration from an XML file and pass this configuration data to all PyCorder modules. Each module selects its own data from this configuration.

`saveConfiguration(self)`
: **Save Configuration...** menu
Read the configuration of each individual PyCorder module and write the entire configuration to an XML file.

`processEvent(self, event)`
: Termination point of all events sent by the PyCorder modules. These are evaluated here and passed to the status bar for display or registration.

DlgConfiguration class

```
class DlgConfiguration(Qt.QDialog,
frmMainConfiguration.Ui_frmConfiguration)
```

Global configuration dialog box in which the various configuration pages for the PyCorder modules are incorporated in the form of tabs.

StatusBarWidget class

```
class StatusBarWidget(Qt.QWidget,
frmMainStatusBar.Ui_frmStatusBar)
```

Status bar of the application with various fields that can be populated by the modules using events:

`labelInfo`
: Display all events of the type `LOGMESSAGE`, `MESSAGE` or `ERROR`

`labelStatus_1`
: Display the sampling rate (event type `STATUS`, field *Rate*)

`labelStatus_2`
: Number of selected amplifier channels (event type `STATUS`, field *Channels*)

`labelStatus_3`
: Display the current configuration file (event type `STATUS`, field *Workspace*)

`progressBarUtilization`
: The utilization value (event type `STATUS`, field *Utilization*) provides a reference point for the time required to process the data block in relation to the time required to record this block. If this value exceeds 100%, processing would take more time than is available, resulting in data loss. In order to avoid this, it is necessary either to optimize processing or to reduce the sampling rate or the number of channels.

DlgLogView class

```
class DlgLogView(Qt.QDialog, frmLogView.Ui_frmLogView)
```

Dialog box for displaying all module events to date of the type `LOGMESSAGE` or `ERROR`.

InstantiateModules() function

This function instantiates the PyCorder modules and passes them to the application in a list. The application then links the modules. The position in the list specifies the sequence in which data is processed. The data source (amplifier) is always the first item in the list and sends its data packages to the next module in the list.

You will find an overview of the PyCorder application, its basic modules, basic functions and communication paths in Appendix A on page 71.

3.2.2 Files and folders: Where can I find...?

Files and folders below the *src* folder of the application:

- *main.py*

  Application class and main window class (`MainWindow`)

- *modbase.py*

  Base class for all recording modules (`ModuleBase`)

- *amplifier.py*

  Module for recording data from the actiCHamp (`AMP_ActiChamp`)

- *filter.py*

  Modules for digital filters (high-cutoff, low-cutoff and notch) (`FLT_Eeg`)

- *trigger.py*

  Trigger detection module (`TRG_Eeg`)

- *impedance.py*

  Module for representing impedances (`IMP_Display`)

- *storage.py*

  Modules for saving data in Vision EEG format (`StorageVision`)

- *rda_server.py*

  RDA server module (`RDA_Server`)

- *rda_client.py*

  RDA client module (`RDA_Client`)

- *display.py*

  Module for displaying EEG data (`DISP_Scope`)

- *res (package)*

  - *__init__.py*

    Run this script to create form definitions from Qt Designer files (*.ui*)

  - *frm*.ui*

    Files for the Qt Designer GUI

  - *frm*.py*

    Form definitions created from UI files

- **.png*

  Bitmaps for the graphical representation of buttons

- *resources.qrc*

  Qt Designer resource file for icons and bitmaps

- *compile_rc.bat*

  Run this Windows® batch file in order to compile resources into a Python source file (*resources_rc.py*)

- *tutorial (package)*
  - *tut_*.py*

    Tutorial modules (`TUT_0` through `TUT_4`)

3.3 What are PyCorder modules?

PyCorder modules are used to process or present data block by block. When they are linked together, they define the total range of functions of the application. All the basic functions of a module are already contained in the base class `ModuleBase`. Modules derived from this class therefore simply have to implement those functions that they actually require. The `process_input` and `process_output` functions are an exception in this respect, because they always have to be implemented (see `TUT_0` in Section 3.4.1 as of page 55).

You will find block diagrams for the three functional areas in Appendix A as of page 71.

A module is divided into three functional areas:

- Execution control and communication (see `TUT_1`, Section 3.4.2 as of page 56 and `TUT_4`, Section 3.4.5 as of page 62)
- Data processing (see `TUT_2`, Section 3.4.3 as of page 57)
- Visualization and configuration (see `TUT_3` in Section 3.4.4 as of page 59)

3.4 How do I create and use my own modules?

Before you create your own modules or work with the programming examples (TUT_*), you should create a working copy of the entire PyCorder folder. If you do not do this, you will lose your changes when you update to a more recent version or uninstall the current version.

The functionality implemented in the examples has been kept very simple and is intended only to illustrate the principles involved.

3.4.1 TUT_0 – First step: Sending and receiving data

The `TUT_0` module is the simplest of the modules described by us since only the functions that are absolutely necessary are implemented in it: The module simply receives data and then forwards this unchanged. You can, for example, use this module as a template for your own modules. This is how the module is created:

In order to ensure that basic functions and user-defined functions are clearly separated, we recommend that you create a new Python package (folder) below the source code folder *src*. In our example we have used *src\tutorial*.

You should create a separate file for each new module class. In our example we have used *tut_0.py*.

Because all modules are derived from the base class ModuleBase, we must import this first: `from modbase import *` By specifying `*` in the import, we are also able to access other libraries which may be useful to us at a later stage and that have already been imported by `modbase`.

Now we create the class that is derived from `ModulBase` - `class TUT_0(ModuleBase)` - and implement the minimum functions required:

`def __init__(self, *args, **keys)`
: Initialization of the base class and module variables

`def process_input(self, datablock)`
: Accept data from the preceding module

`def process_output(self)`
: Send data to the next module

At this point, the module `TUT_0` is already operable.

Now all we have to do is to incorporate it into the module chain of the application as follows:

The file *src\main.py* contains the section `IMPORT AND INSTANTIATE RECORDING MODULES`. Only two entries are required here in order to incorporate an additional module:

Import the module using `from tutorial.tut_0 import TUT_0`

Create the module object and incorporate it in the module chain by making an entry in the list for the function `InstantiateModules()`:

```
modules = [AMP_ActiChamp(),
           TRG_Eeg(),
           StorageVision(),
           FLT_Eeg(),
```

```
        TUT_0(),
        IMP_Display(),
        DISP_Scope(instance=0)]
```

We have now created our first module and incorporated it in the application.

3.4.2 TUT_1 – How do I make the modules in the application communicate with each other?

There are three communication paths in the module chain. Two of these run from top to bottom, i.e. from the data source (amplifier) to the data sink (display), and one runs from bottom to top, beyond the data source into the application.

- Data path

  The data objects `EEG_DataBlock` created in the amplifier are passed from top to bottom on this path. In the module, `process_input()` accepts the data object and `process_output()` forwards it to the next module.

- Control

  Execution control is performed top-down using the functions:

  `process_update()`

  This is always called if the channel configuration of the data object has changed and before every `process_start()`.

  If a module wishes to make changes to the channel configuration (e.g. change the number of channels or the channel names), it must pass this changed data object to the downstream modules at this point using `return params` and ensure that the `process_output` function passes a data object with the same structure.

  `process_start()`

  Notifies that data transfer has started.

  `process_stop()`

  Notifies that data transfer has stopped.

- Events

  Events pass through the module chain from bottom to top, finishing up in the application, where they are evaluated or displayed in the status bar.

  If a module is interested in events from downstream modules, it can evaluate these using the `process_event()` function. Conversely, events can be sent to all upstream mod-

ules or to the application using the `send_event()` function. Those functions communicate the event with a `ModuleEvent` object specifying the message.

- Error handling, displays and temporary output

  Exceptions

  Errors in the form of `exceptions` that occur in the derived functions `process_*` are automatically passed to the application via an event, where they are displayed in the status bar and written to the log file. The status bar shows the module name, the line number and a description of the error.

  Events

  Events of the type `MESSAGE` or `LOGMESSAGE` can be used to display information in the status bar of the application at runtime.

  Console output

  When creating a new module, it can be useful to display interim results or progress messages in the Python console from time to time using the `print()` command.

3.4.3 TUT_2 – What data do I receive and how can I use it?

In `TUT_2` we show how to select channels using masks and how to process these efficiently using `NumPy-Array` functions.

