CONTENTS

1 User Guide 1
  1.1 Installation .................................................. 1
  1.2 How to use RNAseqViewer ............................... 2
  1.3 Supported data and graphs ............................... 12

2 Source Code Documentation 27
  2.1 RNAseqViewer module .................................. 27
  2.2 additionalUI package .................................... 27
  2.3 freeze module ............................................. 32
  2.4 functions module ......................................... 33
  2.5 generateDoc module ...................................... 35
  2.6 lang package ............................................... 35
  2.7 mainUI package ............................................ 36
  2.8 qrc_resources module .................................... 42
  2.9 session package .......................................... 43
  2.10 setup module ............................................. 55
  2.11 tracks package ........................................... 55

Python Module Index 75

Index 77
The User Guide is intended to help you to understand how to use RNAseqViewer. It introduces the main features of the program and describes the different types of supported data and the different types of graphs.

If you have any question about the program, please contact Xavier Rogé (rogex10@mails.tsinghua.edu.cn).

1.1 Installation

There are three ways to install RNAseqViewer. Binaries are provided for some configurations of Windows and Linux. If they don’t work, you can choose the third and installing from the source.

1.1.1 Windows Installer

If you use Windows, the easiest way to install RNAseqViewer may be to download (see the download page) and run the Windows Installer. You can choose some options like the path where to install the program. When installation is finished, you can start using RNAseqViewer.

*The Windows Installer was tested on Windows 7 SP1 and Windows XP pro SP3.*

1.1.2 Linux Binaries

Using Linux binaries can be a quick way to install RNAseqViewer on some Linux distributions. Just download (see the download page) and uncompress the binaries. You can directly run the program RNAseqViewer:

```
tar -xf RNAseqViewer-X.X.X-linux-i686.tar.gz
cd RNAseqViewer-X.X.X-linux-i686/
./RNAseqViewer
```

where X.X.X should be replaced by the latest version number (e.g. 0.7.2).

*Tested on Ubuntu 12.04 and OpenSUSE 12.2.*

There is a specific version of the binaries for Debian (see the download page):

---

1 rogex10@mails.tsinghua.edu.cn
2 http://bioinfo.au.tsinghua.edu.cn/software/RNAseqViewer/#download
3 http://bioinfo.au.tsinghua.edu.cn/software/RNAseqViewer/#download
4 http://bioinfo.au.tsinghua.edu.cn/software/RNAseqViewer/#download
1.1.3 Install from source

The last installation method is general purpose and should work on most systems.

You should first install Python \(^5\) and PySide\(^6\) (only core, gui and svg packages are needed). If you want to visualize SAM/BAM files, you should also install Samtools\(^7\) and if your plan to visualize Tabix-compressed GTF files, you should also install Tabix\(^8\). These programs may be present in your Linux distribution’s default package repository.

For example, on Debian, you should run:

```bash
apt-get install python3-all-dev python3-pyside samtools tabix
```

When you get these programs installed, you can download (see the download page\(^9\)) and uncompress the sources, and run RNAseqViewer.py with Python 3:

```bash
tar -xf RNAseqViewer-X.X.X.tar.gz
cd RNAseqViewer-X.X.X-linux-i686/
python3 RNAseqViewer.py
```

where X.X.X should be replaced by the latest version number (e.g. 0.7.2).

*Tested on Windows 7 and Ubuntu 12.04, with Python 3.2, PySide 1.1.0, Samtools 0.1.12 and Tabix 0.2.5.*

1.2 How to use RNAseqViewer

This section of the user manual is intended to explain how to use the different features and functionalities of the software.

1.2.1 Quick Start

Here are the few steps you may try when opening RNAseqViewer for the first time after installation (see Installation (page 1) page). The data used in this quick tutorial can be downloaded on the download page\(^10\).

First load some reference gene annotations. Loading datasets is made using the Datasets menu.

---

\(^5\)http://www.python.org/
\(^6\)http://qt-project.org/wiki/PySide
\(^7\)http://samtools.sourceforge.net/
\(^8\)http://samtools.sourceforge.net/tabix.shtml
\(^9\)http://bioinfo.au.tsinghua.edu.cn/software/RNAseqViewer/#download
\(^10\)http://bioinfo.au.tsinghua.edu.cn/software/RNAseqViewer/#download
You can choose to load the annotations dataset *hg18.txt* provided in the sample data package. The result should look like this:

The track list now has one item, corresponding to the newly added annotation track. The default position is the very
beginning of the track, which is \textit{chr1:1-32} on this example. You can either search for a gene, input coordinates or navigate along the track to see other regions (see \textit{Navigation} (page 9) page).

Let’s enter the gene name \textit{SLC25A6} in the search field.

Two annotations have been found in \textit{hg18.txt}. Choose the one on chromosome Y and click on the button “See >>”, then you can close the search dialog. Now we can see the gene \textit{SLC25A6} on chromosome Y. Some information appear in the “Items information” box when pointing on the gene.
Now we can add more data like through the “Datasets” menu. For instance, we can load the read alignment file SRR057629.bam and the splicing junctions file SRR057629-junctions.bed.

Icons in the “Track list” let you change the type of view of each track. You can also explore the different menus and
buttons to further customize your visualization.

1.2.2 Main Window

Starting window

When starting the program, the main window looks like the following picture:

The window can be divided into 4 zones:

- the main menu (1)
- the tool bar (2 to 6)
- the track list (7)
- the track zone (8)

Using the tool bar, you can:

- read and input genome coordinates (2)
- navigate along a chromosome (3)
- zoom the view (4)
- hide intronic regions (5)
- search a gene among loaded genome annotations (6)

Visualizing tracks

After you load some datasets using the “Datasets” menu (1) and navigate using buttons (3) and (4), the view looks like this:
The genome coordinates are now visible in the zone (2). Note that the coordinates of the mouse appear in the status bar on the bottom-right corner.

The track list now references two tracks in two different categories. Next to each track’s name, buttons let you change the type of graph. For example, the track “gencode-v7” can be displayed either in expanded view or in compact view, or it can be hidden. Buttons next to each category’s name let you change the view for all the tracks of the category. At the bottom of the list, four buttons can be used to add a track, change the settings of the selected track, duplicate the selected track and delete the selected track.

The track zone contains two tracks. The left part of the track shows the track name on a colored background, which correspond to the color in the track list. The first track shows genome annotations. As for most types of graph, you can see more information by pointing an element with your mouse for a few seconds. You can reorder the tracks by dragging them by their colored left part: click on the colored left part of one track and hold down, then move your mouse to move the track and release on the desired position.

Additional frames

The “Window” menu lets you open other frames to display more information.

When the “Items information” frame is opened, additional information are not displayed in tool tips anymore, but they are visible in the frame. Various identifiers link to their description on reference websites (http://www.ensembl.org, http://www.ncbi.nlm.nih.gov, http://vega.sanger.ac.uk). The following screenshot shows an example, when the mouse hovers over an annotation of the Gencode annotation dataset (version 15).
The “Legend” frame shows the link between colors and values for the tracks which use colors as a way to show scores like the FPKM (Fragments Per Kilobase of exon per Million fragments mapped). The legend appears when you point on a track using colors with your mouse.

Reorganizing the main window

The various frames be undocked or closed using the icons on the top right corner of the list. You can also move the frames to change their order, dock them on the left or the right part of the window, or place them on the top of each
other. The “Window” menu lets you choose which frames should be shown.

When the track list is closed, you can still access most of the track configuration options (change type of view, access track settings, etc.) by clicking on a track with the right button of your mouse. You can still re-open the track list and the other frames via the “Window” menu.

1.2.3 Navigation

Initial position

RNAseqViewer focus on RNA-Seq data at the gene level. It is not possible to visualize large portions of the genome at once. When the first dataset is loaded, the initial view position is at the beginning of the first chromosome or sequence of the dataset.

Panning

There are three ways to move the view along a chromosome. The first one is to drag the track: click on a track, hold down, and move your mouse horizontally. Alternatively, you can click on the navigation buttons of the tool bar. Eventually, you can use the keyboard shortcuts: Ctrl+Left, Left, Right and Ctrl+Right.

Zooming

Similarly, there a three ways to zoom in and out. You can either use your mouse’s wheel while pressing the Ctrl key, using the tool bar’s buttons or press Ctrl++ and Ctrl+-.

Direct access

Coordinate input

If you know the coordinates of the region you want to display, you can select the chromosome and type the coordinates in the two left fields of the tool bar.

Gene search

Alternatively, you can search the genes by name. First you should load a gene annotation dataset (“Dataset” menu). You’ll find the search tool in the “Edit” menu and in right part of the tool bar. Type a gene name and validate. Matching annotations are listed and you can click “See >>” to move the view to the gene’s location. Click “Close” when you don’t need the search tool anymore.
See exons only

In order to get a better view of data mapped to exons, you can hide the intronic regions by clicking on the icon of the tool bar. This option is only available after you first add a genome annotation dataset.

1.2.4 Sessions management

How to handle sessions

When you open RNAseqViewer, a new session will be started. This means you will start with a blank track list and no dataset is loaded yet. You can then add datasets, navigate along the data and customize the tracks.

To avoid having to repeat these operations if you want to visualize the same data a second time, you can save the session through the File menu. This means that all the settings about the loaded datasets, the way tracks are displayed and the current location will be saved in a session file, with the extension .rsv.

If you want to visualize again the data from a previously saved session, you can open the session through the File menu. This will load the datasets and restore the appearance of the software to the same state as when you saved the session.

Session files

Session files are XML files with the extension .rsv. They are designed to be easily edited or created by humans.

The root of the file is a <session> element. It should contain a <datasets> element, which will contain all the loaded datasets. Each dataset is a <dataset> element, which should have two attributes:

- The attribute file should contain the path to the data file. It is possible to mention a URL (over HTTP or FTP) to a distant file, which will be downloaded in the downloaded directory located in the installation path of RNAseqViewer.
• The attribute `type` should contain the name of the type of dataset, among the following list:
  – `sequence` for DNA sequences (Fasta files)
  – `annotations` for genome annotations (RefFlat or GTF files)
  – `reads` for read alignments (SAM/BAM files)
  – `bed` for BED annotations (BED files)
  – `junctions` for splicing junctions (BED files)
  – `transcripts` for transcriptomes (GTF files)
  – `numeric` for numeric data (Wiggle files)

The tracks for each dataset should be defined with `<track>` element inside the a `<dataset>` element. The `<track>` element has the following attributes:

• `name`: the name of the track
• `color`: the color of the handle of the track, in the hexadecimal format (#XXXXXX)
• `viewtype`: the type of view of the track, depending on the type of dataset
• `height` (optional): the height of the track in pixels
• `order` (optional): the rank of the track with regards to the other tracks (0 for the first track, 1 for the second track, etc.)

The information about the view’s settings are given in the `<view>` element, which is a direct child of the `<session>` element. It can contains several optional sub-elements:

• the `<position>` element, with the coordinates of the current view given in its three attributes: `chr, start` and `end`.
• the `<window>` element with the information about the size and the organization of the window’s elements. The `<window>` element contains two attributes: `geometry` and `state`, which are encoded strings.
• the `<introns>` element, which indicates whether introns should be hidden or not. This is given in its `hidden`, which should be either `True` or `False`.

Here is an example of session file:

```xml
<?xml version="1.0" ?>
<session>
  <datasets>
    <dataset file="D:\data\hg18\hg18.txt" type="annotations">
      <track color="#FB6F96" height="145" name="hg18" order="1" viewtype="expanded"/>
    </dataset>
    <dataset file="D:\data\accepted_hits.bam" type="reads">
      <track color="#FFD6F9" height="120" name="accepted_hits" order="2" viewtype="coverage"/>
      <track color="#5687AE" height="145" name="accepted_hits" order="3" viewtype="reads"/>
    </dataset>
    <dataset file="D:\data\hg18\hg18.fa" type="sequence">
      <track color="#42EF51" height="25" name="hg18" order="0" viewtype="sequence"/>
    </dataset>
  </datasets>
  <view>
    <window geometry="AdnQywABAAAA(...)AA==" state="AAAA/wAAAAAD9(...)AA"/>
    <position chr="chr1" end="97352" start="1261"/>
    <introns hidden="True"/>
  </view>
</session>
```
1.2.5 Example workflow

This section of the user manual is intended to provide some help about how to get the data which can be visualized in RNAseqViewer. The different types of data are described in Supported data and graphs (page 12).

Get reference genome files

At first you should obtain the sequence and optionally some annotations of your reference genome. This can be found on some websites.

For example, if you study the human genome, you can use the reference genome hg19, which sequence can be downloaded here\(^{11}\) and RefSeq annotations can be downloaded here\(^{12}\) on UCSC website.

Aligned reads

As a result of a sequencing, you can obtain a fastq file, which contains the sequence and sequencing quality of sequenced reads. Before the reads can be visualized on RNAseqViewer, they should first be aligned, using a tool like TopHat or SpliceMap.

If you want to use TopHat, you will need a Bowtie index. Pre-built indexes can be found at this page of Bowtie’s website\(^{13}\). Then you can run tophat like this:

tophat bowtie_indexes/hg19 SRR490122.fastq

where bowtie_indexes/hg19 is the path to the directory which contains your Bowtie index and SRR490122.fastq is the file containing the reads you want to align. Please refer to TopHat user manual for more details.

TopHat will output two files which can be visualized by RNAseqViewer: the file containing the aligned reads, named accepted_hits.bam and the file describing the splicing junctions, named junctions.bed.

Transcript assembly

RNAseqViewer can also display transcripts in GTF format. You can obtain the transcriptome of a sequencing experiment using assemblers like Cufflinks, for example:

cufflinks accepted_hits.bam

where accepted_hits.bam is the reads alignment file you got from the previous step. Please refer to Cufflinks user manual for more details.

1.3 Supported data and graphs

The program can handle seven types of data:

\(^{11}\)http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz
\(^{12}\)http://hgdownload.cse.ucsc.edu/goldenPath/hg19/database/refFlat.txt.gz
\(^{13}\)http://bowtie-bio.sourceforge.net/tutorial.shtml#preb
1.3.1 DNA Sequences

Supported file format

Only file in FASTA format can be read. For example, you can use the public genome sequences provided by UCSC (hg18<sup>14</sup>, hg19<sup>15</sup> and other sequences<sup>16</sup>) or by Ensembl (GRCh37 release 70<sup>17</sup> and other<sup>18</sup>). When loading a FASTA file sequence.fa, RNAseqViewer will look for the index file sequence.fa.fai in the same folder. If it doesn’t exist, it will be created.

DNA Sequence View

There is only one type of view, but the view changes according to the scale:

- At large scale, the nucleotides are represented by their letter (A, C, G, T), with a different background for each type of nucleotide.
- At medium scale, only the background is shown.
- At small scale, the nucleotides can’t be shown.

![Figure 1.1: A DNA sequence at large scale](image)

![Figure 1.2: A DNA sequence at medium scale](image)

![Figure 1.3: A DNA sequence at small scale](image)

1.3.2 Genome Annotations

Supported file format

RNAseqViewer can read annotations in two different formats: RefFlat and Gene Transfer Format (GTF).

RefFlat is used for example for RefSeq annotations provided by UCSC<sup>19</sup>. The format consists in an eleven-column tab-delimited file, with one line per annotation (geneName, name, chrom, strand, txStart, txEnd, cdsStart, cdsEnd, exonCount, exonStarts, exonEnds).

GTF format is also supported. For faster loading in future sessions and faster gene search, RNAseqViewer will ask you the authorization to create an index file. The index file will be created in the same directory as the GTF file.

<sup>14</sup>http://hgdownload.cse.ucsc.edu/goldenPath/hg18/bigZips/chromFa.zip
<sup>15</sup>http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz
<sup>16</sup>http://hgdownload.cse.ucsc.edu/downloads.html
<sup>17</sup>ftp://ftp.ensembl.org/pub/release-70/fasta/homo_sapiens/dna/
<sup>18</sup>http://www.ensembl.org/info/data/ftp/index.html
<sup>19</sup>http://genome.ucsc.edu/
Alternatively, you can use the compressed GTF format. The original GTF file should be sorted, compressed with bgzip and indexed with Tabix\textsuperscript{20}, as in the following example:

\begin{verbatim}
(grep "^\#" in.gtf; grep -v "^\#" in.gtf | sort -k1,1 -k4,4n) | bgzip > sorted.gtf.gz;
tabix -p gff sorted.gtf.gz;
\end{verbatim}

Compressed GTF files are faster for display operation, but are slower for searching a gene and for hiding intronic regions.

