

# LHCb Configuration Database Visualizer

## LHCB Technical Note

Issue: v1r3  
Revision: 3

Reference: LHCB COMP 2006-028  
Created: 9<sup>th</sup> Jun 2006  
Last modified: 27 Oct 2006

**Prepared By:** Thomas Johansen, Lana Abadie, Eric van Herwijnen, Robbie Shade



## Abstract

The LHCb Configuration Database Visualizer (CdbVis) is a graphical database editor. It is written in Python and runs on both Windows and Linux. It is primarily intended to display the link hierarchy and properties of devices in the LHCb Online Configuration Database. The tool can be used to display, but also to modify or to add data in the database. It can be used for private databases.

## Document Status Sheet

Table 1 Document Status Sheet

<b>1. Document Title: [Project Name Qualification] User Requirements Document</b>			
<b>2. Document Reference Number: [Document Reference Number]</b>			
<b>3. Issue</b>	<b>4. Revision</b>	<b>5. Date</b>	<b>6. Reason for change</b>
v1r1	1	9 <sup>th</sup> jun 2006	First version
V1r2	2	22 sept 2006	Robbie's changes
v1r3	3	27 oct 2006	Lana's changes (spare device added + delete functions)

# Table of Contents

LHCb TECHNICAL NOTE.....	I
ISSUE: V1R1.....	I
TABLE 1 DOCUMENT STATUS SHEET.....	I
<b>1. PREFACE.....</b>	<b>1</b>
<b>2. INSTALLATION.....</b>	<b>2</b>
2.1 PROGRAM REQUIREMENTS .....	2
2.1.1 Linux/Unix (LXPlus/AFS) .....	2
2.1.2 Windows (NICE) .....	2
2.1.3 Requirements.....	2
2.2 INSTALLING ON LINUX (LXPLUS/AFS).....	3
2.2.1 Installing Python.....	3
2.2.2 Installing wxPython.....	3
2.2.3 Installing CdbVis .....	4
2.2.4 Installing Oracle Instant Client 10g .....	5
2.2.5 Installing ConfDB library .....	5
2.2.6 Testing Installation .....	5
2.3 INSTALLING ON WINDOWS (NICE) .....	6
2.3.1 Installing Python.....	6
2.3.2 Installing wxPython.....	7
2.3.3 Installing CdbVis .....	7
2.3.4 Installing Oracle Instant Client 10g .....	7
2.3.5 Installing ConfDB library .....	8
2.3.6 Testing Installation .....	8
<b>3. PROGRAM FEATURES .....</b>	<b>10</b>
3.1 NAVIGATION .....	10
3.1.1 Abstract Use-cases.....	10
3.2 CREATION .....	12
3.2.1 Abstract Use-cases.....	12
<b>4. PROGRAM FUNCTIONS - HOW TO USE THE PROGRAM.....</b>	<b>17</b>
4.1 STARTUP AND SHUTDOWN .....	17
4.2 OTHER USER MODE INDEPENDENT FEATURES.....	17
4.3 NAVIGATION MODE.....	18
4.3.1 Tree view/hierarchical navigation .....	18
4.3.2 Only view links/connections of a given link type and filter by spare parts .....	20
4.3.3 Information window .....	20
4.3.4 The Status bar.....	21
4.3.5 The menu bar .....	22
4.3.6 The tool bar.....	25
4.3.6 The visual window.....	26
4.4 CREATION MODE.....	28
4.4.1 The menu bar .....	28
4.4.2 The tool bar.....	47
4.4.3 The visual window.....	49