You should not use any `for` loops inside the `process_input` and `process_output` data processing functions to access the individual values of the data arrays, as this will inevitably lead to a dramatic reduction in performance.

If you wish to see the difference for yourself, insert the text `_loop` into any channel name. If you do this, a simple calculation is performed in a `for` loop for all channels.

In this example we also show how to create marker objects and insert them into the data stream. Markers describe any type of event and determine the time of that event in the data stream. The storage module writes them to the marker file (*.vmkr*). In order to do this, of course, the module that creates the markers must be located before the storage module.

In order to work with this example, you must create a configuration with the channel names used in the example and a sampling rate of 10 kHz:

- Channel 1: `Ch1_x2`
- Channel 3: `Ch3_x2`
- Channel 4: `Ch4_x2`
- Channel 6: `Ch6_x2`

A selection mask is created in `process_update` for the channels listed here, and this is then used in `process_input` to multiply the values of the selected channels by 2.0.

Let us now have a look at the data passed to the `process_input` function. We receive an object of the class `EEG_DataBlock` with the following fields:

Table 3-1. Fields of the object "EEG_DataBlock"

`sample_counter`	Total number of data points received since the start
`sample_rate`	Sampling rate in Hz
`eeg_channels`	Two-dimensional array of the type `float` containing the channel data. The first axis is the channel and the second axis contains the values.
`trigger_channel`	Two-dimensional array of the type `uint32` containing the values of the digital inputs and outputs. The first axis is the channel (in this case there is only one) and the second axis contains the values.
`sample_channel`	Two-dimensional array of the type `uint64` containing the sample counter returned by the A/D converter. The first axis is the channel (in this case also there is only one) and the second axis contains the values.
`channel_properties`	One-dimensional array of the type `EEG_ChannelProperties` containing the channel configuration. The size of the array always corresponds to the size of the first axis of the channel data (`eeg_channels`) and the indices are assigned 1:1.
`markers`	List of markers of the type `EEG_Marker` generated for this data block.
`impedances`	List of impedance values of the EEG electrodes and the GND electrode. These values are only valid in impedance mode (`recording_mode == IMPEDANCE`).
`block_time`	Time at which this block was recorded
`recording_mode`	Recording mode (`NORMAL`, `IMPEDANCE` or `TEST`)
`self.performance_time_max`	Longest processing time required by modules for data processing. This is used to calculate and display the utilization.

3.4.4 TUT_3 – Configuring parameters and visualizing data

The module `TUT_3` is somewhat more complex than the previous examples. It illustrates the options available for visualizing data and for configuring module parameters during recording and within the framework of the global settings using the example of a simple FFT analysis.

The purpose of the module is to display the FFT for selected channels in a signal pane and to configure the frequency range shown and the size of the data blocks during recording. It should be possible to set the maximum number of channels shown using the global configuration dialog box. Channels are selected by clicking the channel names in the raw signal pane. Clicking once on a channel adds it to the selection and clicking on the same channel again removes it. If the maximum number of channels that can be selected is exceeded, the oldest selection is canceled.

The three parameters *frequency range*, *chunk size* and *plot items* illustrate how to store module settings in the configuration file and how to read them in again.

After you have integrated the module into the application, the selected channels (in our example, these are *Ch4* and *Ch8*, see Figure 3-10) are shown to the left of the raw data pane. The controls (on the right of the interface) have been extended to include the *FFT* group containing two drop-down lists for configuring the *frequency range* and *chunk size* parameters (see Figure 3-11).

Figure 3-10. Displaying selected channels as an FFT

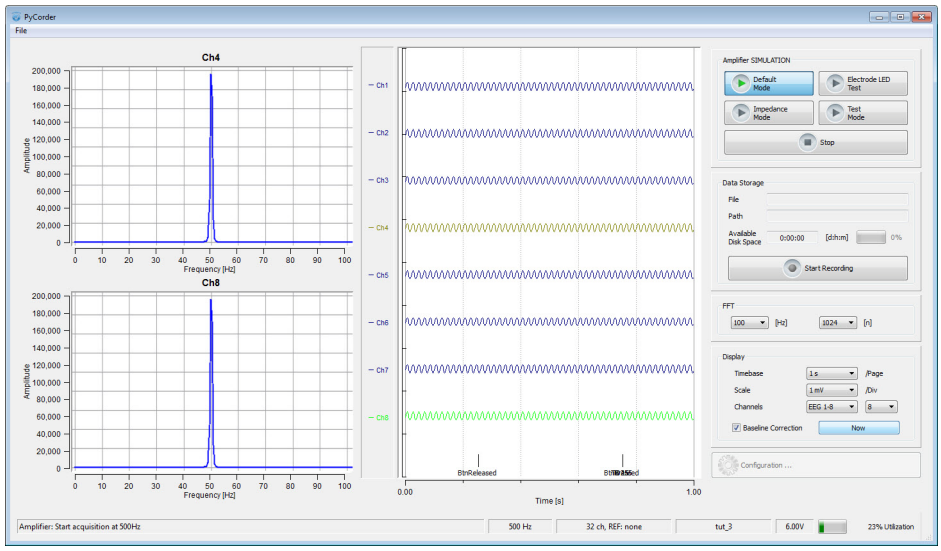
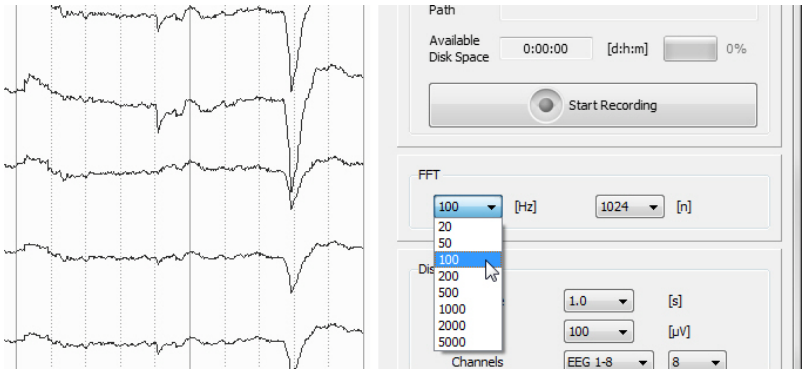
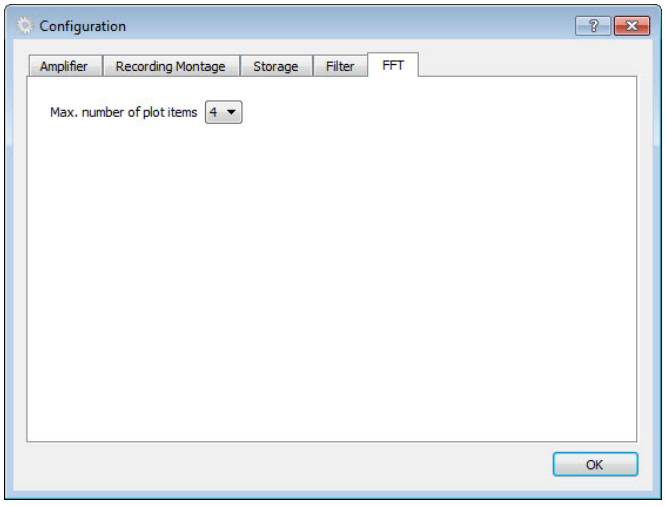


Figure 3-11. New FFT group



The configuration dialog box has been extended to include the tab *FFT*, in which you can configure the *plot items* parameter (see Figure 3-12).

Figure 3-12. Additional tab in the configuration dialog box



Now let us have a look at the required objects and functions in the visualization, data processing and communication area.

We need three objects for visualization:

- `_SignalPane` for displaying the signal
- `_OnlineCfgPane` for the controls
- `_ConfigurationPane` for configuration

You must derive all classes for these objects from Qt.QFrame in order to be able to incorporate them in the application interface. The objects are passed to the application using the

functions `get_display_pane`, `get_online_configuration` and `get_configuration_pane`. The objects for displaying the signal and for control are only queried once by the application – during initialization – and can therefore be instantiated directly in the constructor of the module. The configuration object is called every time the configuration dialog box is called and must always be created again, because QFrame objects can no longer be used after they have been closed (see `get_configuration_pane`).