You can download human annotations on UCSC’s website in RefFlat format (hg18\textsuperscript{21} and hg19\textsuperscript{22}). Ensembl’s website offers annotations for various species in GTF format (human genome\textsuperscript{23} and others\textsuperscript{24}). Human genome annotations can be downloaded from the Gencode Project website\textsuperscript{25}

Annotations can be displayed with two alternative types of view: \textit{Collapse View} (page 14) and \textit{Expanded View} (page 14).

\textbf{Collapse View}

The Collapse View shows all the annotations on the same line. Thick lines represent exons and thin lines represent introns. The name of the gene is shown under the line and the other data are shown in a tool tip when hovering over the annotation or the name. Note that a version of the tool tip with links to reference websites can be activated (see Additional frames (page 7))

\textbf{Expanded View}

The Expanded View shows the annotations on several lines so as to avoid them to collapse.

![Figure 1.4: The annotations from the \textit{hg18} data set is displayed twice. The upper track shows the \textit{Collapse View} (page 14) and the lower track shows the \textit{Expanded View} (page 14).](image)

\textsuperscript{20}http://samtools.sourceforge.net/tabix.shtml
\textsuperscript{21}http://hgdownload.csc.usc.edu/goldenPath/hg18/database/refFlat.txt.gz
\textsuperscript{22}http://hgdownload.csc.usc.edu/goldenPath/hg19/database/refFlat.txt.gz
\textsuperscript{23}ftp://ftp.ensembl.org/pub/release-70/gtf/homo_sapiens
\textsuperscript{24}http://www.ensembl.org/info/data/ftp/index.html
\textsuperscript{25}http://www.gencodegenes.org/
1.3.3 BED Tracks

Supported file format

You can visualize data in BED format. The format is described by UCSC: BED Format[^26]. The twelve columns of the format are taken into account by RNAseqViewer, although only the three first ones are mandatory (chrom, chromStart and chromEnd).

Like Genome Annotations (page 13), BED tracks can be displayed with two alternative types of view: Collapse View (page 15) and Expanded View (page 15).

Collapse View

The Collapse View shows all the annotations on the same line. Thick lines represent exons and thin lines represent introns. The name is shown under the line and the other data are shown in a tool tip when hovering over the annotation or the name.

Expanded View

The Expanded View shows the annotations on several lines so as to avoid them to collapse.

[^26]: http://genome.ucsc.edu/FAQ/FAQformat#format1
Figure 1.6: The program displays the same BED track twice. The upper track shows the **Collapse View** (page 15) and the lower track shows the **Expanded View** (page 15). The track is one of the examples provided by UCSC (http://genome.ucsc.edu/FAQ/FAQformat#format1):

```
chr7  127471196  127472363  Pos1  0  +  127471196  127472363  255,0,0
chr7  127472363  127473530  Pos2  0  +  127472363  127473530  255,0,0
chr7  127473530  127474697  Pos3  0  +  127473530  127474697  255,0,0
chr7  127474697  127475864  Pos4  0  +  127474697  127475864  255,0,0
chr7  127475864  127477031  Neg1  0  -  127475864  127477031  0,0,255
chr7  127477031  127478198  Neg2  0  -  127477031  127478198  0,0,255
chr7  127478198  127479365  Neg3  0  -  127478198  127479365  0,0,255
chr7  127479365  127480532  Pos5  0  +  127479365  127480532  255,0,0
chr7  127480532  127481699  Neg4  0  -  127480532  127481699  0,0,255
```
1.3.4 Reads Alignments

Supported file formats

Reads alignment can be visualized from files in SAM or BAM format. The specification of the two formats have been defined by the SAM Format Specification Working Group, which publishes an online version. The SAM format (Sequence Alignment/Map) is a tab-delimited format, with each line describing a sequence and how it is mapped to the reference. The BAM format is the binary counterpart of SAM, which is usually used to save disk space.

The inner part of RNAseqViewer can only read sorted and indexed BAM files. If SAM files are provided or if the file is not sorted and/or not indexed, RNAseqViewer will offer you to convert, sort and/or index your file before displaying it. These processes can take some time, but they only need to be executed once.

BAM files are usually generated by RNA mappers like TopHat. Single end reads and pair end reads are both supported.

Reads have three type of view: Reads View, Read Coverage View and Heatmap View.

Reads View

The Reads View shows the individual reads mapped to the genome. If the scale is large enough, the nucleotide content of each read is shown. If the reference DNA sequence is also provided, only the nucleotides which don’t match the reference sequence are shown, while the other ones remain grey.

The information about a read and is alignment is shown in tool tips when you hover over the read with your mouse. At the same time the read’s color changes to red and, if it is a paired-end read, its mate is also shown in red.

Read Coverage View

Alternatively, you can choose to see the read coverage. The scale of the plot is dynamically adjusted. You can choose in the Settings (Edit menu) whether you want all the plots to share a common scale or you prefer each one to have its own scale. The upper limit of the scale is displayed in the top left-hand corner of the track.

Heatmap View

The Heatmap View is particularly useful to display a large number of tracks since it can be very compact. It shows the gene expression for each exon of some genome annotations.

The view is divided according the limits of the exons in the loaded annotations. Hence columns are created for each exon or zone of overlapping exons. Each track shows colored bands under each exon and the color depends on the FPKM for the given exon and the given sample. So the view should be regarded as a FPKM heat map, with columns representing exons and lines representing samples.

Defining the heatmap’s columns

Before choosing the Heatmap view, you should first add some annotations in order to define the columns of the heat map. These can be BED Tracks, Genome Annotations or Transcripts. If several annotation tracks are loaded, you can see and change the choice of the annotations which are used for the heat map. Therefore open the track settings (right click on the track) and select the annotation file(s) you want to take as reference.

---

27http://samtools.sourceforge.net/SAM1.pdf
28http://tophat.cbcb.umd.edu
Figure 1.7: An example of the Reads View (page 17). The reads are displayed as grey rectangles. The reference sequence has also been loaded, and the mismatching nucleotides are shown in color. The reads on the left are mapped to a splicing junction. Only the right part of the junction reads are shown, the left part is at the other end of the linking line toward right.

Note: One column is usually defined by one exon. This behavior is affected when several exons overlap. In this case, more columns are added to represent the overlapping sections. For example, if an exon A and an exon B overlap partially, then there will be three columns: 1) exon A without exon B, 2) overlapping region of A and B, and 3) exon B without exon A.

When your cursor hovers over a heatmap cell, more information is shown and the region corresponding to the column is highlighted on the annotation track.

Heatmap’s configuration

In default configuration, the color depends on $\log_{10}(\text{FPKM} + 1)$, but this can be changed in the track settings (right click on the track). The global settings window (Edit menu) offers two other configuration choices:

- Whether the color range should be common to all the tracks or independent for each track. The latter option can be useful for samples with different coverage depth.
- Which color scheme should be used, depending on your needs (intensity or diverging datasets, color-blind friendly, black & white printing-friendly, etc.)

Note: The colors range is dynamically adjusted to represent the data currently on the screen. So the color of a given exon might change if you scroll or zoom since the range might change to take into account other exons. For the best result, you should zoom to see only the gene of interest on your screen.
Figure 1.8: Another example of the *Reads View* (page 17) at smaller scale. The read under the mouse and its pair match are shown in red. Additional data are shown in a tool tip.
Figure 1.9: An example of the Read Coverage View (page 17). 8 BAM files have been loaded and the coverage of the samples are shown with a common scale: every track’s y axis range from 0 to 5,034.

Figure 1.10: An example of the Heatmap View (page 17). 3 BAM files have been loaded and the coverage of the exons is shown as a heatmap. Colors represent expression of the samples (described by the lines) for the exons of the HDAC2 gene which is shown on top. Colors are computed $\log_{10}(\text{FPKM} + 1)$. 

Figure 1.11: This example is very similar to the previous one. With regards to the latter, more datasets have been added and intronic regions have been hidden in order to have a better view of the expression for small exons. Colors of the datasets have also been changed to reflect the two groups: case samples and control samples.
1.3.5 Splicing Junctions

Supported file format

The file format for junctions handled by RNAseqViewer is the same as the output `junctions.bed` computed by the RNA mapper TopHat\(^{29}\), i.e. a special form of BED format\(^{30}\). Each line of the BED file describes a junction and “consists of two connected BED blocks, where each block is as long as the maximal overhang of any read spanning the junction. The score is the number of alignments spanning the junction.”\(^{31}\)

RNAseqViewer can display the junctions in two alternative types of view: Broken Line View (page 22) and Bar View (page 22).

**Broken Line View**

The Broken Line View displays the junctions in the usual way, i.e. as lambda-shape broken lines. The color of the lines depends on the number of reads mapped to the junction. The number is also given in a tool tip when your cursor hovers over a junction.

**Bar View**

The bar view is very close to the way BED Tracks (page 15) are usually displayed, except for the color which is derived from the number of matching reads like for the Broken Line View (page 22).

![Figure 1.12: The screenshot shows one annotation track (hg18) and two identical junction tracks, in Broken Line View (page 22) (top) and Bar View (page 22) (bottom).](image)

\(^{29}\)http://tophat.cbcb.umd.edu
\(^{30}\)http://genome.ucsc.edu/FAQ/FAQformat#format1
\(^{31}\)http://tophat.cbcb.umd.edu/manual.html
1.3.6 Transcripts

Supported file format

Transcripts can be visualized in RNAseqViewer in GTF2 format\(^{32}\) (Gene Transfer Format). Note that GTF is the output format for Cufflinks\(^{33}\) and RNAseqViewer can read the optional attributes of GTF format produced by Cufflinks, which are described in Cufflinks’s manual\(^{34}\).

Collapse View and Expanded View

Like BED Tracks (page 15), transcripts can be shown in an expanded view (default) or a collapse view if you want to display transcripts in a more compact way.

In default configuration, the color of the transcripts depends on the FPKM. This can be changed to $\log_{10}(\text{FPKM} + 1)$ or $\log_2(\text{FPKM} + 1)$ in the track settings (right click on the track).

More information about the transcripts found by Cufflinks are provided in tool tips.

1.3.7 Numeric data

Supported file format

Using RNAseqViewer, you can plot numeric data from files in Wiggle (WIG) format\(^{35}\). Both the variable step variant and the fixed step variant are supported.

Here is an example for the variable step variant:

```plaintext
variableStep chrom=chr17 span=5
56881 37.25
56891 14.9
56901 54.7
```

\(^{32}\)http://mblab.wustl.edu/GTF2.html
\(^{33}\)http://cufflinks.cbcb.umd.edu/
\(^{34}\)http://cufflinks.cbcb.umd.edu/manual.html#cufflinks_output
\(^{35}\)http://genome.ucsc.edu/goldenPath/help/wiggle.html
Figure 1.14: This example shows the transcripts assembled by Cufflinks (lower tracks). For example, the gene CUFF265 has two isoforms: CUFF265.1 and CUFF265.2. The mouse hovers over the transcript CUFF265.2 and some data about it are shown in a tool tip. The reads and junction reads used by Cufflinks for the assembly are shown in the first two tracks.

56911 58.99
56921 49.48

The following example is an equivalent in fixed step variant:

```bash
fixedStep chrom=chr17 start=56881 step=10 span=5
37.25
14.9
54.7
58.99
49.48
```

**Histogram view**

The data in Wiggle format are shown as histograms. The maximum value in the current area is shown on the top left-hand corner of the track. If scale is too small, values which span the same pixel are grouped and only their average value is kept. To see the individual values you should zoom in. The exact numeric value is at a current position is shown in tool tips.
Figure 1.15: One example of the representation of numeric data. The maximum value for the current region is 561 and the value for the position under the mouse is 289.86.
2.1 RNAseqViewer module

This is the entry file of the program. If called with the argument profile, the program will output profile information.

RNAseqViewer.main()
    Launch the application with a mainUI.mainWindow.MainWindow (page 40)

RNAseqViewer.my_excepthook (exctype, value, trace)
    Handle exceptions and log them.

2.2 additionalUI package

This package provides user interfaces that are independent from the main UI.

Content:

2.2.1 about module

This module defines the “About” box.

Classes:

About class

class additionalUI.about.About (parent)
    Bases: PySide.QtGui.QDialog1

    __init__ (parent)
        Pop up a dialog displaying information about the program.

1http://srinikom.github.com/pyside-docs/PySide/QtGui/QDialog.html#PySide.QtGui.QDialog
2.2.2 dialog module

This module defines dialogs to handle messages and questions.

```python
additionalUI.dialog.messageHandler(messageType, message, parent=None)
```

Handler for error messages, which displays the error message in a pop-up dialog and also displays the traceback. If `parent` is specified, it is used as the parent of the message box.

This function is intended to be used with `qInstallMsgHandler()`.

The function does nothing if `Dialog.classParameters['messageHandlerEnabled']` is `False`.

Classes:

**Dialog class**

class additionalUI.dialog.Dialog

```python
classes PySide.QtCore.QObject
```

This static class is used for basic interaction with the user. It should be first installed with `setWindow()` (page 28). Questions can be asked to users using `accept()` (page 28) and information given through the status bar using `setStatus()` (page 28).

```python
classParameters = {'window': None, 'messageHandlerEnabled': True, 'progressBar': None}
```

```python
static setWindow(window)
```

Set the window to be the parent of message boxes.

```python
static accept(question, details='', title='Question')
```

Ask the question `question` and return `True` if the user’s answer is `yes` or `False` if the user’s answer is `no`. The argument `title` specifies the title of the question box.

```python
static notify(message)
```

Show a notification message to users.

```python
static setStatus(message='', timeout=0)
```

Set the given `message` in the status bar for the given number of milliseconds.

```python
static setProgress(minVal=None, maxVal=None, currVal=None)
```

Add a progress bar to the status bar with the given parameters. If the parameters are `None`, the progress bar is hidden.

```python
static getSavePath(title, formats)
```

Open a dialog box to let the user choose the path for saving a file in one of the given `formats`. The allowed `formats` should be given as a list of 2-element tuples, with the first elements being the names of the formats and the second elements being the associated extensions. Return the selected path and format name, or `None` if the user cancels the operation.

```python
static getSaveDir(title, message, defaultDir=None)
```

Open a dialog box to let the user choose the folder for saving a file. Return the selected directory or `None` if the user cancels the operation.

```python
static error(message)
```

Show the given error `message`.

---

**Note:** This is similar to Qt’s `qCritical()`, but it supports unicode.

---

1 http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
**static getOpenFileName** *(caption, directory, filters, selectedFilter= None)*
Returns an existing file selected by the user.

### 2.2.3 globalSettings module

This module manages the modifications of the global settings for the program.

**Classes:**

**GlobalSettings class**

```
class additionalUI.globalSettings.GlobalSettings(window):
    Bases: PySide.QtGui.QDialog
```

Dialog to set the settings of a dataset.

```
__init__(window)
Initialize the settings dialog and write its content
```

```
buttonClicked(button)
Take into account the click on the given button.
```

```
applySettings()
Save and apply the settings in the box.
```

```
static setDefaultSettings()
Set the default settings to the parameters which are not set yet. This method has no effect on settings that are already set.
```

```
restoreDefaults()
Restore default settings after user confirmation.
```

**SettingsPage class**

```
class additionalUI.globalSettings.SettingsPage(settingsDialog, title, menuTitle=None):
    Bases: PySide.QtGui.QWidget
```

One page of the settings dialog.

```
__init__(settingsDialog, title, menuTitle=None)
Initialize the page and add it to the given settings dialog.
```

```
addLine(text, formItem)
Add a new field to the form. *text* is used for the label of the field and the input element (text input, combo box, etc.) is given in *formItem*.
```

```
addViewTypeLine(graphClass)
Add a field to choose the default type of view. Return the created combo box (PySide.QtGui.QComboBox).
```

```
addHeatmapTypeLine(graphClass, label)
Add a field to choose the default type of value to compute heatmap colors. Return the created combo box (PySide.QtGui.QComboBox).
```

---

3 [http://srinikom.github.com/pyside-docs/PySide/QtGui/QComboBox.html#PySide.QtGui.QComboBox](http://srinikom.github.com/pyside-docs/PySide/QtGui/QComboBox.html#PySide.QtGui.QComboBox)
4 [http://srinikom.github.com/pyside-docs/PySide/QtGui/QComboBox.html#PySide.QtGui.QComboBox](http://srinikom.github.com/pyside-docs/PySide/QtGui/QComboBox.html#PySide.QtGui.QComboBox)
2.2.4 printOut module

This module defines the class `PrintOut` (page 30) which manages the printing operations of the tracks.