<b>5. KNOWN ISSUES.....</b>	<b>51</b>
5.1 KNOWN BUGS/PROBLEMS.....	51
5.1.1 General .....	51
5.1.2 Unix/Linux .....	51
5.1.3 Windows.....	51
5.2 MISSING FEATURES .....	52
<b>6. TERMINOLOGY/DEFINITIONS – USER DOCUMENTATION.....</b>	<b>53</b>
<b>7. A DEVELOPER'S GUIDE TO CDBVIS.....</b>	<b>55</b>
7.1 CDBVIS.PY .....	56
7.1.1 Class mainWindow inherits wxFrame implements dbUpdate.....	56
7.1.2 Class mainPanel inherits wxPanel implements dbUpdate.....	60
7.1.3 Class cdbVis inherits wxApp.....	60
7.2 CDBVISCORE.PY .....	61
7.2.1 Class dbUpdate.....	61
7.3 OBJECTCLASSES.PY .....	61
7.3.1 Class DBInfo implements dbUpdate.....	61
7.3.2 Class DBSystem implements dbUpdate.....	63
7.3.3 Class SubSystem inherits DBInfo.....	63
7.3.4 Class DeviceType inherits DBInfo.....	63
7.3.5 Class Device inherits DBInfo.....	63
7.3.5 Class Link inherits DBInfo.....	63
7.3.6 Class LinkType inherits DBInfo.....	64
7.3.7 Class Port inherits DBInfo.....	64
7.4 VISWINDOW.PY.....	64
7.4.1 Class visWindow inherits wxScrolledWindow implements dbUpdate.....	64
7.4.2 Class DrawingObject.....	67
7.5 SELECTPANEL.PY .....	68
7.5.1 Class selectWindow inherits wxPanel, wxColumnSorterMixin implements dbUpdate.....	68
7.6 AREAVIEW.PY.....	69
7.6.1 Class sysWin inherits wxMiniFrame implements dbUpdate .....	69
7.7 CREATEDEVICE.PY .....	69
7.7.1 Class CreateDeviceWindow inherits wxDialog .....	69
7.8 CREATEDEVICETYPE.PY.....	71
7.8.1 Class CreateDeviceTypeWindow inherits wxDialog .....	71
7.9 CREATELINK.PY.....	71
7.9.1 Class CreateLinkWindow inherits wxDialog .....	71
7.10 CREATELINKTYPE.PY.....	71
7.10.1 Class CreateLinkTypeWindow inherits wxDialog .....	71
7.11 CREATEPORTS.PY .....	71
7.11.1 Class CreatePortsWindow inherits wxDialog.....	71
7.12 ENHANCEDSTATUSBAR.PY .....	72
7.12.1 Class EnhancedStatusBarItem inherits object .....	72
7.12.2 Class EnhancedStatusBar inherits wxStatusBar.....	72
7.13 HELPER.PY.....	72
7.13.1 Class TextDropTarget inherits wxTextDropTarget .....	72
7.14 LOGWINDOW.PY .....	73
7.14.1 Class LogWindow inherits wxDialog.....	73
7.15 MYEXCEPTIONS.PY .....	73
7.16 VALIDATION.PY.....	73
7.17 OTHER FILES IN THE CDBVIS DIRECTORY.....	73
7.16 HOW TO CUSTOMIZE THE TOOL FOR YOUR USE.....	74
7.16.1 Add a new subsystem.....	74
7.16.2 Do not show uninterested subsystems in the tree view.....	75

---

7.16.3 Adding new zoom levels for a subsystem .....	75
7.17 CVS .....	86

## List of Figures

Figure 1 Here we choose to duplicate 2 devices and the connection between them, the result shows that we created an identical structure of what we duplicated. ....	14
Figure 2 Clone a device (FEE_MUON_5) and its connection to M5R4#4. A new device is created with the same properties as the FEE_MUON_5, but connected to another port on M5R4#4. ....	14
Figure 3 CdbVis program window (MainWindow), and the names of the windows within that one.....	15
Figure 4 CdbVis program window right after it has been connected to the ConfDB. The miniature window at the bottom right is for use in the Device Level (read below). ...	16
Figure 5 The tree view window in the selection window. ....	18
Figure 6 The Link selection in the selection window. In this screenshot it is set to only show links of the link type DATALINK .....	20
Figure 7 The information window with information for a FEE device in the Muon subsystem.....	21
Figure 8 The status bar. The column with the light bulb is column 1. ....	21
Figure 9 The miniature window which can be used when we are in level -1 (Device level)	24
Figure 10 The toolbar .....	25
Figure 11 The Create Device Type dialog, here the Muon subsystem is the active system. ....	38
Figure 12 The Create Device Dialog, here the Muon subsystem is the active subsystem.	39
Figure 13 The Port Creation Dialog, opened from the Device creation dialog, you can see that in this case we are creating ports for a device named M5R4#5.....	41
Figure 14 The create link type dialog, here we are creating a link type with the name HV.	42
Figure 15 The create link dialog, where we set the devices and ports the link will be connected between.....	43
Figure 16 Modify a device type dialog, in this screenshot it is a FEE that is modified. ...	44
Figure 17 The modify device dialog, in this example an FEE board is modified, the disabled areas are attributes that can not be modified. ....	44