There are two ways of designing graphical objects: You can do so manually as we have done in this example. Alternatively, you can use the Qt Designer that allows you to design objects using WYSIWYG methods. You must then compile the XML files created by the Designer (*.*ui*) using the Qt compiler. You will find examples of this method in the basic modules of the application. The call to the compiler is contained in the file *__init__.py* in the folder *res*.

The controller object `_OnlineCfgPane` contains only a frame and two combo boxes for setting the parameters *frequency range* and *chunk size*. The module uses the functions `setCurrentValues` and `getCurrentValues` to set or read the parameters. The Qt.SIGNAL mechanism informs the module of any changes to the settings. The connection to the relevant events is established in the constructor of the module.

The configuration object `_ConfigurationPane` has an even simpler structure, containing just one combo box for the parameter *plot items*. Any changes are written directly to the module variable `plot_items`.

The signal display `_SignalPane` provides the framework for the actual FFT objects to be represented, which it manages in a list and provides with channel data packets at runtime. A queue object is used to transport the data from the module to the signal display in order to guarantee thread safety. The data is read in a timer function and distributed to the FFT representation objects.

The FFT widgets are derived from the class `QwtPlot`, which is responsible for all of the tasks involved in drawing the signals, the axes and the axis labels.

You will find detailed information on the programming facilities and the configuration of the `QwtPlot` class in the Qwt documentation.

Once we have created the visual components, it is of course necessary to provide them with data. To do this, we first choose the channels to be displayed. This is done using an event sent by the display module when one of the channel names is clicked. Events are accepted by the `process_event` function, and at this point we filter out the event `ChannelSelected`. This event gives us the name of the selected channel, which is added to a FIFO buffer of the size `plot_items` or deleted from the buffer if the buffer already contains this channel. The channel mask is then recalculated in the `updateSignalPane()` function and the signal display is updated.

The same function is used if a new channel configuration is received via `process_update()`.

Now, everything has been prepared to allow us to use the `process_input()` function to receive data and pass it to the signal pane. The data packets that we receive via `process_input()` always have different sizes depending on the sampling rate and the pro-

cessing times. However, because we need blocks of a defined size for the FFT, the data is initially cached and only passed for FFT display when the cache contains at least one block of the required size. The FFT module is now completely operational.

We now want to make some settings to allow the three module parameters that can be set to be written to and read from a configuration file.

The application manages all module parameters in an XML file containing one node for each module. A module node is identified by its name and the two attributes *module* and *instance*. Each module can arrange its parameters in any way it chooses below the module name. The lxml library is used for writing and reading, which considerably facilitates creating and reading complete XML trees.

When the configuration file is created, the application calls the `getXML()` function of each module and builds the file from the returned module nodes. When the configuration file is read in, the entire XML tree is passed to the modules using `setXML()`. Each module then finds its own node and reads its own parameters.

In order to be prepared for structural changes to the XML parameters, we recommend that you write a version number into the module node as an attribute and evaluate this when the file is read.

3.4.5 TUT_4 – Setting the trigger output and reading the My Button

The actiCHamp has eight digital outputs, whose states can be set by the application. It is, however, only possible to use the trigger output when recording data. An event of the type `COMMAND` with `info="TriggerOut"` and `cmd_value=value` is sent to the amplifier module to send a binary value to the trigger output. Each bit in the value sent corresponds to the state of one output line (bit0=D0, bit1=D1, bit2=D2, etc.). The `_OnlineCfgPane` control of this module contains eight check boxes, that can be used to set or reset each output individually. There are also two buttons that allow you to set or reset all the outputs together (see Figure 3-13).

Figure 3-13. Setting the trigger output and showing MY Button

All the elements of the control are enabled in the `process_start()` function when data recording starts and disabled again in `process_stop()` when data recording stops. The module `TUT_4` is notified of every change to the check boxes, and the module initiates a "`TriggerOut`" event, which sets the trigger outputs directly on the amplifier.

On the front of the actiCHamp, there is a control button labeled "MY-Button" to which you can assign your own individual function. The state of the *MY button* is also distributed by means of an event. Modules that have an interest in the state of the button can evaluate this event of the type `COMMAND` with `info="MyButton"` in the function `process_event()` and react accordingly. The value of `cmd_value` is either "`pressed`" or "`released`" and indicates the current state change. `TUT_4` uses this information to show the state of the button in the control object.

3.5 Links to the Python libraries used

PyQt: http://www.riverbankcomputing.co.uk/software/pyqt
PyQwt: http://pyqwt.sourceforge.net/
NumPy: http://www.numpy.org/
SciPy: http://www.scipy.org/
lxml: http://lxml.de
PyWin32: http://sourceforge.net/projects/pywin32/

Chapter 4 Troubleshooting: What to do if...

You will find detailed information on performing hardware tests and checking correct operation of the electrodes in the Operating Instructions for the *actiCHamp*.

Your local dealer or our support forum is also available to provide assistance (see page 11).

4.1 Errors on starting the program

When the program is started, the system checks whether the required Python libraries exist and what version they are. If there are any discrepancies, the following message appears in the console window: “PyCorder: The following libraries are missing or have the wrong version”.

If one library is missing, it is not possible to continue working and the application is terminated.

An “incorrect” version of the library means that we have not tested this version. It is, however, possible that you will be able to work with this version. You must test for yourself whether it is possible to continue working without errors.

4.2 Errors that are trapped and handled by the program

All errors that occur and are trapped during program execution are displayed in the status bar and can thus also be stored in a log file.

You can display the complete sequence of all messages and errors either by choosing **File › Show Log** from the menu or by clicking on the status bar. The *Log History* dialog box then provides you with the option of storing the contents of the log file.

Error messages are displayed and handled differently depending on their severity:

- IGNORE

  Errors of this type are highlighted in yellow when they are displayed. They are for information only and have no effect on the execution of the program.

- NOTIFY

  These errors are shown in red. They generally lead to data loss and must be eliminated. Recording, however, continues.

- STOP

  These are serious errors that cause recording to be aborted. Errors of this type remain in the status bar either until they are overwritten by a further error of this type or until they are deleted by calling *Show Log*.

4.3 Possible error messages and their causes

Table 4-1. Error messages and causes

Module	Notification of errors	Severity	Cause	Solution
Amplifier	actiChamp: failed to open library ('libname')	STOP	It is not possible to access the Windows® library *ActiChamp_x86.dll* or *ActiChamp_x64.dll* for the amplifier, because it is missing or faulty.	Check that the library with the file name *libname* is located in the same folder as the file *actichamp_w.py*.
	actiChamp: hardware not available	STOP	No amplifier was found at the USB port.	Check that the amplifier is connected and is shown in the Windows® Device Manager under the name “ActiChamp EEG amplifier”.
	actiChamp: failed to open device	STOP	An amplifier was found, but it was not possible to initialize it.	Check the power supply of the amplifier. Disconnect the amplifier from the USB bus, reconnect it and restart the application.
	actiChamp: device not open	STOP	This error and the errors below indicated by 'error' = “Invalid handle” or “Invalid function parameter(s)” only occur when the control functions of the amplifier have been called without having been initialized successfully beforehand or when the parameters passed to one of the library functions lie outside the valid range. If 'error' has the value “Function fail (internal error)”, this indicates a problem communicating with the hardware.	If you have made changes to the standard modules, check whether this error also occurs with the original installation. If so, contact your local dealer or the support forum (see also page 11).
	actiChamp: failed to setup device -› 'error'	STOP	See above	
	actiChamp: failed to get device properties -› 'error'	STOP	See above	
	actiChamp: failed to start device -› 'error'	STOP	See above	
	actiChamp: failed to stop device -› 'error'	STOP	See above	
	actiChamp: failed to read data from device -› 'error'	STOP	See above	

Table 4-1. Error messages and causes

Module	Notification of errors	Severity	Cause	Solution
	actiChamp: failed to read impedance values -> 'error'	STOP	See above	
	actiChamp: failed to set LED impedance range -> 'error'	STOP	See above	
	actiChamp: failed to set trigger output -> 'error'	STOP	See above	
	no input channels selected!	STOP	No channel has been selected in the amplifier configuration.	Open the configuration dialog box (*Configuration...*) and select at least one channel for recording.
	connection to hardware is broken!	STOP	The connection to the hardware has been interrupted during recording.	Check the USB connection and the power supply of the amplifier.
	XML Configuration: wrong version	NOTIFY	An attempt has been made to read a configuration file with a higher module version number than the version currently implemented.	Configuration files created with a later program version cannot be read by older program versions.
	Faulty internal voltage(s):...	NOTIFY	One or more of the internal supply voltages are outside the tolerance range.	Please contact your local dealer or the support forum (see also page 11).
Display	XML Configuration: wrong version	NOTIFY	See 'Amplifier'	
	Input queue FULL, overrun!	NOTIFY	The module is not able to process the incoming data quickly enough, resulting in an overflow in the input queue.	This is generally a problem with the performance of the computer hardware used. Reduce the sampling rate or the number of channels until the Utilization value drops below approximately 80%.
Filter	XML Configuration: wrong version	NOTIFY	See 'Amplifier	
	Input queue FULL, overrun!	NOTIFY	See 'Display'	
Impedance	XML Configuration: wrong version	NOTIFY	See 'Amplifier'	
	Input queue FULL, overrun!	NOTIFY	See 'Display'	
Main	Load Configuration: 'filename' is not a valid PyCorder configuration file	NOTIFY	An attempt was made to read an XML file that does not contain valid PyCorder configuration data.	