Classes:

PrintOut class

class `PrintOut` additionalUI.printOut

Bases: `PySide.QtCore.QObject`\(^7\)

This class defines the methods for printing out the tracks, including physical printer, PDF export and image export. The method `initialize()` should be initialized before any call to the other methods.

```python
__init__ (window)
```

Initialize the printers.

```python
previewPrint ()
```

Open a print preview dialog.

```python
printTracks ()
```

Print out the tracks.

```python
saveAsImage ()
```

Save the screen shot of the tracks in an image.

```python
paintTracks (printer=None, output='printer', noHeader=False, svgPath=None)
```

Paint the track. The behavior varies according to the `output`. If `output` is `'printer'`, the tracks are painted on the given `printer`. If `output` is `'svg'`, an SVG file is saved on the given path (`svgPath`). If `output` is `'image'`, an instance of `QImage` is created, the tracks are painted, and the instance is returned by the method. Except if `noHeader` is set to `True`, some headers will be added.

Note: There should be at least one track.

2.2.5 search module

This module defines the search engine interface.

Classes:

SearchInput class

class `SearchInput` additionalUI.search

Bases: `PySide.QtGui.QLineEdit`\(^8\)

Text field for search input.

```python
focusInEvent (event)
```

Update the completer of the search field with logged requests.

---

\(^7\)http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject

\(^8\)http://srinikom.github.com/pyside-docs/PySide.QtGui/QLineEdit.html#PySide.QtGui.QLineEdit
SearchUI class

class additionalUI.search.SearchUI(session, trackManager, window, request=None)
Bases: PySide.QtGui.QDialog

__init__(session, trackManager, window, request=None)
Call the search engine and call the appropriate function to display them.

search()
Search the annotations for the request in the input field and show the results.

seeGene(gene)
Move the browser’s view to the region of the gene.

logRequest(request)
Log the request and update the completer of the input field with it.

2.2.6 trackHeight module

This module defines the interface to change the height of the tracks

Classes:

trackHeightUI class

class additionalUI.trackHeight.trackHeightUI(trackManager, window, track=None)
Bases: PySide.QtGui.QDialog

User interface to edit the height of all the tracks at once.

__init__(trackManager, window, track=None)
Initialize the dialog and its content and show it.

resizeTracks()
Resize the tracks with the height entered in the self.trackHeight field

2.2.7 trackSettings module

This module manages the modifications of the settings for the datasets.

Classes:

ColorPushButton class

class additionalUI.trackSettings.ColorPushButton(parent, initialColor)
Bases: PySide.QtGui.QPushButton

Button to choose a color.

colorChanged = <PySide.QtCore.Signal object at 0x05B39F40>
Signal emitted when the selected color has changed.

__init__(parent, initialColor)
Initialize the button with the given initialColor.

9http://srinikom.github.com/pyside-docs/PySide/QtGui/QDialog.html#PySide.QtGui.QDialog
10http://srinikom.github.com/pyside-docs/PySide/QtGui/QDialog.html#PySide.QtGui.QDialog
openColorPicker()  
Open the dialog to change the color.

updateColor()  
Change the appearance of the button according to currently selected color.

**DatasetListModel class**

class additionalUI.trackSettings.DatasetListModel (datasets, parent=None)

Bases: PySide.QtCore.QAbstractListModel

This class provides a model for a simple list of datasets.

__init__(datasets, parent=None)  
Initialize the model.

rowCount (parent)  
Return the number of elements in the list.

datasetRole = PySide.QtCore.Qt.ItemDataRole.UserRole

data (index, role)  
Return the data of type role for the element at given index.

**TrackSettings class**

class additionalUI.trackSettings.TrackSettings (tracks, window)

Bases: PySide.QtGui.QDialog

Dialog to set the settings of a track.

__init__(tracks, window)

validateSettings()  
Save and quit.

**2.3 freeze module**

Module to freeze the program into an executable.

Should be used as:

```
python3 freeze.py build -b <build directory>
```

To release the program, the following operations should be done before freezing. Additional freezing on other systems doesn’t need these operations to be done again.

- Build resources:
  
  ```
  pyside-rcc resources.qrc -py3 -o qrc_resources.py
  ```

- Release translation files:

  ```
  lrelease lang/project.pro
  ```

- Build documentation:

  ```
  http://srinikom.github.com/pyside-docs/PySide/QtCore/QAbstractListModel.html#PySide.QtCore.QAbstractListModel
  ```

  ```
  http://srinikom.github.com/pyside-docs/PySide/QtGui/QDialog.html#PySide.QtGui.QDialog
  ```
2.4 functions module

This module defines some useful functions for the whole program.

**functions.sortedNicely** *(l, key=None)*

Sort the given iterable in the way that humans expect.

```python
>>> sortedNicely(['Bar11.1', 'foo102', 'bar7', 'bar9', 'bar11.2', 'foo37'])
['bar7', 'bar9', 'Bar11.1', 'bar11.2', 'foo37', 'foo102']
```

**functions.formatPosition** *(chromosome=None, start=None, end=None)*

Return a formatted string with the given position arguments. Positions are assumed to be 0-based and they are displayed as 1-based coordinates.

```python
>>> formatPosition('chr1', 526348, 526527)
'chr1:526,349-526,528'
>>> formatPosition('chr1', 526348)
'chr1:526,349'
>>> formatPosition(start=526348, end=526527)
'526,349-526,528'
>>> formatPosition('chr1')
'chr1'
>>> formatPosition(start=526348)
'526,349'
```

**functions.importClass** *(fullClassName)*

Return the class which fully qualified class name is fullClassName.

```python
>>> importClass('collections.OrderedDict')
<class 'collections.OrderedDict'>
```

**functions.hcl2rgb** *(h, c, l, hexa=True)*

Convert a color from the color space HCL (Hue, Chroma, Lightness, derived from CIE 1976 (L*, a*, b*) color space) to RGB. Return the tuple (R, G, B) corresponding to the given values of HCL if hexa is False or the hexadecimal representation if hexa is True.

```python
>>> hcl2rgb(60, 0.7, 1, False)
(198, 168, 96)
>>> hcl2rgb(60, 0.7, 1)
'#C6A860'
```

*Note:* The code is adapted from chroma.js\(^{14}\). An explanation is given on vis4.net\(^{15}\).

*Note:* You can use the function testColors() (page 34) to visualize the result. The HCL color picker\(^{16}\) may also help to find correct values for HCL.

**functions.colorPreview** *(colors, width=160, height=200)*

This function return an image representing the colors given in the given list of colors.

---

\(^{14}\)https://github.com/gka/chroma.js

\(^{15}\)http://vis4.net/blog/posts/avoid-equidistant-hsv-colors/

\(^{16}\)http://tristen.ca/hcl-picker
functions.testColors(colors)
This function is used to visualize colors. The argument colors should be a list of hexadecimal representations of colors.

```python
>>> colors = [hcl2rgb(i, 2, 1) for i in range(0, 360, 20)]
>>> testColors(colors)
```

```python
>>> colors1 = [hcl2rgb(0, 2, i / 10) for i in range(9, -1, -1)]
>>> colors2 = [hcl2rgb(90, 2, i / 10) for i in range(0, 10)]
>>> testColors(colors1 + colors2)
```

```python
>>> colors = [hcl2rgb(240, 1+1.7*i/50, 1.5-1.2*i/50) for i in range(0,50)]
>>> testColors(colors)
```

functions.callSamtools(args, outputFile=None)
Call the program Samtools with the arguments given as a list in args. If outputFile is a valid resource, the output will be written in it and the method return True in case of success and False otherwise. If outputFile is None, the output is directly returned.

functions.callTabix(args, outputFile=None)
Call the program Tabix with the arguments given as a list in args and return the output.

functions.tr(textId, noTranslate=False)
Return the translate version of textId in the default context. If noTranslate is True, textId is returned. This is useful when a string should be detected added to the .ts files but not actually translated.

functions.readGtfAttributes(rawAttribute)
Parse the attribute field of the line of a GTF file. Return a dictionary with the attribute names as keys and the attributes values as values.

functions.chromName(chrom)
Return the normalized chromosome name.

Classes:

### 2.4.1 Pool class

class functions.Pool
Bases: builtins.object

Class providing to a unique pool to launch multi-processed tasks.

```python
static instance()
   Return the unique pool.

__dict__ = dict_proxy({'__module__': 'functions', '__instance': []}, 'instance': <staticmethod object at 0x03F06E90>, '__weakref__' of 'Pool' objects>, '__doc__': '
 Class providing to a unique pool to launch multi-processed tasks.
 '})

__weakref__
list of weak references to the object (if defined)

static apply(func, args)
Run func with arguments args on the pool. This method is similar to multiprocessing.Pool.apply, nut it can handle exceptions.
2.4.2 RandomColor class

class functions.RandomColor  
Bases: builtins.object

Class to generate random colors.

colorPalette = ['#68F5A7', '#F991F8', '#FB902B', '#5DCEFB', '#D5EB0C', '#B89484', '#91A74A', '#FB6F96', '#42EF51', '#FFD6F9', ... '#15ED81', '#FAD333', '#BBA200', '#99F9D2', '#F0F84D', '#E0878D', '#8DA94D', '#26E769', '#7CAAA4', '#FDBB97', '#FCD095']

List of random colors smartly chosen and smartly sorted. Colors were generated by http://tools.medialab.sciences-po.fr/iwanthue/ with 0<hue<360, 0<chroma<1.8, 0.9<lightness<1.5, 300 colors from ‘force vector’.

currentIndex = [0]

Random initialization.

static generate()  
Return a random color within the same palette as the previous one, but not too close from the previously generated colors.


__weakref__  
list of weak references to the object (if defined)

2.5 generateDoc module

This module automatically generates restructured text files to build the documentation with Sphinx and autodoc.

generateDoc.docPath = ‘D:\RNAseqViewer\src/../doc/src’  
Path of the directory to build the documentation in.

generateDoc.generatePackage (packageName, parents='')  
Generate the restructured text document for the documentation of the package packageName. If packageName, generate the documentation of the package docPath (page 35). The argument parents is the relative path from the program root to the package, with / as a directory separator.

generateDoc.generateModule (moduleName, parents)  
Generate the restructured text document for the documentation of the module moduleName. The argument parents is the relative path from the program root to the module, with / as a directory separator.

generateDoc.generateClass (className, parents)  
Generate the restructured text document for the documentation of the class className. The argument parents is the relative path from the program root to the module of the class, with / as a directory separator.

2.6 lang package

Content:

2.6.1 generateTs module

Script to generate the translation files (.ts)
2.7 mainUI package

Content:

2.7.1 dataTree module

This module defines the elements which constitute the list of data which have been loaded in the program.

Classes:

**DataTreeDelegate class**

class mainUI.dataTree.DataTreeDelegate

   Bases: PySide.QtGui.QStyledItemDelegate

   This delegate can handle interaction with the user to let him parameterize the way the datasets are displayed.

   viewChangeRequested = <PySide.QtCore.Signal object at 0x05BF5AC0>

   Emitted when the user wants to change the view type of a dataset. The dataset and the name of the view type are transmitted.

   viewChangeAllRequested = <PySide.QtCore.Signal object at 0x05BF5AE0>

   Emitted when the user wants to change the view type of all the datasets of a category. The name of the category and the name of the view type are transmitted.

   hideRequested = <PySide.QtCore.Signal object at 0x05BF5B00>

   Emitted when the user wants to hide the track of a dataset. The dataset is transmitted.

   hideAllRequested = <PySide.QtCore.Signal object at 0x05BF5B20>

   Emitted when the user wants to hide all track of a category. The name of the category is transmitted.

   editorEvent (event, model, option, index)

   Handle interactions with the user. Emit the signals corresponding to user clicks.

**DataTreeDock class**

class mainUI.dataTree.DataTreeDock (tree, menu, window)

   Bases: PySide.QtGui.QDockWidget

   Dock to display the track list.

   __init__ (tree, menu, window)

   Initialize the dock and defines its layout.

**DataTreeMenu class**

class mainUI.dataTree.DataTreeMenu (dataTree, window)

   Bases: PySide.QtGui.QToolBar

   __init__ (dataTree, window)

   Initialize and populate the menu.

---

17http://srinikom.github.com/pyside-docs/PySide/QtGui/QStyledItemDelegate.html#PySide.QtGui.QStyledItemDelegate

18http://srinikom.github.com/pyside-docs/PySide/QtGui/QDockWidget.html#PySide.QtGui.QDockWidget

19http://srinikom.github.com/pyside-docs/PySide/QtGui/QToolBar.html#PySide.QtGui.QToolBar
newAction(text, icon, slot, disable=False)
Create an action and add it to the toolbar. Return the created action.

addStretch()
Add an auto-expanding space in the toolbar

setCurrentTrack()
Save the current dataset and update the controls of the menu. This function should be called each time the selection has changed. Return a boolean indicating if a track is currently selected.

deleteTrack()
This method is called when a user clicks on the ‘delete’ button.

duplicateTrack()
This method is called when a user clicks on the ‘delete’ button.

DataTreeModel Class

class mainUI.dataTree.DataTreeModel(session, trackManager)
Bases: PySide.QtCore.QAbstractItemModel

Model used by DataTreeView. Dataset types are the top-level nodes, datasets are the leaves.

categoryContentChanged = <PySide.QtCore.Signal object at 0x05BF5A60>
Emitted when the content of a row has been modified.

nodeTypeRole = PySide.QtCore.Qt.ItemDataRole.UserRole
trackRole = 33
categoryRole = 34
viewTypeRole = 35
activatedRole = 36

__init__(session, trackManager)

rowCount (parent)
Return the number of children of the given parent.

columnCount (parent=None)
Return the number of columns for the tree. The parameter parent is not (and should not be) taken into account.

data (index, role)
Return the data with the role role at position index.

index (row, column, parent)
Return the index of the row-th child of parent

parent (index)
Return the index of the parent of the element with given index.

nodeType (index)
Return the type of the node referred by the given index. The possible results are ‘root’, ‘category’, ‘track’ and ‘other’.

categoryFromIndex (index)
Return the category name which is referred by the given index.

---

20http://srinikom.github.com/pyside-docs/PySide/QtCore/QAbstractItemModel.html#PySide.QtCore.QAbstractItemModel

2.7. mainUI package
indexOfCategory (category, column=0)
Return the index of the given category.

trackFromIndex (index)
Return the dataset referred by the given index.

rowOfCategory (category)
Return the row of a category.

rowOfTrack (track)
Return the row of a dataset in its category.

addLine (track)
Emit the appropriate signals to update the view after the dataset has been added. Also watch the dataset to keep informed of any change.

emitDataChanged (track)
Emit the signal dataChanged after the given dataset has changed.

DataTreeView class

class mainUI.dataTree.DataTreeView (session, trackManager, window)
Bases: PySide.QtGui.QTreeView

Subclass of QTreeView to display the dataset list as a tree.

treeRefreshed = <PySide.QtCore.Signal object at 0x05BF5680>
Emitted when the tree is refreshed.

__init__ (session, trackManager, window)
Initialize the tree view

adjustColumnsWidth ()
Adjust the width of the columns, so that there is no scrollbar (unless the tree widget is very too small) and all the buttons are visible.

resizeEvent (event)
Call adjustColumnsWidth () when the list is resized.

refresh (category)
Refresh the part of the tree situated under the given category, which can be a true category name or ‘root’. If it is true category, its content is updated and the category is expanded. If it is the root, the list of categories is updated and, if there are new categories, they are expanded.

This method does not update the layout, which is done instead in DataTreeModel.addLine () by emitting the signal layoutChanged.

resetAll ()
Refresh the whole tree and its selected index.

saveAsExpanded (index)
Log the category referred by index as expanded.

saveAsCollapsed (index)
Log the category referred by index as collapsed.

21http://srinikom.github.com/pyside-docs/PySide/QtGui/QTreeView.html#PySide.QtGui.QTreeView
2.7.2 infoDock module

This module defines a dock to display data from track elements tool tip.

Classes:

InfoDock class

class mainUI.infoDock.InfoDock(window)
    Bases: PySide.QtGui.QDockWidget

    Dock to display data from track elements tool tip.

    __init__(window)
        Initialize the dock and defines its layout.

    showInfo(text)
        Populate the dock with the given text.

2.7.3 legendDock module

This module defines a dock to display the legend of the tracks.

Classes:

LegendDock class

class mainUI.legendDock.LegendDock(window)
    Bases: PySide.QtGui.QDockWidget

    Dock to display the legend of the tracks.

    __init__(window)
        Initialize the dock and defines its layout.

    setTrack(track=None)
        Redraw the legend for the given track. If track is None, no legend is shown.

    redraw()
        Redraw the legend for the current track.

    hideLegend()
        Hide the legend.

2.7.4 mainWindow module

This module defines the class MainWindow (page 40).

Classes:

http://srinikom.github.com/pyside-docs/PySide/QtGui/QDockWidget.html#PySide.QtGui.QDockWidget

http://srinikom.github.com/pyside-docs/PySide/QtGui/QDockWidget.html#PySide.QtGui.QDockWidget
**MainWindow class**

```python
class mainUI.mainWindow.MainWindow (parent=None)
Bases: PySide.QtGui.QMainWindow

Class for the main window of the application. @todo: updateVisibleArea when window’s size change or splitters move

__init__ (parent=None)
Initialize the main window.

createAction (text, slot=None, shortcut=None, ico=None, tip=None)
Customized function to create an action

addActions (target, actions)
Customized function to add actions and separators to a target menu The actions are given as a list, with None meaning a separator.

openSession ()
Let the user select the session file and restore the data from the session which has been previously saved

saveSession ()
Save the current session.

newSession ()
Create a new session. Return False if the user refused to close the current session (new session aborted) or True if it was successful

closeSession ()
Close the current session.

saveWindowState ()
Save the state and the geometry of the main window, the tool bars and the docks.

restoreWindowSate ()
Restore the state and the geometry of the main window, the tool bars and the docks.

addData (dataType)
Let the user select a data file of the given type of data to add to the current session.

updateChromList ()
Update the chromosome list in the navigation bar

closeEvent (event)
Clean and save the session before quitting.

okToQuit ()
Ask the user to confirm when he/she quits the program.

launchSearch ()
Launch the search UI.

launchAbout ()
Open the ‘About’ dialog.

launchHelp ()
Open the ‘Help’ dialog.

launchSettings ()
Open the dialog to change general settings.
```

24http://srinikom.github.com/pyside-docs/PySide.QtGui/QMainWindow.html#PySide.QtGui.QMainWindow
launchTrackHeight()
Open the dialog to change general settings.

### 2.7.5 toolBar module

This module defines the tool bars of the application. All the tool bars are subclasses of the meta-class `ToolBar` (page 41).

Classes:

**ToolBar class**

```python
class mainUI.toolBar.ToolBar(trackManager, session, parent)

Bases: PySide.QtGui.QToolBar
```

Tool bar to see and set the location to display.

**moveRequested** = `<PySide.QtCore.Signal object at 0x063B62E0>`

Emitted when the user wants to move forward. The size of the move is given in term of a fraction of a screen width. A positive move indicates a move forward; a negative move indicates a move backward.

**zoomRequested** = `<PySide.QtCore.Signal object at 0x063B62C0>`

Emitted when the user wants to zoom. The ratio of the zoom is given.

```python
__init__(trackManager, session, parent)
```

Initialize and populate the tool bar.

```python
addStretch()
```

Add a spacer widget to the tool bar.

```python
setHideIntronButtonState()
```

Enable or disable the button to hide introns, according to the presence of annotations.

```python
setChromList(newChromList)
```

Set the list of chromosome in the combobox.

```python
setChromosome(chromosome, noError=False)
```

Select the given chromosome in the chromosome list. Raise a `KeyError` if the chromosome is not found in the chromosome list, unless if `noError` is `True`.

```python
updatePosition()
```

Update the location text field and the chromosome combo box with the view’s coordinates.

```python
validateAreaLabel()
```

Check if the input of the area label is correct and change the view accordingly if it is.

```python
showError(text)
```

Display a tool tip in red next to the location input field to show the error message

```python
setPositionFormEnabled(enable=True)
```

Enable (or disable if `enable` is `False`) the controls in the location bar

```python
validateSearch()
```

Launch the search UI with current search input.

```python
updateButtonsAvailability()
```

Enable or disable the view control buttons according to what is possible.

---


26[http://docs.python.org/3.2/library/exceptions.html#KeyError]
2.7.6 trackSplitter module

This module defines the class `TrackSplitter` (page 42) for the layout of the tracks and the class `SplitterSpacer` (page 42), which is used by `TrackSplitter` (page 42).

Classes:

**SplitterSpacer class**

class `mainUI.trackSplitter.SplitterSpacer (splitter)`  
Bases: `PySide.QtGui.QLabel`  
A `SplitterSpacer` is a spacer for splitters (like `QSpacerItem` for layouts)

__init__ (splitter)

minimumSizeHint ()

dragEnterEvent (event)  
Specify if a widget can be dragged here.

dropEvent (event)  
Move the track when the user drops it.

**TrackSplitter class**

class `mainUI.trackSplitter.TrackSplitter (parent=None)`  
Bases: `PySide.QtGui.QSplitter`  
This method defines a vertical splitter to display tracks. It handles carefully the height of the tracks. Use `addWidget()` to insert a track’s view in last position.

As the splitter has a spacer after the last track’s view, be careful when using `insertWidget()`.

__init__ (parent=None)

insertWidget (index, widget)  
Add a widget at the given index. This method takes care of the relative height of widget and their hint height.

addWidget (widget)  
Add a widget in the last position before the spacer. This method takes care of the relative height of widget and their hint height.

2.8 qrc_resources module

`qrc_resources.qInitResources ()`

`qrc_resources.qCleanupResources ()`

---

27 http://srinikom.github.com/pyside-docs/PySide/QtGui/QLabel.html#PySide.QtGui.QLabel

28 http://srinikom.github.com/pyside-docs/PySide/QtGui/QSpacerItem.html#PySide.QtGui.QSpacerItem

29 http://srinikom.github.com/pyside-docs/PySide/QtGui/QSplitter.html#PySide.QtGui.QSplitter
2.9 session package

This package contains the model of the datasets. The class `Session` (page 53) manages a group of datasets which are displayed together. Each kind of dataset is defined in its own module by its own class, which inherits the abstract class `Dataset` (page 46).

Content:

2.9.1 annotation module

This module provides a handler for RefSeq files. A RefSeq file is managed by an instance of the class `Annotation` (page 43), which links to several instances of `AnnotationLine` (page 43).

Classes:

Annotation class

class `session.annotation.Annotation` (filename)

Bases: `session.dataset.Dataset` (page 46)

Structure to store genome annotations. This is an abstract class.

__init__ (filename)

Initialize the structure.

getLines (chromosome, start=None, end=None)

Return the list of `AnnotationLines` corresponding to the given chromosome. Must be implemented in subclasses.

search (request)

Return the list of annotations which match the given request. Must be implemented in subclasses.

AnnotationFactory class

class `session.annotation.AnnotationFactory`

Bases: `builtins.object`

Entry class for genome annotation data sets. This class only defines the method __new__ () and it creates an instance of the appropriate class according to the file extension.

static __new__ (filename)

If someone tries to construct a AnnotationFactory object, a GtfAnnotation, RefFlatAnnotation or InvalidAnnotation object is constructed instead, depending on the file's extension.

__dict__ = dict_proxy({'__dict__': <attribute '__dict__' of 'AnnotationFactory' objects>, '__module__': 'session.annotation', ...

__weakref__

list of weak references to the object (if defined)

AnnotationLine class

class `session.annotation.AnnotationLine` (annotationFormat, rawData)

Bases: `session.regions.Region` (page 51)

One line of Annotation
ForwardStrand = ‘+’
ReverseStrand = ‘-’

__init__(annotationFormat, rawData)
Initialize the annotation. If annotationFormat is ‘RefFlat’, rawData is a list of all the cells from one line in a RefSeq file. If annotationFormat is ‘GTF’, rawData is a tuple of a list of exons and a list of CDS regions. The lines of each list are lists of the values from a line of a GTF file with feature equal to exon or CDS.

GtfAnnotation class

class session.annotation.GtfAnnotation (filename)
Bases: session.annotation.Annotation (page 43)
Structure to store genome annotations for datasets in GTF format.

__init__(filename)
Initialize the structure, check file consistency and populate chromosome list.

getLines(chromosome, start=None, end=None)
Return the list of AnnotationLines corresponding to the given chromosome.

indexFile()
Return the path of the search index for the GTF file.

search(request)
Return the list of annotations which match the given request.

GtfGzAnnotation class

class session.annotation.GtfGzAnnotation (filename)
Bases: session.annotation.Annotation (page 43)
Structure to store genome annotations for big datasets in GTF format. The GTF file should be sorted, compressed with bgzip and indexed by tabix.

__init__(filename)
Initialize the structure, check file consistency and populate chromosome list.

getLines(chromosome, start=None, end=None)
Return the list of AnnotationLines corresponding to the given chromosome between left and right

search(request)
Return the list of annotations (class:AnnotationLine) which match the given request.

InvalidAnnotation class

class session.annotation.InvalidAnnotation (filename)
Bases: session.annotation.Annotation (page 43)
An invalid subclass of Annotation.

__init__(filename)
**RefFlatAnnotation class**

```python
class session.annotation.RefFlatAnnotation(filename)
Bases: session.annotation.Annotation (page 43)
```

Structure to store genome annotations for datasets in RefFlat format

```python
__init__(filename)
Initialize the structure, check file consistency and populate chromosome list.
```

```python
getLines(chromosome, start=None, end=None)
Return the list of AnnotationLines corresponding to the given chromosome.
```

```python
search(request)
Return the list of annotations (class:AnnotationLine) which match the given request.
```

### 2.9.2 dataTypes module

This module gives the parameters for the various possible data types.

```python
session.dataTypes.dataTypes = [session.dataTypes.DataType object at 0x05B37810], <session.dataTypes.DataType object at 0x05B37850>, ...
```

List of available data types.

```python
session.dataTypes.getTypeInfo(typeName, role)
Return the data corresponding to the given role for the given data type.
```

```python
session.dataTypes.getTypeByName(name)
Return the element of dataTypes which name is given.
```

```python
session.dataTypes.getTypeByClass(c)
Return the element of dataTypes which model is defines by the class c.
```

```python
session.dataTypes.getTypeByTrack(track)
Return the element of dataTypes which can be displayed by the given track.
```

Classes:

**DataType class**

```python
class session.dataTypes.DataType(name, title, modelClass, graphClass, extensions, subclassNames={})
Bases: builtins.object
```

Define a type of dataset.

```python
__init__(name, title, modelClass, graphClass, extensions, subclassNames={})
Set the instance variables
```

```
__dict__= dict_proxy({'__module__': 'session.dataTypes', 'viewTypes': <function viewTypes at 0x05AC6B70>, '__dict__': <attribute ... of 'DataType' objects>, '__doc__': '
 Define a type of dataset.
 ', '__init__': <function __init__ at 0x05AC6B28>})
```

```python
__weakref__
list of weak references to the object (if defined)
```
2.9.3 dataset module

This module defines the abstract class `Dataset` (page 46), which is inherited by all the other classes defining datasets. During a dataset initialization, if it turns out that the dataset is not usable and should not be used in the program, it should raise the exception `DatasetNotValid` (page 47), which is also defined in this module.

Classes:

**Dataset class**

class `Dataset` (filename)

Bases: `PySide.QtCore.QObject`

Structure to store any data to be displayed by the software, including reference genome, annotations, read alignment, etc.

`Dataset()` is an abstract class, and some of its methods should be defined in subclasses.

```
invalid = <PySide.QtCore.Signal object at 0x065A9C40>
```

Emitted if the dataset is found to be invalid or should be deleted.

```
__init__(filename)
```

Initialize a dataset.

If this method is re-implemented in subclasses, it should raise the exception `DatasetNotValid` (page 47) if the initialization fail and the dataset can’t be used.

```
getFullPath()
```

Return the full path of the file which contains the data.

```
getFileName()
```

Return the filename of the file which contains the data.

```
getFormat()
```

Return the type of data.

```
getChromosomes()
```

Return a list of name of the chromosomes considered in the dataset.

---

**Note:** Attribute `self.chromosomes` must be populated in subclasses.

```
addChromosome(chrom)
```

Add the given chromosome to the chromosome list.

```
chromID(chromName)
```

Return the sequence ID like it is given in the dataset. This method is the inverse of the function `functions.chromName()` (page 34).

```
readCSV()
```

Open the file to be read as a CSV file.

```
addTrack(track)
```

Inform the dataset that a new track has been added.

```
removeTrack(track)
```

Inform the dataset that a track has been deleted.

```
bTracks()
```

Return the number of tracks which use the dataset.

---

30http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
DatasetNotValid class

class session.dataset.DatasetNotValid
    Bases: builtins.Exception

    This exception is raised when a dataset appears to be invalid.

    __weakref__
    list of weak references to the object (if defined)

2.9.4 junctions module

This module provide a handler for splicing junctions.

Classes:

Junctions class

class session.junctions.Junctions(filename)
    Bases: session.regions.Regions (page 52)

    Structure to store junctions

    __init__(filename)
    Read the file and initialize the model

2.9.5 numeric module

This module provide a handler for numeric data sets.

Classes:

FixedStepNumeric class

class session.numeric.FixedStepNumeric(filename)
    Bases: session.numeric.Numeric (page 48)

    __init__(filename)
    Initialize a numeric dataset stored in a fixed step Wiggle file. Get the start and step headers.

    getValues(chrom, start, end)
    Return at least the values defined on the region from start to end on the given chromosome. Return a range of the positions and a list of the corresponding values.

InvalidNumeric class

class session.numeric.InvalidNumeric(filename)
    Bases: session.numeric.Numeric (page 48)

    An invalid subclass of Numeric.

    __init__(filename)
**Numeric class**

```python
class session.numeric.Numeric(filename)
```

Bases: `session.dataset.Dataset` (page 46)

Structure to store numeric data sets. This is an abstract class.

```python
__init__(filename)
```

Initialize a numeric data set. Read and save the `chrom` and `span` headers of the WIG file.

```python
getValues(chromosome, start, end)
```

Return the values defined on the region from `start` to `end` on the given `chromosome`. This method should be implemented in subclasses.

**NumericFactory class**

```python
class session.numeric.NumericFactory
```

Bases: `builtins.object`

Entry class for numeric data sets. This class only defines the method `__new__()` and it creates an instance of the appropriate class according to the file header.

```python
static __new__(filename)
```

If someone tries to construct a `NumericFactory` object, a `FixedStepNumeric`, `VariableStepNumeric` or `InvalidNumeric` object is constructed instead, depending on the file’s header.

```python
__dict__ = dict_proxy({'__dict__': <attribute '__dict__' of 'NumericFactory' objects>, '__module__': 'session.numeric', ... instance of the appropriate class according to the file header.
 ', '__new__': <staticmethod object at 0x0682DB70>})
```

```python
__weakref__
```

list of weak references to the object (if defined)

**VariableStepNumeric class**

```python
class session.numeric.VariableStepNumeric(filename)
```

Bases: `session.numeric.Numeric` (page 48)

```python
__init__(filename)
```

Initialize a numeric dataset stored in a fixed step Wiggle file. Store values in memory.

```python
getValues(chromosome, start, end)
```

Return the values defined on the region from `start` to `end` on the given `chromosome`. Return a list of the positions and a list of the corresponding values.

### 2.9.6 reads module

This module provide a handler for Reads Alignment files. The class `Reads` (page 50) can read SAM/BAM files. An instance of `Read` (page 50) defines a unique read.

The class `NucleotideCache` (page 49) and `NucleotideCacheBlock` (page 49) are used to manage the cache of the read coverage at each nucleotide-level position. The aim is to reduce the number of calls to Samtools and to make most of the calls asynchronous. The function `getReadCoverage()` (page 48) is the function which is called asynchronously to retrieve the read coverage from Samtools.