Table 4-1. Error messages and causes

Module	Notification of errors	Severity	Cause	Solution
	Load Configuration: 'filename' wrong version 'vf' › 'va'	NOTIFY	An attempt has been made to read a configuration file with a higher program version number than the version currently implemented.	Configuration files created with a later program version cannot be read by older program versions.
	Save Configuration: 'error' -› 'filename'	NOTIFY	An error has occurred on saving the configuration file filename.	'error' indicates the cause of the error, e.g. write-protected, insufficient permissions or insufficient storage space.
	PyCorder: Failed to write log file 'filename' -› 'error'	NOTIFY	An error has occurred on saving the log file *filename*.	'error' indicates the cause of the error, e.g. write-protected, insufficient permissions or insufficient storage space.
RDA Client	XML Configuration: wrong version	NOTIFY	See 'Amplifier'	
RDA Server	XML Configuration: wrong version	NOTIFY	See 'Amplifier'	
	Input queue FULL, overrun!	NOTIFY	See 'Display'	
	RDA Client input queue FULL, overrun!	NOTIFY	The connected RDA client or the TCP/IP connection is not able to process the incoming data quickly enough, resulting in an overflow in the input queue.	The only remedy here is a faster network connection or a reduction in the sampling rate or the number of channels.
Storage	XML Configuration: wrong version	NOTIFY	See 'Amplifier'	
	Input queue FULL, overrun!	NOTIFY	See 'Display'	
	out of disk space ('minspace'GB) on 'path'	NOTIFY	The selected storage medium does not have enough free storage space.	Choose a different storage medium or reduce the space required using the configuration dialog box.
	out of disk space (‹'minspace'GB), recording stopped	NOTIFY	The free storage space available fell below the required threshold during recording, and storage was aborted.	
	failed to create 'filename' -› 'error'	STOP	It was not possible to create the file *filename*.	'error' indicates the cause of the error, e.g. write-protected, insufficient permissions/storage space.
	write to file 'filename' failed	NOTIFY	A write error occurred on the specified file during recording.	See above

Table 4-1. Error messages and causes

Module	Notification of errors	Severity	Cause	Solution
	'n' samples missing (device errors = 'd')	NOTIFY	A discontinuity was detected when checking the constant data counter. 'n' indicates the number of lost data points. “Device errors” indicates whether the data was lost in the amplifier hardware or during USB data transfer.	This error generally occurs in combination with “Input queue FULL”. If the “device error” counter › 0, it may be that other CPU processes with higher priority such as defragmentation, Windows® Update or Windows® Defender are interfering with the USB data transfer (see also Section 1.4 on page 19).

Appendix A Module-based structure of the PyCorder

Figure A-1. Overview of the PyCorder application: basic modules, basic functions and communication paths