```python
session.reads.getReadCoverage(bamFile, chrom, start, end)
```

Ask Samtools for the coverage of the given `bamFile` in the region from `start` to `end` of chromosome `chrom`. 

---

**Chapter 2. Source Code Documentation**
Return a list of lists (converted to a string by pickling). The first element of each list is a nucleotide index (start is 0, start + 1 is 1, etc.). The second to fifth are the number of covering A, C, G and T (in this order).

Classes:

**NucleotideCache class**

```python
class session.reads.NucleotideCache(bamFile, chromosome)
```

Cache for per-nucleotide read count in a BAM file.

The cache has several blocks (session.reads.NucleotideCacheBlock (page 49)), with the oldest block being reused when new data should be stored in cache.

```python
__init__(bamFile, chromosome)
```

Initialize the cache.

```python
getValues(start, end, total=True)
```

Return the values from start to end. If they are not in cache, compute them and store them in cache. If total is True, the total coverage fore all A,C,G,T is returned. Else a list of tuples with coverage for each A,C,G,T is returned.

```python
getCacheBlock(blockIndex)
```

Return the block with index blockIndex. Create it if not found.

```python
clearOldCaches()
```

Keep only the nbBlocksMax first blocks and delete the others.

```python
__weakref__
```

list of weak references to the object (if defined)

**NucleotideCacheBlock class**

```python
class session.reads.NucleotideCacheBlock(nucleotideCache, start, end)
```

One block of the nucleotide cache (session.reads.NucleotideCache (page 49)).

Samtools is launched asynchronously when the instance is initialized, but the processing of the result (i.e. converting to an array index => count) is done by saveSamtoolsResult() the first time a value of the block is requested.

```python
__init__(nucleotideCache, start, end)
```

Initialize the part of the cache and start Samtools asynchronously.

```python
getValues(start, end, total=True)
```

Return the values from start to end. If total is True, the total coverage for all A, C, G, T is returned. Else a tuple of four lists is returned with the coverage for each A, C, G and T.

```python
saveSamtoolsResult()
```

Populate the attribute cachedContent after Samtools has finished.

```python
__weakref__
```

list of weak references to the object (if defined)
Read class

class session.reads.Read(qname, flag, rname, pos, mapq, cigar, rnext, pnext, tlen, seq, qual, opt)

This class defines a read.

__init__(qname, flag, rname, pos, mapq, cigar, rnext, pnext, tlen, seq, qual, opt)
    Initialize a read.

    qname, rname, cigar, rnext, seq and qual are strings. flag, pos, mapq, pnext and tlen are integers. opt is a dictionary.

    Positions (pos and pnext) are 0-based.

__dict__ = dict_proxy(_module_={'session.reads', _weakref_: <attribute '__weakref__' of 'Read' objects>, __doc__:
    This class defines a read.
    __init__': <function __init__ at 0x05A41E40>})

__weakref__
    list of weak references to the object (if defined)

limits()
    Return the position of the first and the last nucleotide.

exonLimits()
    Return a list of exon start positions and a list of exon end positions

activatedFlags()
    Return a list of activated flags.

quality(position)
    Return the quality of the base at the given position. If the quality is not known, return None.

Note: A base quality is the phred-scaled base error probability which equals
-10 \log_{10} \Pr\{\text{base is wrong}\}.

Reads class

class session.reads.Reads(filename)

Bases: session.dataset.Dataset (page 46)

Class to handle reads alignment file, in SAM/BAM format.

Note: There are two main caches. The first one is regionCache, which stores the final result of read count requests. It consists in one dictionary for each mode of the method readsCount(). The second cache is nucleotideCache, an instance of NucleotideCache. It stores the number of reads mapped to each nucleotide.

__init__(filename)
    Initialize the model to access the BAM file located at filename. If the file is in SAM format, the user is prompted to convert it in BAM format.

resetCache(chromosome=None)
    Reset all the variables used for the cache and the cache management.

maxReadCount(chrom, start, end)
    Return the maximum value of the read count attained by a position on chrom, in a region including the region between start and end.
Note: To be able to use the region cache smartly, the considered region is larger than the region between start and end, and its limits are round numbers. For instance, if start = 1200 and end = 1800, then the method will consider the region [1000-2000]. It will be also the case for start = 1318 and end = 1637 for example.

`readsCount(chrom, start=None, end=None, mode='sum')`
Count the number of reads that intersect the region from start to end on chromosome chrom. If start is None, the whole chromosome is considered. If end is None, only the nucleotide at position start is considered.

If the mode is 'sum', then return the sum of total read coverage for all positions and all A,C,G,T. If the mode is 'max', return the highest coverage value among the considered positions (read count value for the position with maximum A+C+G+T). If mode is 'detail', return a tuple with the number of A, C, G and T mapped to start position. If mode is 'region', return the number of unique reads mapped between start and end.

Note: The code assume that the executable samtools is available in the same directory and is runnable on the user platform. If it is not the case, an error message appear to the user.

Hence, when using the program on a new platform, users should first download samtools for their platform and make it available from the directory of the program.

`updateTotalReadCount()`
Update totalReadCount with the total number of reads for the chromosome chromosome. Return the read count.

`getChromosomes(lengths=False)`
Read the header of the BAM file to get the list of chromosomes and their lengths.

If lengths is False, return a list of the chromosome names. If it is True, return a dictionary (chromosome name => chromosome length).

`isSorted()`
Check if the read alignment file is sorted.

`getReads(chromosome, start, end)`
Return a list of Read, which contains all the reads mapped to the chromosome chromosome between start and end.

2.9.7 regions module

This module provides a handler for BED files. A BED file is managed by an instance of the class Regions (page 52), which links to several instances of Region (page 51).

Classes:

**Region class**

```python
class session.regions.Region(rawData)
Bases: PySide.QtCore.QObject
```

Define a region (one line of a BED file). A region may be made of several discontinuous blocks.

```python
ForwardStrand = '+'
```
ReverseStrand =  '-'

__init__(rawData)

rawData is a list of all the cells from one line in a BED file.

getBlockLimits(fromZero=False)

Return the list of all block limits. Each list element corresponds to a block, it is a tuple which first element
is the start index and the second element is the end index. If fromZero is True, the coordinates begin from
the beginning of the chromosome instead of the beginning of the annotation.

limits()

Return the coordinate of the first and the last nucleotide of the region.

Regions class

class session.regions.Regions(filename)

Bases: session.dataset.Dataset (page 46)

Structure to store BED tracks.

__init__(filename)

Initialize the structure, check file consistency and populate chromosome list.

getLines(chromosome=None, start=None, end=None)

Return the list of Region corresponding to the given chromosome. If chromosome is None, return all
the regions. If chromosome, start and end are all provided, only the regions between start and end are
guaranteed to be returned. The default implementation returns all the regions of the chromosome.

getchromosomes()

Return the list of chromosome names in the BED file

2.9.8 save module

This module contains functions to save and restore sessions

Classes:

SessionSaving class

class session.save.SessionSaving

Bases: PySide.QtCore.QObject

This class contains functions to save and restore sessions.

openSession(window)

Let the user select the session file and restore the data from the session which has been previously saved

saveSession(window)

Save the current session.

static reportProgress(blocknum, blocksize, totalsize)

Refresh the progress bar with current download state

download(url)

Download the file at the given URL. If the file is a .gz file, it will be uncompress. Return the path to the
downloaded file. If the file has already been downloaded in the past, it will not be downloaded again.

http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
2.9.9 sequence module

This module provides a handler for DNA sequences files.

Classes:

**Sequence class**

```python
class session.sequence.Sequence(filename)
Bases: session.dataset.Dataset (page 46)
```

Structure to store DNA sequences

```python
__init__(filename)
Initialize the instance with a fasta file. An index file is created if necessary, thus allowing a random access
time to the sequence in O(1).
```

```python
getSeq(chromosome, start=None, end=None)
Return the sequence of the given chromosome as a string. If both start and end are specified, only the
part of the sequence form start to end is returned. If end is None, only the nucleotide at position start is
returned.
```

```python
length(chromosome)
Return the length of the sequence. Return 0 if the given chromosome does not exist.
```

```python
computeFileInfo()
Some code is from Biopython's SeqIO.FastaIO.FastaIterator()
(http://biopython.org/wiki/Biopython)
```

2.9.10 session module

This module defines a session.

Classes:

**Session class**

```python
class session.session.Session
Bases: PySide.QtCore.QObject
```

Manage a session and its associated data. @todo: Starting assistant

```python
datasetAdded = <PySide.QtCore.Signal object at 0x065B7900>
datasetRemoved = <PySide.QtCore.Signal object at 0x065BB520>
annotationsAdded = <PySide.QtCore.Signal object at 0x065BB560>
sequencesChanged = <PySide.QtCore.Signal object at 0x065BB5E0>
dataChanged = <PySide.QtCore.Signal object at 0x065BB600>
```

```python
__init__()
Initialize a session
```

```python
addDataset(filename, fileFormat)
Add a data file to the session data list, and returns the data model
```

33http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
removeDataset (dataset)
    Remove the given dataset from the session.

addAnnotations (annotations)
    Add an annotation file to the session

addSequence (sequence)
    Add an sequence to the session

nbDatasets (dataType=None)
    Return the number of datasets added to the session

gedata (dataType=None)
    Return the list of all Data added to the session. If dataType is not None, only data of the type dataType are returned.

getCategories ()
    Return the list of all the types of datasets already loaded.

getAnnotationDataset ()
    Return the list of all the datasets which define some annotations. They are not necessarily of type session.annotation.Annotation (page 43).

getSequence (chromosome)
    Return the sequence corresponding to the given chromosome

chromosomeList ()
    Return a sorted list, which contains the names of all chromosomes in the annotation dataset

search (request)
    Search for the given request among the annotations in the session. Return a list of corresponding genes (AnnotationLine) or, if the session contains no annotation at all, return False.

intronList (chromosome)
    Return an ordered list of tuples representing the start and end coordinates of zones covered only by introns.

2.9.11 transcripts module

This module provides a handler for transcript files. A RefSeq file is managed by an instance of the class Transcripts (page 55), which links to several instances of Transcript (page 54).

The current implementation focuses on the transcript files produced by Cufflinks in GTF format (http://cufflinks.cbcb.umd.edu/manual.html#cufflinks_output)

Classes:

Transcript class

class session.transcripts.Transcript (exons)
    Bases: session.regions.Region (page 51)

    Define a transcript. A transcript may be made of several exons.

    __init__ (exons)
    Initialize a transcript with the given exons. The parameters exons is a list of exons. Each exon is described as a list of the values from a line of a GTF file with feature equal to exon.
**Transcripts class**

```python
class session.transcripts.Transcripts(filename)
Bases: session.dataset.Dataset

Structure to store transcripts.
```

**Note:** Implementing `Transcripts` (page 55) as a subclass of `session.regions.Regions` (page 52) should be considered.

```python
__init__(filename)
Initialize the structure, check file consistency and populate chromosome list.

getLines(chromosome=None, start=None, end=None)
Return the list of Transcript corresponding to the given chromosome. If chromosome is None, return all the transcripts.
```

### 2.10 setup module

This module is used for the installation of the program.

Distribute:

```
python setup.py sdist --dist-dir ../build/
```

Install: Uncompress and enjoy!

### 2.11 tracks package

This package contains the elements to handle and display the tracks. It contains a class `trackManager` (page 68) which manages all the tracks currently displayed. The tracks are instances of one of the subclasses of the virtual class `Track` (page 66). In order to add interactive features to the tracks, they use subclasses of `QGraphicsView`\(^{34}\) and `QGraphicsScene`\(^{35}\), which are defined in the modules `views` (page 72) and `scene` (page 65).

**Content:**

#### 2.11.1 annotationGraph module

This module defines the class `AnnotationGraph` (page 55) to display annotations. It also contains the definitions of the class `AnnotationItem` (page 56), a subclass of `QGraphicsItem` used to draw the annotations.

**Classes:**

```python
class tracks.annotationGraph.AnnotationGraph(regions, trackManager, viewType=None)
Bases: tracks.regionGraph.RegionGraph

Widget to display annotations
```

---

\(^{34}\)http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsView.html#PySide.QtGui.QGraphicsView

\(^{35}\)http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsScene.html#PySide.QtGui.QGraphicsScene
**drawRegion** *(annotation)*  
Draw a given annotation on the scene. Return the instance of *tracks.annotationGraph.AnnotationItem* (page 56).

**AnnotationItem class**

class *tracks.annotationGraph.AnnotationItem* *(region, graph)*  
Bases: *tracks.regionGraph.RegionItem* (page 64), *PySide.QtCore.QObject*  
Graphic element to display an annotation.

toolTip()  
Return the data for the tool tip.

### 2.11.2 graphicsItem module

This module defines the elements needed to draw graphs. It defines a class *GraphicsItem* (page 57) which defines common attributes and several subclasses of it, which defines the graphic items used for the different graphs.

*tracks.graphicsItem.colorPalettes* = *OrderedDict* [('intensity', [PySide.QtGui.QColor.fromRgbaF(0.682353, 0.807843, 1.000000, 1.000000), ... 0.105882, 0.490196, 1.000000), PySide.QtGui.QColor.fromRgbaF(0.556863, 0.003922, 0.321569, 1.000000)])

Palettes of colors.

*tracks.graphicsItem.defaultColor* = *PySide.QtGui.QColor.fromRgbaF(0.000000, 0.533333, 0.000000, 1.000000)*  
Default color for the graphic items.

*tracks.graphicsItem.defaultBrush* = *<PySide.QtGui.QBrush(QColor(ARGB 1, 0, 0.533333, 0) , SolidPattern ) at 0x05B44878>*  
Default brush for the graphic items.

*tracks.graphicsItem.defaultPen* = *<PySide.QtGui.QPen(0,QBrush(QColor(ARGB 1, 0, 0.266667, 0) , SolidPattern ) , SolidLine , 16 , 64 , QVector() , 0 , 2 ) at 0x05B44918>*  
Default pen for the graphic items.

*tracks.graphicsItem.heatColors*()  
Return a set of colors for heatmaps.

Classes:

**BlankReadItem class**

class *tracks.graphicsItem.BlankReadItem* *(starts, ends, readItem, graph)*  
Bases: *tracks.graphicsItem.GraphicsItem* (page 57), *PySide.QtCore.QObject*  
Graphic element to display a read, with no information about its content. Junctions and pair-end reads are taken into account.

*brush* = *<PySide.QtGui.QBrush(QColor(ARGB 1, 0, 0.4, 0.4) , SolidPattern ) at 0x05B44940>*  
*brushHover* = *<PySide.QtGui.QBrush(QColor(ARGB 1, 1, 0, 0) , SolidPattern ) at 0x05B44968>*  
*pen* = *<PySide.QtGui.QPen(2,QBrush(QColor(ARGB 1, 0, 0.4, 0.4) , SolidPattern ) , SolidLine , 16 , 64 , QVector() , 0 , 2 ) at 0x05B44918>*  
*penHover* = *<PySide.QtGui.QPen(2,QBrush(QColor(ARGB 1, 1, 0, 0) , SolidPattern ) , SolidLine , 16 , 64 , QVector() , 0 , 2 ) at 0x05B44918>*

__init__ *(starts, ends, readItem, graph)*  
Initialize the instance and create all the rectangles and lines to be painted. *start* and *ends* are two lists of integers specifying the limits of the exons. *height* is the height in pixels of the read. If *height* is not provided, the height for nucleotides is used. If *readItem* is given, it will be used to look for a potential pair read.

---

36http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
37http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
boundingRect()
    Return the bounding rectangle.

paint(painter, option, widget=None)
    Paint the read.

toolTip()
    Return the read’s tool tip

hoverEnterEvent(event)
    Highlight the read when it is under the mouse

hoverLeaveEvent(event)
    Delete highlight when mouse leave the read

GraphicsItem class

class tracks.graphicsItem.GraphicsItem
    Bases: PySide.QtGui.QGraphicsItem
    Subclass of QGraphicsItem.

    static formatToolTip(dictionary)
        Return the formatted text for a tooltip displaying the data in the dictionary. The dictionary can be ordered
        (OrderedDict).

    setDictToolTip(dictionary)
        Create a tooltip displaying the data in the dictionary. The dictionary can be ordered (OrderedDict).

HeatmapBar class

class tracks.graphicsItem.HeatmapBar(start, end, color, graph)
    Bases: tracks.graphicsItem.GraphicsItem (page 57)

    A blank exon with highlighting capability: the region under the exon is highlighted on all tracks when the exon
    has the cursor over it.

    __init__(start, end, color, graph)

    boundingRect()

    paint(painter, option, widget=None)
        Paint the bar.

    hoverEnterEvent(event)
        Highlight the genomic area of the read.

    hoverLeaveEvent(event)
        Remove the highlight.

HistogramColumn class

class tracks.graphicsItem.HistogramColumn(position, value, graph, rectWidth=1)
    Bases: tracks.graphicsItem.GraphicsItem (page 57)

    Subclass of QGraphicsItem which describes a column of a histogram.
__init__ (position, value, graph, rectWidth=1)
Initialization. The position is the index of a nucleotide along the chromosome. The height of the bar depends on the value.

boundingRect ()
Return the bounding rectangle

paint (painter, option, widget=None)
Paint the histogram bar.

InsertionNucleotide class

class tracks.graphicsItem.InsertionNucleotide (sequence, position, graph, parent=None)
Bases: tracks.graphicsItem.Nucleotide (page 58)
Special type of tracks.graphicsItem.Nucleotide (page 58) for representing insertions. A ‘+’ is displayed in normal state, and the content of the insertion is displayed on hover event.

__init__ (sequence, position, graph, parent=None)

hoverEnterEvent (event)
Draw inserted sequence.

hoverLeaveEvent (event)
Hide inserted sequence.

Nucleotide class

class tracks.graphicsItem.Nucleotide (letter, position, graph, quality=None, parent=None)
Bases: tracks.graphicsItem.GraphicsItem (page 57)
Graphic element to display a nucleotide. It consists in a colored box and, if zoomed enough, the letter of the nucleotide.

p = {}
Parameters of the class, initialized in initializeClass()

static initializeClass (trackManager)
Initialize the class parameters.

__init__ (letter, position, graph, quality=None, parent=None)
Creates a nucleotide of type ‘letter’ at the position ‘position’ in the nucleotide sequence

boundingRect ()
Return the bounding rectangle.

paint (painter, option, widget=None)
Paint the nucleotide.

toolTip ()
Return the tooltip for the nucleotide if there are additional information to display.

ReadItem class

class tracks.graphicsItem.ReadItem (read, graph)
Bases: PySide.QtCore.QObject40

40http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
This class defines a read, made of a `tracks.graphicsItem.BlankReadItem` (page 56), and some `tracks.graphicsItem.Nucleotide` (page 58) can be added.

```python
__init__(read, graph)
```

```python
addNucleotideContent()  
Draw the nucleotides in the read using the CIGAR string
```

```python
removeNucleotideContent()  
Remove the nucleotides from the scene.
```

```python
pos()  
Return the position of the read in the scene.
```

```python
setPos(x, y)  
Set the position of the read in the scene.
```

```python
left()  
Return the abscissa of the leftmost pixel of the read
```

```python
right()  
Return the abscissa of the rightmost pixel of the read
```

```python
removeFromScene()  
Remove the graphics items contained in the instance
```

## 2.11.3 junctionGraph module

This module defines the class `JunctionGraph` (page 59) to display splicing junctions.

Classes:

### JunctionBarItem class

```python
class tracks.junctionGraph.JunctionBarItem(*args, **kwargs)
```

```python
Bases: tracks.regionGraph.RegionItem (page 64), tracks.junctionGraph.JunctionItem (page 60)
```

Graphic element to display a junction as a horizontal bar.

```python
__init__(*args, **kwargs)
```

Initialize the junction.

```python
shape()  
Return the shape on which the tooltip should be activated. This implementation return the bounding rectangle.
```

```python
toolTip()  
The right version to use for the method `toolTip()` is the one defined in `JunctionItem`.
```

```python
color()  
The right version to use for the method `color()` is the one defined in `JunctionItem`.
```

### JunctionGraph class

```python
class tracks.junctionGraph.JunctionGraph(regions, trackManager, viewType=None)
```

```python
Bases: tracks.regionGraph.RegionGraph (page 64)
```

Widget to display splicing junctions.

## 2.11. tracks package
viewTypes = [('lines', 'Broken lines'), ('bar', 'Horizontal bars')]
List of the possible view types. The first one is the default view.

draw (drawingId=None)
Draw the junctions that should be visible in the view.

drawLines (start=0, end=1000)
Draw the junctions which are mapped to the sequence from start to end as lambda-like lines. If the
junctions have already been drawn, do not draw them again.

drawBars (start=0, end=1000)
Draw the junctions which are mapped to the sequence from start to end as horizontal bars. If the junctions
have already been drawn, do not draw them again.

drawRegion (junction)
Draw a given junction on the scene as a bar.

Note: This function is rewritten from tracks.regionGraph.RegionGraph.drawRegion
(page 64). It is called by tracks.regionGraph.RegionGraph.drawRegions().

legendInfo ()
Return elements to show the legend: label, minimum value, middle value and maximum value.

JunctionItem class

class tracks.junctionGraph.JunctionItem
    Bases: tracks.graphicsItem.GraphicsItem(page 57), PySide.QtCore.QObject

    Meta-class for graphic elements to display junctions

toolTip ()
    Return the tooltip for the junction.

color ()
    Return the color in which the junction should be painted.

JunctionLineItem class

class tracks.junctionGraph.JunctionLineItem (junction, graph)
    Bases: tracks.junctionGraph.JunctionItem(page 60)

    Graphic element to display a junction as a lambda-like line.

    __init__ (junction, graph)

    boundingRect ()
    Return the bounding rectangle.

    paint (painter, option, widget=None)
    Paint the line.

2.11.4 numericGraph module

This module defines the class NumericGraph (page 61) to display numeric plots.

Classes:

41http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
**NumericGraph class**

class tracks.numericGraph.NumericGraph(dataset, trackManager, viewType=None)

Bases: tracks.track.Track

```
viewTypes = [('histogram', 'Histogram')]
List of the possible view types. The first one is the default view.
```

**initializeView()**
Initialize the view. Reimplements tracks.track.Track.initializeView to also initialize the legend.

**initializeScene()**
Initialize the scene. Reimplements tracks.track.Track.initializeScene to also initialize the cache and the width.

**draw(drawingId=None)**
Draw the plot

**clear(chromosome=None, step=0)**
Reset the cache and cache information.

---

**2.11.5 readsGraph module**

This module defines the class ReadsGraph to display mapped reads.

Classes:

**CoverageColumn class**

class tracks.readsGraph.CoverageColumn(chromosome, position, step, values, maxValue, graph)

Bases: tracks.graphicsItem.GraphicsItem

```
__init__(chromosome, position, step, values, maxValue, graph)
Initialize the column for the coverage graph. The locus of the column should be given by chromosome and position and its extension by step. The parameter values is a list of 2-element tuples, the first element is the base and the second the read count. This list may have only one element, indicating all bases are considered indifferently. The scale of the graph is fixed by the given maximum value (maxValue).
```

**boundingRect()**
Return the bounding rectangle.

**paint(painter, option, widget=None)**
Paint nothing.

**toolTip()**
Return the text for the tool tip.

**CoverageColumnPart class**

class tracks.readsGraph.CoverageColumnPart(y, width, height, color, parent)

Bases: tracks.graphicsItem.GraphicsItem

QGraphicsItem which describes a part of the coverage column (CoverageColumn). They are the drawn parts of the column. Each part represents one base, or there is only what part describing all bases at once.

---

42http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
__init__(y, width, height, color, parent)
    Initialization.

boundingRect()
    Return the bounding rectangle

paint(painter, option, widget=None)
    Paint the histogram bar.

toolTip()

ReadsGraph class

class tracks.readsGraph.ReadsGraph(reads, trackManager, viewType=None)
    Bases: tracks.track.Track (page 66)
    Widget to display mapped reads.

viewTypes = [('coverage', 'Coverage graph'), ('reads', 'Reads'), ('heatRegions', 'Heatmap')]
    List of the possible view types. The first one is the default view.

heatmapTypes = ['Read count', 'FPKM', 'log10 (FPKM + 1)', 'log10^2 (FPKM + 1)']
    Types of value used to compute the colors of the heatmap.

instances = []
    List of all instances of ReadsGraphs. Each graph needs to know the other to have a common color scale.

parameters = {'columns': None, 'lastDrawingId': '', 'annotations': None, 'trackManager': None, 'heatmapType': None}
    Parameters for the display.

__init__(reads, trackManager, viewType=None)

initializeView(height=None)
    Initialize the view. Re-implement tracks.track.Track.initializeView (page 67) to set a tracks.views.ReadsView if a heatmap is required.

initializeScene()
    Initialize the scene. Reimplement tracks.track.Track.initializeScene (page 67) to also initialize the cache for the view type ‘reads’.

__len__()
    Return the number of nucleotides defined in the track.

draw(drawingId=None)
    Draw the graph. This method call the method corresponding to the value of self.viewType

drawHeatRegions(drawingId)
    Draw the annotations of another track, with color depending on the read count.

drawReadCoverage()
    Draw a line plot of the read count.

drawReads()
    Draw the reads

clear()
    Clear the scene.
redrawNucleotides()
In reads view type, redraw the scene completely, including nucleotide content of the reads. This is useful when the reference sequence has changed and the nucleotides should be refreshed to match the new reference sequence.

setSceneRect (chrom=None)
Set the scene rect to be able to display the data of the read alignment.

static updateColumns (computeLength=False)
Update the columns for the heatmaps. Columns are defined as non-overlapping exons or overlapping areas of exons, i.e. each column is a part of the genome over which the number of annotated exons doesn’t change. Columns with a width smaller than 20 nucleotides are discarded.

The result is stored in the class variable columns. It consists in a list of tuples. Each tuple describes a column, but the tuple differ depending on the parameter computeLength. If computeLength is True, the tuple is (exonStart, exonEnd, geneSet, adjustedRelativeWidth). Else it is only (exonStart, exonEnd, geneSet). The result adjustedRelativeWidth is the length of the exon as it should be displayed.

updateValues()
Update the attribute values with the values to be displayed in each column. The limits of the columns are given in ReadsGraph.parameters[‘columns’]. The type of value is given by ReadsGraph.heatmapType().

setQueryType (viewType)
Subclass of Track.setViewType() to check for annotations in heatmap view.

cHECKAnnotations()
Check for the presence of annotations and print an error if there are no annotation when in heatmap view.

classmethod heatmapType ()
Return the type of value used to compute colors in heatmaps.

setHeatmapType (heatmapType)
Set the given heatmapType as the type of value used to compute colors in heatmaps. Redraw the heatmap if there was a change.

setAnnotations (annotations)
Set the given list of annotations as the reference to compute columns in heatmaps.

static getAnnotations ()
Return the list of annotation datasets which should be used as the reference to compute columns in heatmaps.

If no annotations has been set using setAnnotations(), the annotations are either all the instances of session.annotation.Annotation (page 43) or, if there are no such instance, all the instances of session.regions.Regions (page 52) or session.transcripts.Transcripts (page 55).

legendInfo ()
Return elements to show the legend: label, minimum value, middle value and maximum value.

### 2.11.6 regionGraph module

This module defines the class RegionsGraph to display genome regions. It also contains the definitions of the classes RegionItem and RegionLabel, which are two subclasses of QGraphicsItem used to draw the regions.

Classes:
RegionGraph class

class tracks.regionGraph.RegionGraph (regions, trackManager, viewType=None)

Bases: tracks.track.Track (page 66)

Widget to display genome regions

viewTypes = [('expanded', 'Expanded view'), ('collapse', 'Collapse view')]

List of the possible view types. The first one is the default view.

__init__ (regions, trackManager, viewType=None)

initializeScene ()

Initialize the variables necessary for the region graph.

draw (drawingId=None)

Draw the regions that should be visible in the view

drawCollapseRegions (start=0, end=1000)

Draw the regions which are mapped to the sequence from start to end. If the regions have already been
drawn, do not draw them again. All regions are drawn on the same line.

drawExpandedRegions (start=0, end=1000)

Draw the regions which are mapped to the sequence from start to end. If the regions have already been
drawn, do not draw them again. The vertical coordinate of the regions is adjusted so that the regions don’t
overlap.

drawRegion (region)

Draw a given region on the scene. Return the instance of tracks.regionGraph.RegionItem
(page 64).

updateChromosome ()

Update region list with the regions of the selected chromosome and restore initial state.

computeLength ()

Compute length of the genome described by the regions

clear ()

Erase all drawn regions

RegionItem class

class tracks.regionGraph.RegionItem (region, graph)

Bases: tracks.graphicsItem.GraphicsItem (page 57), PySide.QtCore.QObject

Graphic element to display a genome region.

boxHeight = 16.0

Height for the graphic element depicting the region excluding label

__init__ (region, graph)

toolTip ()

Set the tooltip for the region.

boundingRect ()

Return the bounding rectangle.

paint (painter, option, widget=None)

Paint the region.

43 http://srinikom.github.com/pyside-docs/PySide/QtCore/QObject.html#PySide.QtCore.QObject
color()
    Return the color in which the region should be painted.

hoverEnterEvent (event=None)
    Highlight when mouse hovers over.

hoverLeaveEvent (event=None)
    Remove the highlight when mouse leaves.

RegionLabel class

class tracks.regionGraph.RegionLabel (region, parent, graph)
    Bases: PySide.QtGui.QGraphicsItem

Legend for a region.
    __init__ (region, parent, graph)

boundingRect ()
    Return the bounding rectangle.

paint (painter, option, widget=None)
    Paint the label.

hoverEnterEvent (event=None)
    Highlight when mouse hovers over.

hoverLeaveEvent (event=None)
    Remove the highlight when mouse leaves.

2.11.7 scene module

This module defines the class GraphicsScene (page 65).

Classes:

GraphicsScene class

class tracks.scene.GraphicsScene (parent)
    Bases: PySide.QtGui.QGraphicsScene

Subclass of QGraphicsScene which removes the highlight rectangle when the scene is cleared.
    __init__ (parent)
        Initialize the scene.

clear ()
    Clear the scene’s content.

event (event)
    Handle events, especially tooltips display

---

44http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsItem.html#PySide.QtGui.QGraphicsItem
45http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsScene.html#PySide.QtGui.QGraphicsScene
46http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsScene.html#PySide.QtGui.QGraphicsScene
2.11.8 sequenceGraph module

This module defines the class `SequenceGraph` (page 66) to display nucleotide sequences.

Classes:

**SequenceGraph class**

```python
class tracks.sequenceGraph.SequenceGraph (sequence, trackManager, viewType=None)
    Bases: tracks.track.Track (page 66)

    Widget to display a reference nucleotide sequence.

    viewTypes = [('sequence', 'Nucleotide sequence')]
    List of the possible view types. The first one is the default view.

    __init__ (sequence, trackManager, viewType=None)

    drawSequence (chromosome, start=0, end=100)
    Draw a part of the nucleotide sequence, from start to end. If the nucleotides have already been drawn, do not draw them again.

    draw (drawingId='')
    Draw the part of the sequence visible in the view.

    updateChromosome ()
    Update the graph with the selected chromosome.

    clear ()
    Clear the graph.

    highlight (start=None, end=None)
    Sequence graph can't be highlighted
```

2.11.9 track module

This module defines tracks. A track is made of an instance of a subclass of `Track` (page 66). It has a layout with an instance of `TrackHandle` (page 68) on the left and an instance of `tracks.views.GraphicsView` (page 72) on the right.

Classes:

**Track class**

```python
class tracks.track.Track (dataset, trackManager, viewType=None)
    Bases: PySide.QtGui.QSplitter

    Class defining a track in the visualization program.

    Note: This is a virtual class, and some methods need to be implemented in the subclasses.

    viewTypes = []
    List of the possible view types. The first one is the default view.