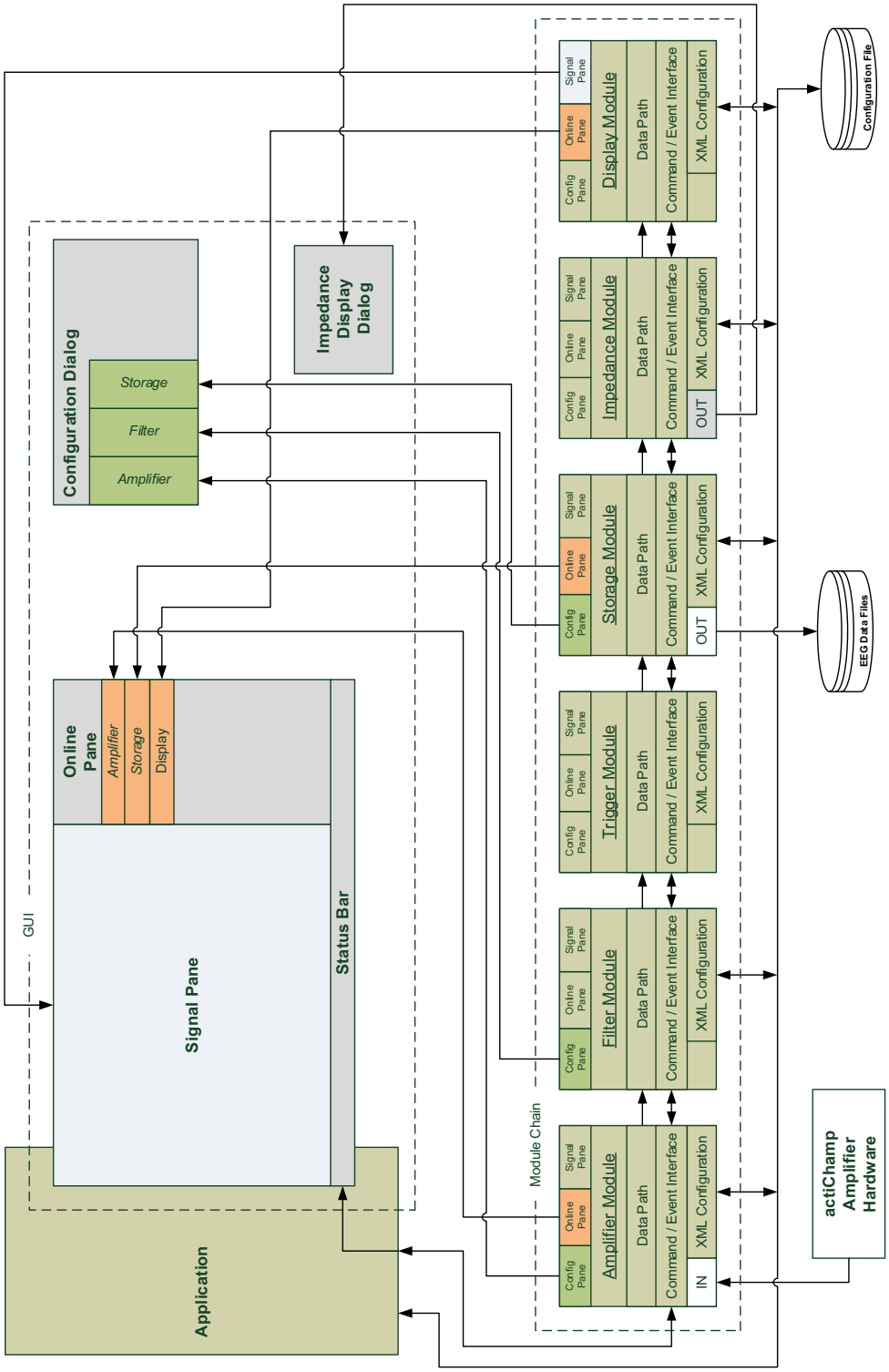


Figure A-2. Execution control and communication

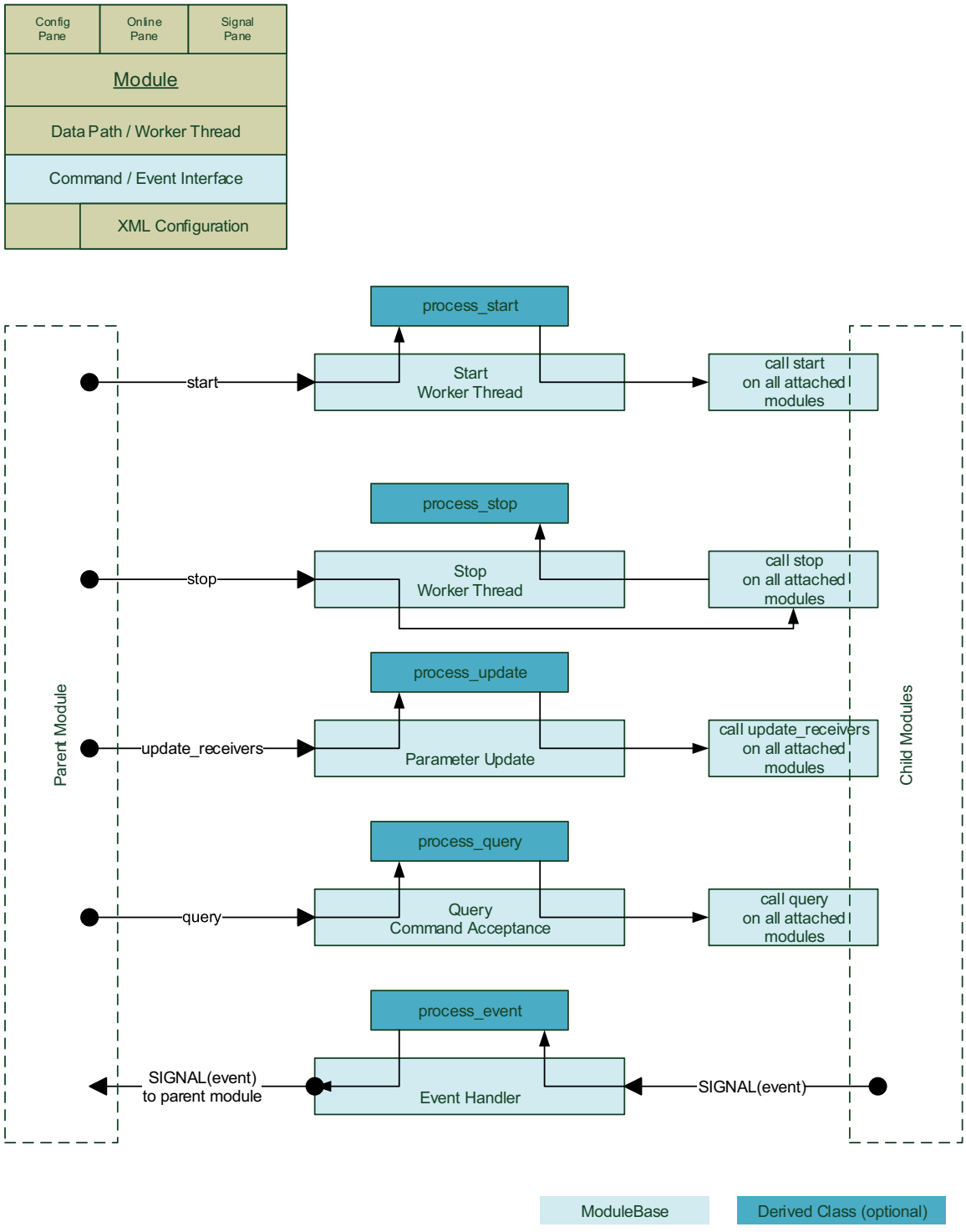


Figure A-3. Data processing

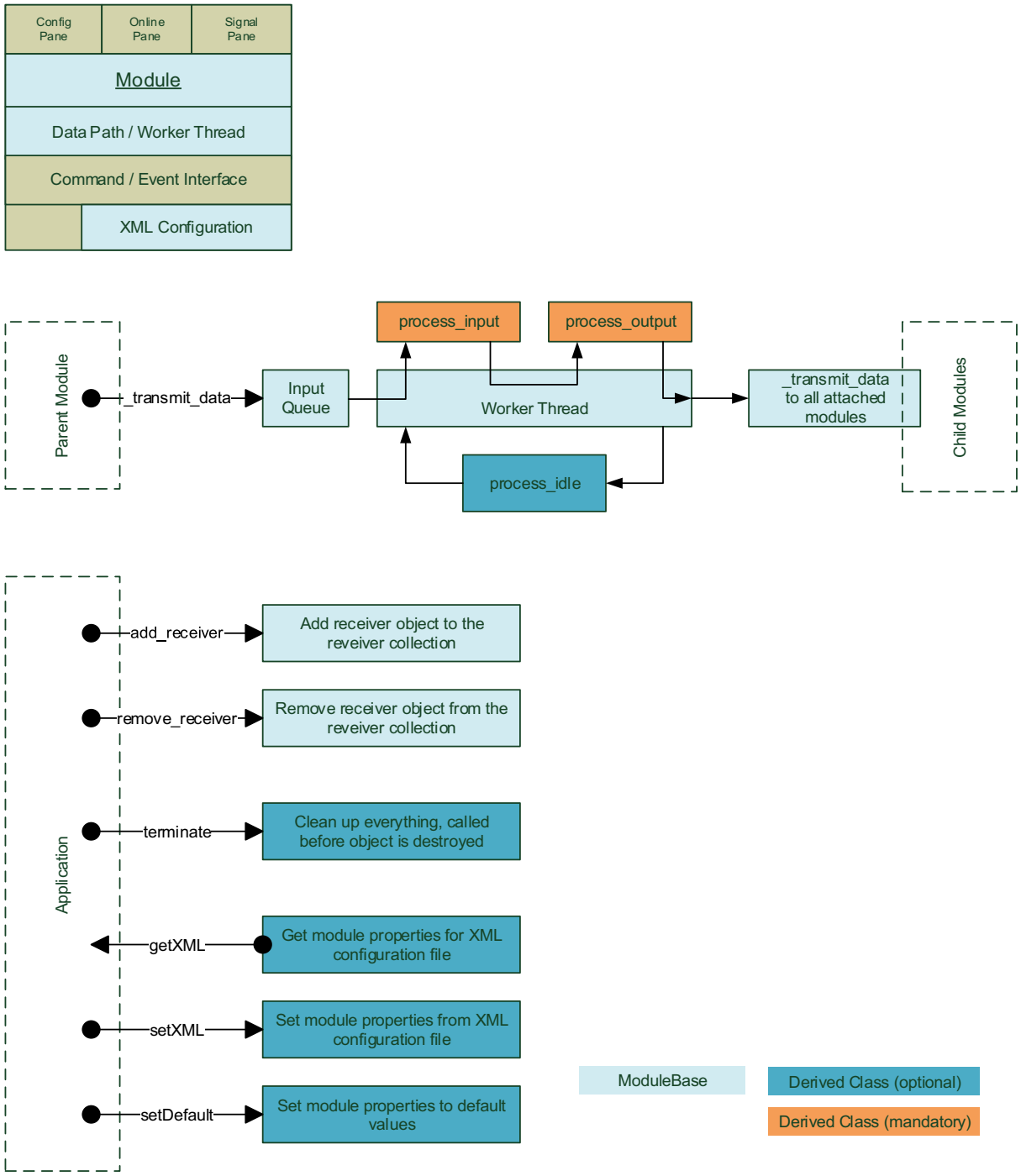
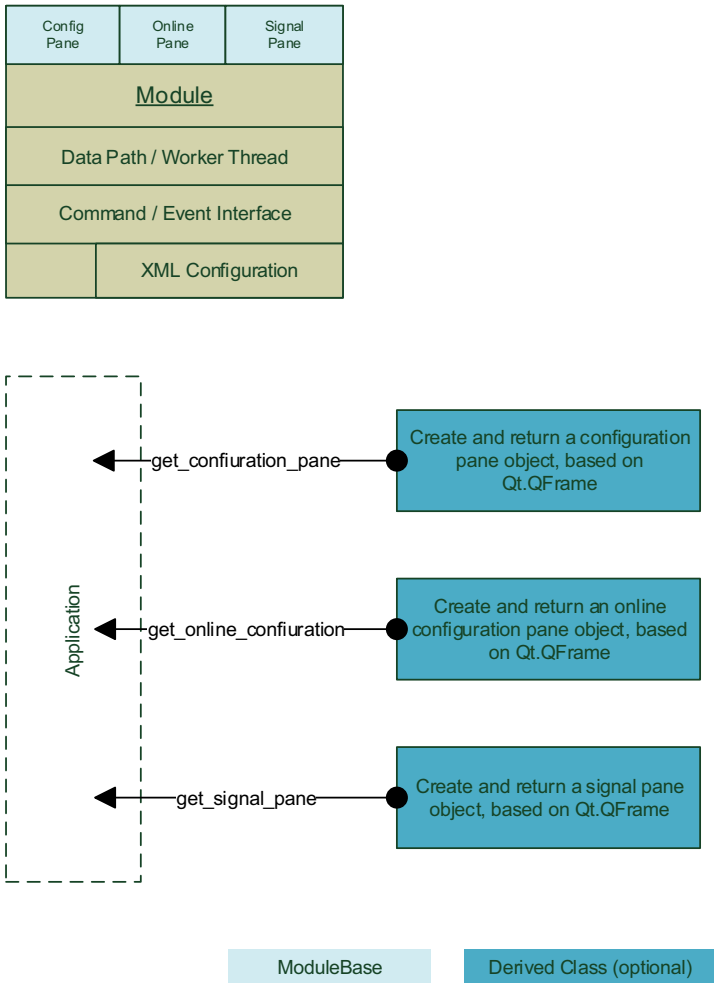