    Note: Need to be implemented in the subclasses.
```

---

scaleChanged = <PySide.QtCore.Signal object at 0x05B39FC0>

Emitted when the scale of the graph has changed.

__init__(dataset, trackManager, viewType=None)

Initialize the track.

initializeView(heigt=None)

Initialize the view. Default implementation creates a tracks.views.GraphicsView (page 72) which allows moving and scaling.

initializeScene()

Initialize the scene. Default implementation creates a QGraphicsScene.

draw(drawingId=None)

Draw the track.

Note: Need to be implemented in the subclasses.

update()

Schedule a repaint event for the visible part of the scene.

__len__()

Return the number of nucleotides defined in the track.

Note: Subclasses must define the attribute self.length.

classmethod defaultViewType()

Return the default type of view.

classmethod setDefaultViewType(viewType)

Set the default type of view to be viewType. Return a boolean indicating if the default type has changed.

classmethod defaultHeatmapType()

Return the default type of value to compute colors in heatmaps. If no other value has been chosen in the settings, the value is the last one of heatmapTypes.

classmethod setDefaultHeatmapType(heatmapType)

Set the default type of value to compute colors in heatmaps. Return a boolean indicating if the default type has changed.

highlight(start=None, end=None)

Highlight the region of the track from start to end. If end is None, only the nucleotide at index start will be highlighted. If start is also None, the highlight will be removed. This default implementation draws a yellow rectangle in the background.

drawIntronLimits()

In ‘hide introns’ mode, show vertical lines to show intron limits.

contextMenuEvent(event)

hasViewType(viewType)

Return a boolean indicating if the given viewType is supported by the track.

setWidthType(viewType)

Change the type of graph displayed in the track.

---

48http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsScene.html#PySide.QtGui.QGraphicsScene
clear()  
  Clear the scene.

dragEnterEvent (event)  
  Specify if a widget can be dragged here.

dropEvent (event)  
  Move the track when the user drops it.

setData (name=None, color=None)  
  Set track information.

saveAsImage ()  
  Save the track as an image

enterEvent (*args, **kwargs)  
  Show the legend when pointing to the track

leaveEvent (*args, **kwargs)  
  Hide the legend when pointing out of the track

legendInfo ()  
  Return elements to show the legend: label, minimum value, middle value and maximum value.

**TrackHandle class**

class tracks.track.TrackHandle (track)  
  Bases: PySide.QtGui.QWidget

Define a colored area next to the track. User can drag this handle to move the track.

__init__ (track)  
  Initialize the handle for the given dataset.

sizeHint ()

refreshData ()  
  Set the color according to the given dataset.

resizeEvent (event)

paintEvent (QPaintEvent)  
  Paint the area.

mousePressEvent (event)  
  Log position when the user clicks on the handle.

mouseMoveEvent (event)  
  Start dragging if appropriate.

**2.11.10 trackManager module**

This module defines the class TrackManager (page 69).

Classes:

49<http://srinikom.github.com/pyside-docs/PySide/QtGui/QWidget.html#PySide.QtGui.QWidget>
**TrackManager class**

```python
class TrackManager(window)
    Bases: PySide.QtCore.QObject
```

Manager for the tracks displayed in the MainUI

- **trackAdded** = `<PySide.QtCore.Signal object at 0x0636A600>`
  Emitted when a new track is added.

- **trackRemoved** = `<PySide.QtCore.Signal object at 0x0636AAC0>`
  Emitted when a track is removed.

- **viewTypeChanged** = `<PySide.QtCore.Signal object at 0x0636A4C0>`
  Emitted when the type of view of a track has changed. The name of the category is transmitted.

- **visibleAreaChanged** = `<PySide.QtCore.Signal object at 0x06365B80>`
  Emitted when the area displayed by the tracks has changed.

- **levelOfDetailsChanged** = `<PySide.QtCore.Signal object at 0x06365740>`
  Emitted when the level of details has changed.

- **__init__**(window)

- **addTrack**(dataset, viewType=None, index=None)
  Add a track. If viewType is provided, it will be the view type of the newly created track. If index is provided, the track will be positioned at the given index. Return the newly created track.

- **removeTrack**(track)
  Remove a track.

- **drawAll**(dataType=None, viewType=None, drawingId=None)
  Draw all the tracks. If dataType is not None, only the tracks which display datasets of type dataType are drawn. If viewType is not None, only the tracks which display graphs of type viewType are drawn.

  The drawingId is usually computed internally. It avoids drawing several times the same time, by checking if the drawingId has changed. If specified, the given drawingId is used.

- **highlightAll**(start=None, end=None)
  Highlight the region of all tracks from start to end. If end is None, only the nucleotide at index start will be highlighted. If start is also None, the highlight will be removed.

- **switchArea**(chromosome=None, start=None, end=None, forceChange=False)
  Moves the view of all tracks to display the area from start to end on the given chromosome. If start and end are None, the default position will be used. Unless if forceChange is True, the tracks are updated only if the coordinates have changed. Return True if the coordinates have been accepted or False if not.

- **updateVisibleArea**(view, noRedraw=False)
  Update the index of the leftmost and rightmost nucleotides which can be seen on the view. Except if noRedraw is True, the tracks are updated if the visible area has changed.

- **visibleChromArea**(extended=False)
  Return the displayed chromosome, leftmost nucleotide index and rightmost nucleotide index. If extended is True, the area also includes one screen-length on left and one screen-length on right

- **zoom**(factor, tracks=None)
  Scale all the tracks by the given factor. If tracks is a list of tracks, only these tracks are scaled.

- **move**(factor)
  Translate all the view by a screen length multiplied by the factor.
**transformAll** *(refView, doNotUpdate=False)*
Update all the views to have the same viewport transformation as the view given in argument. The tracks created in the future will also have this same viewport.

**transformLike** *(tracks, refView=None)*
Transform all the tracks so that their views have the same viewport transformation as the refview. If no refview is specified, the refView of the track manager is used.

**setSceneWidth** *(newWidth, tracks=None, unit=None)*
Update the scene width of all the elements of the list of tracks tracks, so as they display at least newWidth nucleotides.

If unit is set to 'pixels', newWidth is considered as a width in pixels instead of a number of nucleotides.

If tracks is None, all the tracks are considered.

**sceneWidth** *
Return the scene width of the tracks.

**updateLevelOfDetail** *(view)*
Updates the variable self.levelOfDetail.

**nucleotideWidth** *
Return the width of one nucleotide in default view

**nucleotideInfo** *
Return all the size information about the nucleotides

**setViewType** *(track, viewType)*
Change the type of view for the given track.

**setAllViewType** *(dataType, viewType)*
Change the type of graph for all the tracks which display datasets of type dataType.

**hide** *(track)*
Hide the given track.

**hideAll** *(dataType)*
Hide the graph of all the tracks which display datasets of type dataType.

**updateAll** *(dataTypes)*
Repaint the graph of all the tracks which display a dataset of a type listed in the list dataTypes.

**updatePositionStatus** *(positionStart=None, positionEnd=None)*
Update the label in the status bar to display the position of the cursor. The position (index of the nucleotide under the mouse) should be given in positionStart. If positionEnd is not None, then the position is a region from positionStart to positionEnd. If positionStart is None, the label is reset to blank state.

**redrawAfterResize** *(track, event)*
Plan a redraw of the given track. If multiple redraw requests are received within 500 ms, the track will be redrawn only once, 500 ms after the first request.

**setTrackHeight** *(height, track=None)*
Set the height of the given track to the given height in pixels. If track is None, all the tracks are resized.

**averageTrackHeight** *
Return the average height of the tracks

**setHandleWidth** *(width, _index)*
Set the width of the handle of the tracks to width.

**Note:** Tracks with fixed height (like DNA sequences) are not resized.
**setHideIntronsMode** *(boolean=None)*
Activate (if *boolean* is `True`) or disable (if *boolean* is `False`) the ‘hide introns’ mode. If *boolean* is `None`, the mode commutes.

**abscissa** *(x)*
Return the abscissa at which should be displayed an element at coordinate *x* on the current chromosome.
This take into account the mode with hidden introns.

**pixelToCoordinate** *(x)*
Return the coordinate along the current chromosome corresponding to the pixel at position *x* on the scene.
This method is the inverse of **abscissa**().

**isInfoDockVisible** ()
Tell if the dock to display items info is visible.

**populateInfoDock** *(scene, pos)*
Populate the dock to display information about the item at position *pos* on the given *scene*. Not more than one update happens every 200 ms to avoid lagging because **itemAt**() is slow.

## 2.11.11 transcriptGraph module

This module defines the class **TranscriptGraph** (page 71) to display transcripts. It also contains the definitions of the class **TranscriptItem** (page 72), a subclass of **QGraphicsItem** used to draw the transcripts.

Classes:

**TranscriptGraph class**

```python
class tracks.transcriptGraph.TranscriptGraph(*args, **kwargs)
    Bases: tracks.regionGraph.RegionGraph (page 64)
```

Widget to display transcripts

**heatmapTypes** = ['FPKM', 'log10 (FPKM + 1)', 'log10^2 (FPKM + 1)']
Types of value used to compute the colors of the transcripts.

**__init__** (*args, **kwargs*)

**drawCollapseRegions** *(start=0, end=1000)*
Draw the transcripts which are mapped to the sequence from *start* to *end*. If the transcripts have already been drawn, do not draw them again. All transcripts are drawn on the same line.

**drawExpandedRegions** *(start=0, end=1000)*
Draw the transcripts which are mapped to the sequence from *start* to *end*. If the transcripts have already been drawn, do not draw them again. The vertical coordinate of the transcripts is adjusted so that the regions don’t overlap.

**drawRegion** *(transcript)*
Draw a given transcript on the scene. Return the instance of **tracks.transcriptGraph.TranscriptItem** (page 72).

**computeMinMax** *(start, end)*
Define max score and min FPKM, which are used to define the color.

**classmethod** **defaultViewType** ()
Return the default type of view (expanded).

**heatmapType** ()
Return the type of value used to compute colors.
**setHeatmapType** (*heatmapType*)
Set the given *heatmapType* as the type of value used to compute colors of the transcripts. Redraw the transcripts if there was a change.

**legendInfo**()
Return elements to show the legend: label, minimum value, middle value and maximum value.

**TranscriptItem class**

```python
class tracks.transcriptGraph.TranscriptItem(*args, **kwargs)
```
Bases: `tracks.regionGraph.RegionItem` (page 64)

Graphic element to display an annotation.

```
__init__(*args, **kwargs)
```
Initialize the transcript.

```
toolTip()
```
Return the data for the tool tip.

```
color()
```
Return the color in which the junction should be painted.

### 2.11.12 views module

This module defines the views (subclasses of `QGraphicsView`\(^{51}\)) which are used by the instances of `Track` (page 66) to display the graphs.

Classes:

**GraphicsView class**

```python
class tracks.views.GraphicsView(graph, trackManager=None, height=None)
```
Bases: `PySide.QtGui.QGraphicsView`\(^{52}\)

Subclass of `QGraphicsView`\(^{53}\) widget to display genomic data.

```
__init__(graph, trackManager=None, height=None)
```
Initialize the view. If `height` is provided, the view is initialized with the given `height`.

```
sizeHint()
```
Return the preferred size for the view.

```
wheelEvent(event)
```
Override `wheelEvent()` to let zoom all the instances of `GraphicsView` (page 72) at once when the mouse wheel is rolled over one of them.

```
mouseDoubleClickEvent(event)
```
Zoom on the clicked part of the view

```
mousePressEvent(event)
```
Synchronize view when dragging with ScrollHandDrag

```
mouseMoveEvent(event)
```
Display the position of the nucleotide under the mouse.

\(^{51}\)http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsView.html#PySide.QtGui.QGraphicsView
\(^{52}\)http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsView.html#PySide.QtGui.QGraphicsView
\(^{53}\)http://srinikom.github.com/pyside-docs/PySide/QtGui/QGraphicsView.html#PySide.QtGui.QGraphicsView
**disableHighlight**(disable=True)
Don’t automatically highlight current position anymore.

**leaveEvent**(event)
Reset the mouse position status when the mouse leave the view.

**setSceneRect**(w=None, h=None, unit=None)
Set the dimensions of the rectangle of the view’s scene. The default unit for the width \( w \) is a number of nucleotides, but it can be changed to a number of pixels if \( unit \) is set to \( \text{’pixels’} \). The unit for the height \( h \) is a number of pixels. Both arguments are optional. If a dimension is not provided, it size will not be changed.

If the dimension is provided in nucleotides, 10 more nucleotides are added as a margin.

**dragEnterEvent**(event)
Drag and drop methods should be handled by the track, not the view.

**dragLeaveEvent**(event)
Drag and drop methods should be handled by the track, not the view.

**dragMoveEvent**(event)
Drag and drop methods should be handled by the track, not the view.

**dropEvent**(event)
Drag and drop methods should be handled by the track, not the view.

**resizeEvent**(event)

**fitWidth**(x, y)
Fits the view to the area from \( x \) to \( y \). This is a simplified re-implementation of `PySide.QtGui.QGraphicsView.fitInView()` optimized to avoid approximations. Moreover, the final scale can’t be greater than 1. Return the level of detail.
PYTHON MODULE INDEX

a
additionalUI, 27
additionalUI.about, 27
additionalUI.dialog, 28
additionalUI.globalSettings, 29
additionalUI.printOut, 30
additionalUI.search, 30
additionalUI.trackHeight, 31
additionalUI.trackSettings, 31
f
freeze, 32
functions, 33
g
generateDoc, 35
l
lang, 35
lang.generateTs, 35
m
mainUI, 36
mainUI.dataTree, 36
mainUI.infoDock, 39
mainUI.legendDock, 39
mainUI.mainWindow, 39
mainUI.toolBar, 41
mainUI.trackSplitter, 42
q
qrc_resources, 42
r
RNaseqViewer, 27
s
session, 43
session.annotation, 43
session.dataset, 46
session.dataTypes, 45
session.junctions, 47
session.numeric, 47
session.reads, 48
session.regions, 51
session.save, 52
session.sequence, 53
session.session, 53
session.transcripts, 54
setup, 55
t
tracks, 55
tracks.annotationGraph, 55
tracks.graphicsItem, 56
tracks.junctionGraph, 59
tracks.numericGraph, 60
tracks.readsGraph, 61
tracks.regionGraph, 63
tracks.scene, 65
tracks.sequenceGraph, 66
tracks.track, 66
tracks.trackManager, 68
tracks.transcriptGraph, 71
tracks.views, 72
Index
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>boundingRect()</code></td>
<td>(tracks.readsGraph.CoverageColumn method), 61</td>
</tr>
<tr>
<td><code>boundingRect()</code></td>
<td>(tracks.readsGraph.CoverageColumnPart method), 62</td>
</tr>
<tr>
<td><code>boundingRect()</code></td>
<td>(tracks.regionGraph.RegionItem method), 64</td>
</tr>
<tr>
<td><code>boundingRect()</code></td>
<td>(tracks.regionGraph.RegionLabel method), 65</td>
</tr>
<tr>
<td><code>boxHeight</code></td>
<td>(tracks.regionGraph.RegionItem attribute), 64</td>
</tr>
<tr>
<td><code>brush</code></td>
<td>(tracks.graphicsItem.BlankReadItem attribute), 56</td>
</tr>
<tr>
<td><code>brushHover</code></td>
<td>(tracks.graphicsItem.BlankReadItem attribute), 56</td>
</tr>
<tr>
<td><code>buttonClicked()</code></td>
<td>(additionalUI.globalSettings.GlobalSettings method), 29</td>
</tr>
<tr>
<td><code>callSamtools()</code></td>
<td>(in module functions), 34</td>
</tr>
<tr>
<td><code>callTabix()</code></td>
<td>(in module functions), 34</td>
</tr>
<tr>
<td><code>categoryContentChanged</code></td>
<td>(mainUI.dataTree.DataTreeModel attribute), 37</td>
</tr>
<tr>
<td><code>categoryFromIndex()</code></td>
<td>(mainUI.dataTree.DataTreeModel method), 37</td>
</tr>
<tr>
<td><code>categoryRole</code></td>
<td>(mainUI.dataTree.DataTreeModel attribute), 37</td>
</tr>
<tr>
<td><code>checkAnnotations()</code></td>
<td>(tracks.readsGraph.ReadsGraph method), 63</td>
</tr>
<tr>
<td><code>chromID()</code></td>
<td>(session.dataset.Dataset method), 46</td>
</tr>
<tr>
<td><code>chromName()</code></td>
<td>(in module functions), 34</td>
</tr>
<tr>
<td><code>chromosomeList()</code></td>
<td>(session.session.Session method), 54</td>
</tr>
<tr>
<td><code>classParameters</code></td>
<td>(additionalUI.dialog.Dialog attribute), 28</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>(tracks.numericGraph.NumericGraph method), 61</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>(tracks.readsGraph.ReadsGraph method), 62</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>(tracks.regionGraph.RegionGraph method), 64</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>(tracks.scene.GraphicsScene method), 65</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>(tracks.sequenceGraph.SequenceGraph method), 66</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>(tracks.track.Track method), 67</td>
</tr>
<tr>
<td><code>clearOldCaches()</code></td>
<td>(session.reads.NucleotideCache method), 49</td>
</tr>
<tr>
<td><code>closeEvent()</code></td>
<td>(mainUI.mainWindow.MainWindow method), 40</td>
</tr>
<tr>
<td><code>closeSession()</code></td>
<td>(mainUI.mainWindow.MainWindow method), 40</td>
</tr>
<tr>
<td><code>color()</code></td>
<td>(tracks.junctionGraph.JunctionBarItem method), 59</td>
</tr>
<tr>
<td><code>color()</code></td>
<td>(tracks.junctionGraph.JunctionItem method), 60</td>
</tr>
<tr>
<td><code>color()</code></td>
<td>(tracks.regionGraph.RegionItem method), 64</td>
</tr>
<tr>
<td><code>color()</code></td>
<td>(tracks.transcriptGraph.TranscriptItem method), 72</td>
</tr>
<tr>
<td><code>colorChanged</code></td>
<td>(additionalUI.trackSettings.ColorPushButton attribute), 31</td>
</tr>
<tr>
<td><code>colorPalette</code></td>
<td>(functions.RandomColor attribute), 35</td>
</tr>
<tr>
<td><code>colorPalettes</code></td>
<td>(in module tracks.graphicsItem), 56</td>
</tr>
<tr>
<td><code>colorPreview()</code></td>
<td>(in module functions), 33</td>
</tr>
<tr>
<td><code>ColorPushButton</code></td>
<td>(class in additionalUI.trackSettings), 31</td>
</tr>
<tr>
<td><code>columnCount()</code></td>
<td>(mainUI.dataTree.DataTreeModel method), 37</td>
</tr>
<tr>
<td><code>computeFileInfo()</code></td>
<td>(session.sequence.Sequence method), 53</td>
</tr>
<tr>
<td><code>computeLength()</code></td>
<td>(tracks.regionGraph.RegionGraph method), 64</td>
</tr>
<tr>
<td><code>computeMinMax()</code></td>
<td>(tracks.transcriptGraph.TranscriptGraph method), 71</td>
</tr>
<tr>
<td><code>contextMenuEvent()</code></td>
<td>(tracks.track.Track method), 67</td>
</tr>
<tr>
<td><code>CoverageColumn</code></td>
<td>(class in tracks.readsGraph), 61</td>
</tr>
<tr>
<td><code>CoverageColumnPart</code></td>
<td>(class in tracks.readsGraph), 61</td>
</tr>
<tr>
<td><code>createAction()</code></td>
<td>(mainUI.mainWindow.MainWindow method), 40</td>
</tr>
<tr>
<td><code>currentIndex</code></td>
<td>(functions.RandomColor attribute), 35</td>
</tr>
<tr>
<td><code>data()</code></td>
<td>(additionalUI.trackSettings.DatasetListModel method), 32</td>
</tr>
<tr>
<td><code>data()</code></td>
<td>(mainUI.dataTree.DataTreeModel method), 37</td>
</tr>
<tr>
<td><code>dataChanged</code></td>
<td>(session.session.Session attribute), 53</td>
</tr>
<tr>
<td><code>Dataset</code></td>
<td>(class in session.dataset), 46</td>
</tr>
<tr>
<td><code>datasetAdded</code></td>
<td>(session.session.Session attribute), 53</td>
</tr>
<tr>
<td><code>datasetRole</code></td>
<td>(additionalUI.trackSettings.DatasetListModel attribute), 32</td>
</tr>
<tr>
<td><code>DatasetNotValid</code></td>
<td>(class in session.dataset), 47</td>
</tr>
<tr>
<td><code>datasetRemoved</code></td>
<td>(session.session.Session attribute), 53</td>
</tr>
<tr>
<td><code>defaultBrush</code></td>
<td>(in module tracks.graphicsItem), 56</td>
</tr>
<tr>
<td><code>defaultColor</code></td>
<td>(in module tracks.graphicsItem), 56</td>
</tr>
<tr>
<td><code>defaultHeatmapType()</code></td>
<td>(tracks.track.Track class method), 67</td>
</tr>
<tr>
<td><code>defaultPen</code></td>
<td>(in module tracks.graphicsItem), 56</td>
</tr>
<tr>
<td><code>defaultViewType()</code></td>
<td>(tracks.track.Track class method), 67</td>
</tr>
<tr>
<td><code>defaultViewType()</code></td>
<td>(tracks.transcriptGraph.TranscriptGraph class method), 71</td>
</tr>
<tr>
<td><code>deleteTrack()</code></td>
<td>(mainUI.dataTree.DataTreeMenu method), 37</td>
</tr>
<tr>
<td><code>Dialog</code></td>
<td>(class in additionalUI.dialog), 28</td>
</tr>
<tr>
<td><code>disableHighlight()</code></td>
<td>(tracks.views.GraphicsView method), 72</td>
</tr>
<tr>
<td><code>docPath</code></td>
<td>(in module generateDoc), 35</td>
</tr>
<tr>
<td><code>download()</code></td>
<td>(session.save.SessionSaving method), 52</td>
</tr>
<tr>
<td><code>dragEnterEvent()</code></td>
<td>(mainUI.trackSplitter.SplitterSpacer method), 42</td>
</tr>
</tbody>
</table>
dragEnterEvent() (tracks.track.Track method), 68
dragEnterEvent() (tracks.views.GraphicsView method), 73
dragLeaveEvent() (tracks.views.GraphicsView method), 73
dragMoveEvent() (tracks.views.GraphicsView method), 73
draw() (tracks.junctionGraph.JunctionGraph method), 60
draw() (tracks.readsGraph.ReadsGraph method), 62
draw() (tracks.regionGraph.RegionGraph method), 64
draw() (tracks.sequenceGraph.SequenceGraph method), 66
draw() (tracks.track.Track method), 67
drawAll() (tracks.trackManager.TrackManager method), 67
drawBars() (tracks.junctionGraph.JunctionGraph method), 60
drawCollapseRegions() (tracks.regionGraph.RegionGraph method), 64
drawCollapseRegions() (tracks.transcriptGraph.TranscriptGraph method), 71
drawExpandedRegions() (tracks.regionGraph.RegionGraph method), 64
drawExpandedRegions() (tracks.transcriptGraph.TranscriptGraph method), 71
drawHeatRegions() (tracks.readsGraph.ReadsGraph method), 62
drawIntronLimits() (tracks.track.Track method), 67
drawLines() (tracks.junctionGraph.JunctionGraph method), 60
drawReadCoverage() (tracks.readsGraph.ReadsGraph method), 62
drawReads() (tracks.readsGraph.ReadsGraph method), 62
drawRegion() (tracks.annotationGraph.AnnotationGraph method), 55
drawRegion() (tracks.junctionGraph.JunctionGraph method), 60
drawRegion() (tracks.regionGraph.RegionGraph method), 64
drawRegion() (tracks.transcriptGraph.TranscriptGraph method), 71
drawSequence() (tracks.sequenceGraph.SequenceGraph method), 66
dropEvent() (mainUI.trackSplitter.SplitterSpacer method), 42
dropEvent() (tracks.track.Track method), 68
dropEvent() (tracks.views.GraphicsView method), 73
duplicateTrack() (mainUI.dataTree.DataTreeMenu method), 37
editorEvent() (mainUI.dataTree.DataTreeDelegate method), 36
getSavePath() (additionalUI.dialog.Dialog static method), 28
getSeq() (session.sequence.Sequence method), 53
getSequence() (session.sequence.Session method), 54
getTypeByClass() (in module session.dataTypes), 45
typeByclass() (in module session.dataTypes), 45
typeByName() (in module session.dataTypes), 45
typeByTrack() (in module session.dataTypes), 45
typeInfo() (in module session.dataTypes), 45
getValues() (session.numeric.FixedStepNumeric method), 47
getValues() (session.numeric.Numeric method), 48
getValues() (session.numeric.VariableStepNumeric method), 48
getValues() (session.reads.NucleotideCache method), 49
getValues() (session.reads.NucleotideCacheBlock method), 49
GlobalSettings (class in additionalUI.globalSettings), 29
GraphicsItem (class in tracks.graphicsItem), 57
GraphicsScene (class in tracks.scene), 65
GraphicsView (class in tracks.views), 72
GtfAnnotation (class in session.annotation), 44
GtfGzAnnotation (class in session.annotation), 44
hasViewType() (tracks.track.Track method), 67
hcl2rgb() (in module functions), 33
heatColors() (in module tracks.graphicsItem), 56
HeatmapBar (class in tracks.graphicsItem), 57
heatmapType() (tracks.readsGraph.ReadsGraph class method), 63
heatmapType() (tracks.readsGraph.transcriptGraph.TranscriptGraph method), 71
heatmapTypes (tracks.readsGraph.ReadsGraph method), 62
heatmapTypes (tracks.transcriptGraph.TranscriptGraph method), 71
heatmaps (tracks.readsGraph.ReadsGraph method), 71
hide() (tracks.trackManager.TrackManager method), 70
hideAll() (tracks.trackManager.TrackManager method), 70
hideAllRequested (mainUI.dataTree.DataTreeDelegate method), 36
hideLegend() (mainUI.legendDock.LegendDock method), 39
hideRequested (mainUI.dataTree.DataTreeDelegate method), 36
highlight() (tracks.sequenceGraph.SequenceGraph method), 66
highlight() (tracks.track.Track method), 67
highlightAll() (tracks.trackManager.TrackManager method), 69
HistogramColumn (class in tracks.graphicsItem), 57
hoverEnterEvent() (tracks.graphicsItem.insertionNucleotide method), 58
hoverEnterEvent() (tracks.regionGraph.RegionItem method), 65
hoverEnterEvent() (tracks.regionGraph.RegionLabel method), 65
hoverLeaveEvent() (tracks.graphicsItem.heatmapBar method), 57
hoverLeaveEvent() (tracks.graphicsItem.insertionNucleotide method), 58
hoverLeaveEvent() (tracks.regionGraph.RegionItem method), 65
hoverLeaveEvent() (tracks.regionGraph.RegionLabel method), 65
importClass() (in module functions), 33
indexOfCategory() (mainUI.dataTree.DataTreeModel method), 37
indexFile() (session.annotation.GtfAnnotation method), 44
initializeClass() (tracks.graphicsItem.Nucleotide static method), 58
initializeScene() (tracks.numericGraph.NumericGraph method), 61
initializeScene() (tracks.readsGraph.ReadsGraph method), 62
initializeScene() (tracks.regionGraph.RegionGraph method), 64
initializeScene() (tracks.track.Track method), 67
initializeView() (tracks.numericGraph.NumericGraph method), 61
initializeView() (tracks.readsGraph.ReadsGraph method), 62
initializeView() (tracks.track.Track method), 67
insertWidget() (mainUI.trackSplitter.TrackSplitter method), 42
instance() (functions.Pool static method), 34
instances (tracks.readsGraph.ReadsGraph attribute), 62
intronList() (session.session.Session method), 54
invalid (session.dataset.Dataset attribute), 46
InvalidAnnotation (class in session.annotation), 44
InvalidNumeric (class in session.numeric), 47
isInfoDockVisible() (tracks.trackManager.TrackManager method), 71
isSorted() (session.reads.Reads method), 51
JunctionBarItem (class in tracks.junctionGraph), 59
JunctionGraph (class in tracks.junctionGraph), 59
JunctionItem (class in tracks.junctionGraph), 60
Index 81
Index
sequencesChanged (session.session.Session attribute), 53
Session (class in session.session), 53
session (module), 43
session.annotation (module), 43
session.dataset (module), 46
session.dataTypes (module), 45
session.junctions (module), 47
session.numeric (module), 47
session.reads (module), 48
session.regions (module), 51
session.save (module), 52
session.sequence (module), 53
session.session (module), 53
session.transcripts (module), 54
SessionSaving (class in session.save), 52
setAllViewType() (tracks.trackManager.TrackManager method), 70
setAnnotations() (tracks.readsGraph.ReadsGraph method), 63
setChromList() (mainUI.toolBar.ToolBar method), 41
setChromosome() (mainUI.toolBar.ToolBar method), 41
setCurrentTrack() (mainUI.dataTree.DataTreeMenu method), 37
setData() (tracks.track.Track method), 68
setDefaultHeatmapType() (tracks.track.Track class method), 67
setDefaultSettings() (additionalUI.globalSettings.GlobalSettings static method), 29
setDefaultViewType() (tracks.track.Track class method), 67
setDictToolTip() (tracks.graphicsItem.GraphicsItem method), 57
setHandleWidth() (tracks.trackManager.TrackManager method), 70
setHeatmapType() (tracks.readsGraph.ReadsGraph method), 63
setHeatmapType() (tracks.transcriptGraph.TranscriptGraph method), 71
setHideIntronButtonState() (mainUI.toolBar.ToolBar method), 41
setHideIntronsMode() (tracks.trackManager.TrackManager method), 70
setPositionFormEnabled() (mainUI.toolBar.ToolBar method), 41
setProgress() (additionalUI.dialog.Dialog method), 28
setSceneRect() (tracks.readsGraph.ReadsGraph method), 63
setSceneRect() (tracks.views.GraphicsView method), 73
setSceneWidth() (tracks.trackManager.TrackManager method), 70
setStatus() (additionalUI.dialog.Dialog method), 28
SettingsPage (class in additionalUI.globalSettings), 29
setTrack() (mainUI.legendDock.LegendDock method), 39
setTrackHeight() (tracks.trackManager.TrackManager method), 70
setup (module), 55
setViewType() (tracks.readsGraph.ReadsGraph method), 63
setViewType() (tracks.track.Track method), 67
setViewType() (tracks.trackManager.TrackManager method), 70
setWindow() (additionalUI.dialog.Dialog method), 28
shape() (tracks.junctionGraph.JunctionBarItem method), 59
showError() (mainUI.toolBar.ToolBar method), 41
showInfo() (mainUI.infoDock.InfoDock method), 39
sizeHint() (tracks.track.TrackHandle method), 68
sizeHint() (tracks.views.GraphicsView method), 72
sortedNicely() (in module functions), 33
splitterSpace (class in mainUI.trackSplitter), 42
switchArea() (tracks.trackManager.TrackManager method), 69
testColors() (in module functions), 34
toolBar (class in mainUI.toolBar), 41
toolTip() (tracks.annotationGraph.AnnotationItem method), 56
toolTip() (tracks.graphicsItem.BlankReadItem method), 57
toolTip() (tracks.graphicsItem.Nucleotide method), 58
toolTip() (tracks.junctionGraph.JunctionBarItem method), 59
toolTip() (tracks.junctionGraph.JunctionItem method), 60
toolTip() (tracks.readsGraph.CoverageColumn method), 61
toolTip() (tracks.readsGraph.CoverageColumnPart method), 62
toolTip() (tracks.regionGraph.RegionItem method), 64
toolTip() (tracks.transcriptGraph.TranscriptItem method), 72
tr() (in module functions), 34
Track (class in tracks.track), 66
trackAdded (tracks.trackManager.TrackManager attribute), 69
trackFromIndex() (mainUI.dataTree.DataTreeModel method), 38
TrackHandle (class in tracks.track), 68
trackHeightUI (class in additionalUI.trackHeight), 31
TrackManager (class in tracks.trackManager), 69
trackRemoved (tracks.trackManager.TrackManager attribute), 69
trackRole (mainUI.dataTree.DataTreeModel attribute), 37