Figure A-4. Visualization and configuration



Appendix B Anti-aliasing filter

To provide for the hardware sampling rates of 100 kHz, 50 kHz and 10 kHz supplied by the *actiCHamp* amplifier, a 20 kHz (-3 dB) anti-aliasing filter (*actiCHamp* Rev. 01) or an 8 kHz (-3 dB) anti-aliasing filter (as of actiCHamp Rev. 02) is installed in the hardware ahead of the Sigma-Delta A/D converter. The decimators contain appropriate high-cutoff filters ( see also Appendix E of the *actiCHamp Operating Instructions*).

In addition, at these sampling rates, the signals pass through an FIR filter in the *actiCHamp* library with the Z-transform $H(z) = 0.5 + 0.5 * z^{-1}$.

The other sampling rates required by the *PyCorder* application are not supplied directly by the amplifier hardware but are generated by the software in the *actiCHamp* library. In this case, the signals pass through a four or three-stage CIC (Cascaded Integrator-Comb) anti-aliasing filter prior to the decimation of the sampling rates.

Table B-1. PyCorder anti-aliasing filter

Sampling rate (software)	**Sampling rate (hardware)**	**Decimation factor**	**Software anti-aliasing filter**	**Corner frequency[a] for actiCHamp Rev. 01**	**Corner frequency for actiCHamp Rev. 02**
100 kHz	100 kHz	1	FIR filter	15.2 kHz	7.5 kHz
50 kHz	50 kHz	1	FIR filter	10.3 kHz	6.6 kHz
25 kHz	50 kHz	2	CIC filter (4)[b]	6.5 kHz	4.9 kHz
10 kHz	10 kHz	1	FIR filter	2.5 kHz	2.4 kHz
5 kHz	10 kHz	2	CIC filter (4)	1.3 kHz	1.3 kHz
2 kHz	10 kHz	5	CIC filter (3)	530 Hz	530 Hz
1 kHz	10 kHz	10	CIC filter (3)	260 Hz	260 Hz
500 Hz	10 kHz	20	CIC filter (3)	130 Hz	130 Hz
200 Hz	10 kHz	50	CIC filter (3)	52 Hz	52 Hz

a. -3 dB, hardware and software filter combined

b. In parentheses: Number of CIC stages

Figure B-1. Frequency response of the anti-aliasing filter, sampling rate 10 kHz/200 Hz

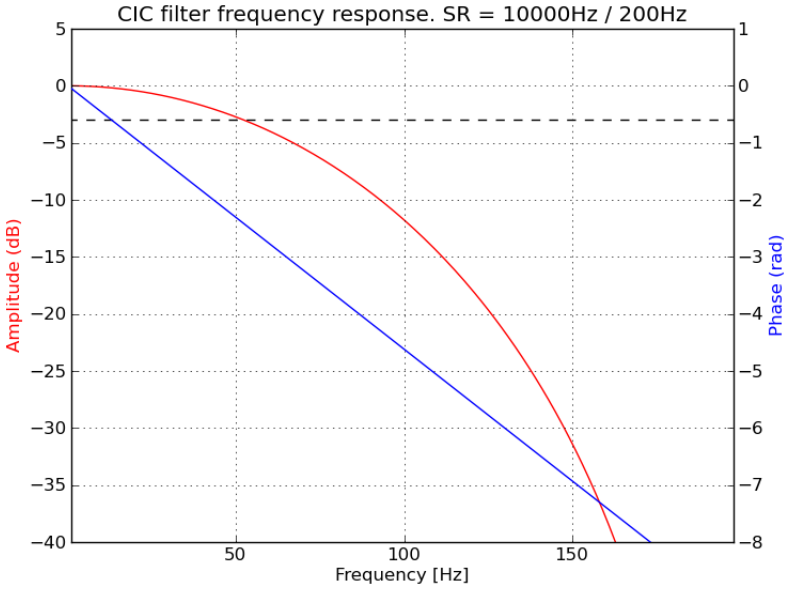


Figure B-2. Frequency response of the anti-aliasing filter, sampling rate 10 kHz/500 Hz

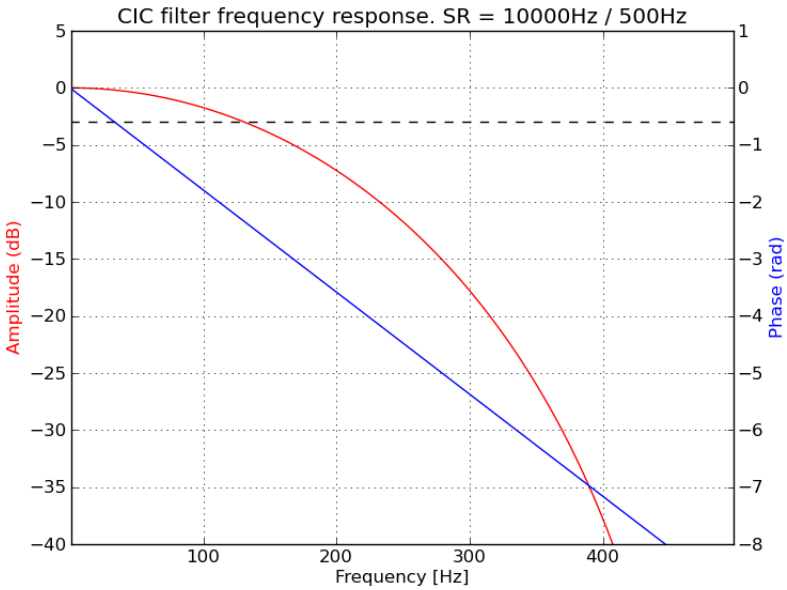


Figure B-3. Frequency response of the anti-aliasing filter, sampling rate 10 kHz/1 kHz

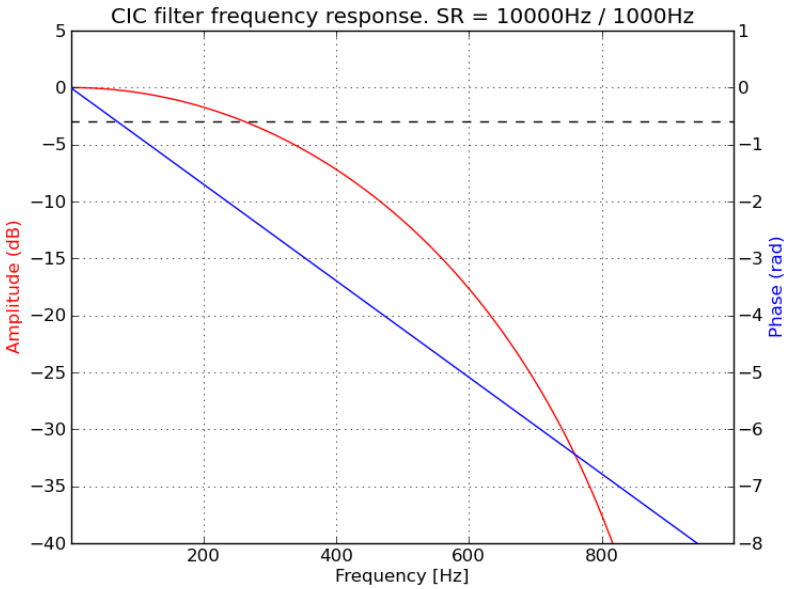


Figure B-4. Frequency response of the anti-aliasing filter, sampling rate 10 kHz/2 kHz

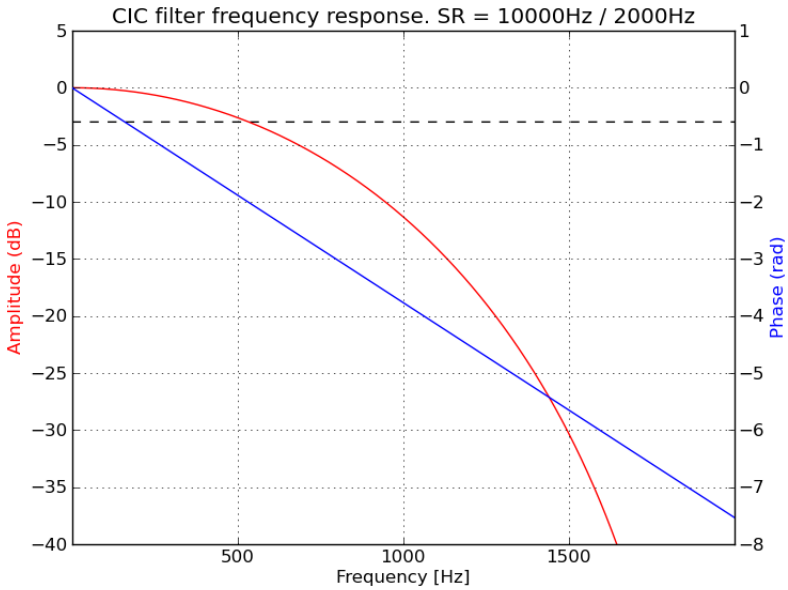


Figure B-5. Frequency response of the anti-aliasing filter, sampling rate 10 kHz/5 kHz

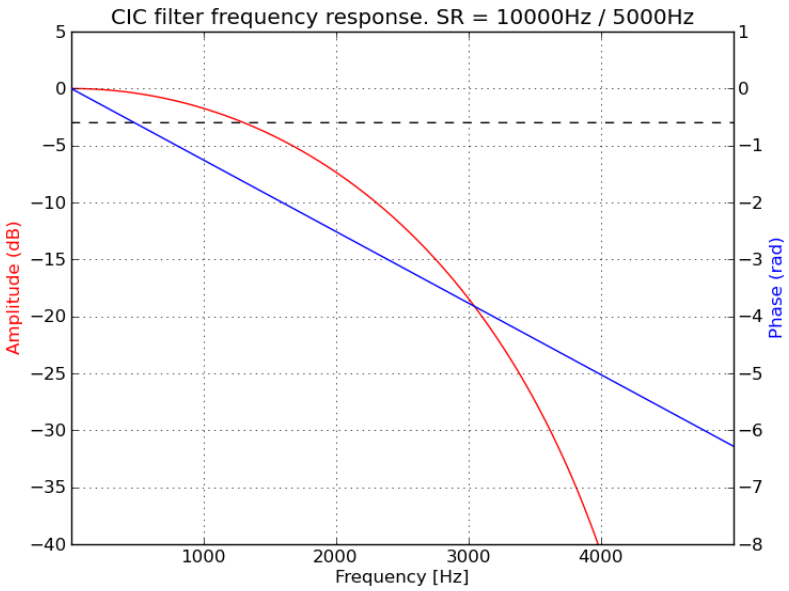
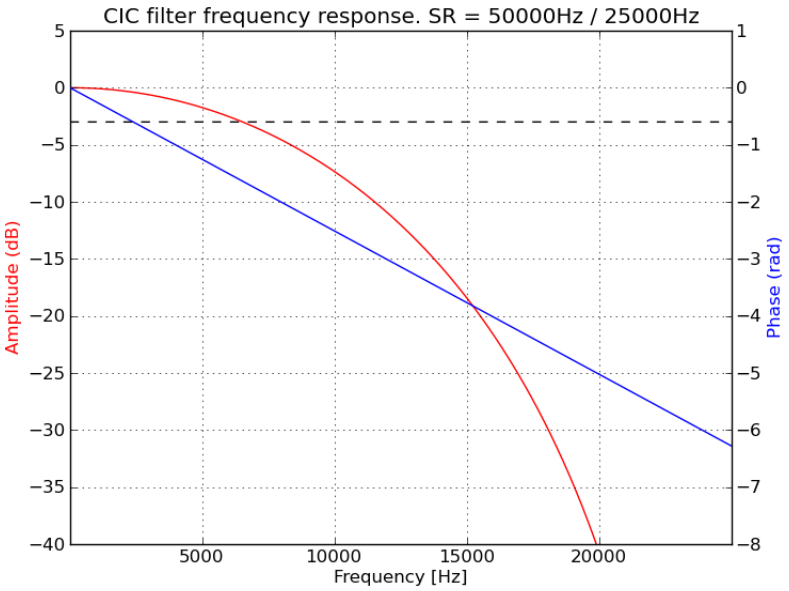


Figure B-6. Frequency response of the anti-aliasing filter, sampling rate 50 kHz/25 kHz



Appendix C EEG file formats

An EEG recording is saved in the following three files:

- Header file (*.VHDR)
- Marker file (*.VMRK)
- EEG raw data file (*.EEG)

The header file describes the setup and is configured as an ASCII file. It will normally be given the same base name as the raw data EEG that is described in it. The header file is stored in the raw data folder of the workspace.

Format of the header file

The format of the header file is based on the Windows® INI format and both this file and the marker file can be opened and edited using the Windows® Notepad. It consists of various named sections containing keywords/values. Below is an extract from a header file by way of an example:

```
Brain Vision Data Exchange Header File Version 1.0
; Data created by the actiCHamp PyCorder

[Common Infos]
Codepage=UTF-8
DataFile=Example_01.eeg
MarkerFile=Example_01.vmrk
DataFormat=BINARY
; Data orientation: MULTIPLEXED=ch1,pt1, ch2,pt1...
DataOrientation=MULTIPLEXED
NumberOfChannels=32
; Sampling interval in microseconds
SamplingInterval=100

[Binary Infos]
BinaryFormat=IEEE_FLOAT_32
```

```
[Channel Infos]
; Each entry: Ch<Channel number>=<Name>,<Reference channel name>,
; <Scaling factor in “Unit”>,<Unit>, Future extensions..
; Fields are delimited by commas, some fields might be omitted (empty).
; Commas in channel names are coded as “\1”.
Ch1=Ch1,REF,1.0,µV
Ch2=Ch2,REF,1.0,µV
Ch3=Ch3,REF,1.0,µV
Ch4=Ch4,REF,1.0,µV
Ch5=Ch5,REF,1.0,µV
Ch6=Ch6,REF,1.0,µV
Ch7=Ch7,REF,1.0,µV
Ch8=Ch8,REF,1.0,µV
Ch9=Ch9,REF,1.0,µV
Ch10=Ch10,REF,1.0,µV
Ch11=Ch11,REF,1.0,µV
Ch12=Ch12,REF,1.0,µV
Ch13=Ch13,REF,1.0,µV
Ch14=Ch14,REF,1.0,µV
Ch15=Ch15,REF,1.0,µV
Ch16=Ch16,REF,1.0,µV
Ch17=Ch17,REF,1.0,µV
Ch18=Ch18,REF,1.0,µV
Ch19=Ch19,REF,1.0,µV
Ch20=Ch20,REF,1.0,µV
Ch21=Ch21,REF,1.0,µV
Ch22=Ch22,REF,1.0,µV
Ch23=Ch23,REF,1.0,µV
```

```
Ch24=Ch24,REF,1.0,µV

Ch25=Ch25,REF,1.0,µV

Ch26=Ch26,REF,1.0,µV

Ch27=Ch27,REF,1.0,µV

Ch28=Ch28,REF,1.0,µV

Ch29=Ch29,REF,1.0,µV

Ch30=Ch30,REF,1.0,µV

Ch31=Ch31,REF,1.0,µV

Ch32=Ch32,REF,1.0,µV


[Comment]

PyCorder V1.0.6

Amplifier

actiCHamp (5001) SN: 11020002

Module 1  (5010) SN: 11020011

Module 2  (5010) SN: 11020015

Module 3  (5010) SN: 11020012

Module 4  (5010) SN: 11020014

Module 5  (5010) SN: 11020013

Version: DLL_18.13.04.11, DRV_03.04.01.146, CTRL_04.11.04.28, FP-
GA_44.00.00.00, DSP_06.11.05.25


Reference channel: none
```

The first line identifies the header file and is mandatory.

A semicolon at the beginning of a line identifies a free-text comment. This line is ignored. Blank lines are also ignored. A section is identified by a line with a heading enclosed in square brackets. The header extract above, for example, contains the “Common Infos” section. A header file can contain an unlimited number of sections.

The following lines contain keywords for the corresponding section and the associated values. A keyword may only occur once in a section. Its meaning depends on the section in which it occurs. There must not be a space before or after the equals sign. Most of the predefined keywords have specific values that are used by the Generic Data Reader.

The amplifier setup parameters are listed in the “Amplifier-Setup” section.

The various predefined sections with keywords, their meanings and default values are listed below.

Header file (*.VHDR) in detail

Table C-1. “Common Infos” section of the header file

This section contains general information on the EEG file.		
Keyword	*Meaning*	*Default value*
`DataFile`	Name of the EEG file. If the name does not contain a path, it is assumed that the EEG file is in the same folder as the header file. The placeholder $b can be used in the file name. It is replaced by the base name of the header file when the file is read in. Example: If the name of the header file is *Test.vhdr*, the entry `DataFile=$b-EEG.dat` is interpreted as `DataFile=Test-EEG.dat`.	None, a value must be specified.
`MarkerFile`	Optional marker file. The marker file contains a list of markers assigned to the EEG. If no path is specified explicitly, the marker file is searched for in the folder containing the header file. The format of the marker file is explained on page 85. The placeholder $b can be used in the file name.	–
`DataFormat`	Data format: `BINARY`	

`DataOrientation`	Data orientation. Possible values:
	`VECTORIZED`
	The file begins with all the data points of the first channel, followed by all the data points of the second channel, and so on.
	`MULTIPLEXED`
	All the channels come one after the other for every data point. In other words, the data structure is multiplexed.
		`MULTIPLEXED`

`DataType`	Data type. Possible values:
	`TIMEDOMAIN`
	The data is in the time domain.
	`FREQUENCYDOMAIN`
	The data is in the frequency domain.
		`TIMEDOMAIN`

`NumberOfChannels`	Number of channels in the EEG file.	None, a value must be specified.
`SamplingInterval`	Sampling interval. The interval is specified in µs in the time domain and in hertz in the frequency domain.	None, a value must be specified.

Table C-1. “Common Infos” section of the header file

This section contains general information on the EEG file.		
Keyword	*Meaning*	*Default value*
`Averaged`	This indicates whether the data set to be read in has been averaged. It is particularly relevant to the enabling and disabling of transforms in the *Analyzer's* **Transformations** menu. Possible values are: `YES` – Yes, the data set represents data that has been averaged. `NO` - No, the data set represents data that has not been averaged.	`NO`
`AveragedSegments`	Number of segments included in averaging. This value is only evaluated when “`Averaged=YES`” is set.	`0`
`SegmentData-Points`	If the data is segmented evenly, the number of data points per segment can be specified at this point.	`0`

SegmentationType	Segmentation type. Like `Averaged`, this variable is relevant to the enabling and disabling of transforms in the *Analyzer's* **Transformations** menu.
	Possible values are:
	`NOTSEGMENTED`
	The data set has not been segmented.
	`MARKERBASED`
	The data set has been segmented on the basis of one or more marker positions. All segments have the same length.
	`FIXTIME`
	Segmentation was based on fixed times. All segments have the same length.
		`NOTSEGMENTED`

`DataPoints`	Number of data points in the EEG file. If no predefined value has been specified, the data is read in up to the end of the file. In the case of binary data, the `TrailerSize` parameter in the [Binary Infos] section can be set as an alternative.	`0`
`Codepage`	Codepage used in the header file. Possible values: `UTF-8`, `ANSI`	`ANSI`

Table C-2. “ASCII Infos” section of the header file

This section is only relevant if ASCII is set for “DataFormat” in the “Common Infos” section.		
Keyword	*Meaning*	*Default value*
`DecimalSymbol`	Decimal character used in the EEG file. This symbol can be either a point or a comma. In the header file, the decimal symbol is always a point.	Point (.)
`SkipLines`	Number of header lines to be skipped	
`SkipColumns`	Number of columns to be skipped at the beginning of a line.	

Table C-3. “Channel Infos” section of the header file

Channel information. This section lists the individual channels and their properties.		
Keyword	*Meaning*	*Default value*
`Ch<x>`. x stands for the channel number. In other words, the keyword for the first channel is Ch1, for the second channel Ch2, etc.	Individual properties for the channel are specified separated by commas: ‹channel name›,‹reference channel name›, ‹resolution in “unit”›,[‹unit›] Example: Ch1=Fp1,,1 The first channel has the channel name Fp1. The common reference channel is taken as the reference channel because no entry has been made. The resolution is 1 µV. The resolution is the value by which the value of the data point is multiplied to convert it to µV or to the selected unit.	

Table C-4. “Binary Infos” section of the header file

This section is only relevant if BINARY is set for “DataFormat” in the “Common Infos” section.		
Keyword	*Meaning*	*Default value*

`BinaryFormat`	Binary format. Possible values:
	`IEEE_FLOAT_32`
	IEEE floating-point format, single precision, 4 bytes per value
	`INT_16`
	16-bit signed integer
	`UINT_16`
	16-bit unsigned integer
		`INT_16`

`ChannelOffset`	Channel offset at which the data starts. The offset is only relevant to vectorized data. `ChannelOffset` and `DataOffset` can be used simultaneously.	`0`
`DataOffset`	Size of the offset in the file at which the actual data starts.	`0`
`SegmentHeader-Size`	If the data is segmented evenly, the size of the segment header can be entered here in bytes.	`0`
`TrailerSize`	Size of the trailer of the EEG file in bytes. This parameter can be specified as an alternative to `DataPoints` in [Common Infos] in order to stop reading in the data before the end of the EEG file is reached.	`0`

`UseBigEndianOrder`	This only applies to integer formats. It specifies whether big Endian order is used, i.e. whether the most significant byte is stored first (Macintosh, Sun). Possible values are:
	`YES`
	Yes, big Endian order is used.
	`NO`
	No, little Endian order is used (corresponds to the Intel specification).
		`NO`

Marker file (*.VMKR) in detail

The marker file is based on the same principle of sections and keywords as the header file. The first line identifies the marker file, as follows:

```
Brain Vision Data Exchange Marker File Version 1.0
```

The various predefined sections with keywords, their meanings and default values are listed below.

Table C-5. “Common Infos” section of the marker file

This section contains general information on the marker file.		
Keyword	*Meaning*	*Default value*
`DataFile`	Name of the EEG file. If the name does not contain a path, it is assumed that the EEG file is in the same folder as the marker file. This information is not evaluated by the Generic Data Reader.	–

Table C-6. “Marker Infos” section

Marker information. The individual markers and their properties are listed in this section.		
Keyword	*Meaning*	*Default value*

`Mk<x>` “x” stands for the marker number. In other words, the keyword for the first marker is Mk1, for the second marker Mk2, etc.	Individual properties for the channel are specified separated by commas:
	‹type›,‹description›,‹position›,‹points›,‹channel number›,‹date›
	Example:
	`Mk1=Time 0,,26,1,0`
	The first marker in this example has the type “Time 0”, no description, its position is at data point 26, its length is 1 data point, and the channel number is 0, which means that this marker applies to all channels.
	The date is optional. It is only evaluated if the marker type is “New Segment”.
	The date has the following format:
	4 digits = year
	2 digits = month
	2 digits = day
	2 digits = hour (24-hour system)
	2 digits = minute
	2 digits = second
	6 digits = microsecond
	The result is a time resolution of a microsecond.
	Specifying a date
	`19990311140312000000`
	means 11 March 1999, 14:03:12.000000
		–