---

Figure 18 The Modify Link Type Dialog, in this example it is the data link link type that is modified. ....	45
Figure 19 The modify link/connection dialog, here you can change the connected ports and devices and other attributes. ....	46
Figure 20 The Delete Device type dialog .....	46
Figure 21 The delete link type dialog .....	47
Figure 22 The class model above show the relations between the classes in the different modules (shown as packages), and the member variables that is responsible for the association/reference between the classes are shown. ....	55



## List of Tables



## 1. Preface

The LHCb Configuration Database Visualizer (hereafter known as CdbVis) is a program tool to make it easier for people to work with the information stored in the LHCb Configuration Database (hereafter known as ConfDB) through a graphical user interface (GUI). The ConfDB is a database where all devices in the LHCb detector including the connectivity information between the devices are stored.

Felix Schmidt-Eisenlohr, a former summer student at CERN, started the work on the program. He managed to develop some of the basic functionality for a previous version of the ConfDB, and also the basic functionality of the visual appearance of the connections and devices in the ConfDB.

Thomas Johansen, has continued his work, and finished the most-needed functionality of the program in addition to some specific features for the use of the program in the LHCb Muon group at CERN.

The program is developed in Python using the widget library wxPython, and it also uses the specific LHCb Configuration Database Library (hereafter ConfDB Library) written by Lana Abadie, a Ph.D. student at CERN.

**IMPORTANT NOTE:** in this document, “devices” are functional devices. For consistency, the device name and device type should match the names they were given in PVSS and vice-versa.

## 2. Installation

### 2.1 Program requirements

The program was developed and tested using the following versions of external software/libraries:

#### 2.1.1 Linux/Unix (LXPlus/AFS)

##### Test environment

Python 2.3.5  
wxPython 2.6.3.0 ANSI version  
ConfDB Library 2.11  
Scientific Linux 3.0

The combination of Python version below 2.3 and wxPython version below 2.4 (unknown reason), causes memory leaks when running the program, and will eventually fail with a segmentation fault. To be sure that the program will run smoothly, install the Python 2.3.5 and wxPython 2.6.3.0 or newer.

#### 2.1.2 Windows (NICE)

##### Test environment

Python 2.3.5  
wxPython 2.6.3.0 ANSI version  
ConfDB Library version >2.11  
Windows XP SP2 (Windows 2000 and above is recommended)

Python 2.3.5 is used due to compatibility issues with 2.4.2 and the ConfDB Library Python wrapper. There are some issues on the visual appearance and repainting of the program in Windows, but they are negligible and do not root in any functional issues.

If you run < Windows 2000 you will have to install additional libraries, explained in the installation section.

#### 2.1.3 Requirements

The program should in theory run smoothly with:

Python Python 2.3.X and above  
wxPython 2.6.3.0 and above  
ConfDB library version > 2.11 (and later minor bug revisions)  
Linux/Unix distributions or Windows 95 or above  
(AFS Client so that the Boost library can be found on the CERN network.)

## 2.2 Installing on Linux (LXPlus/AFS)

### 2.2.1 Installing Python

You can find the recommended stable versions of Python here: <http://www.python.org/download/>. Remember that there are some restrictions on which version of Python that have to be installed for the different versions of wxPython. Also note that on most Linux distributions Python is preinstalled. To check the version of Python you are running, run the following command in a terminal window:

**python -V**

The version number of the already installed Python will be echoed to the screen. If you are running a version  $\geq 2.2.3$  you will most likely not need to install an upgrade or a new version of Python. Note that an installation of a newer Python version will most likely install the new Python version in a different place than your previous Python version, and also a new version of the python executable. It is possible to run two or more versions of Python in parallel, but not recommended unless you know what you are doing.

If you have to install a new version of Python, you can choose between different installation options depending on your system and expertise; packages, or compile from source which is needed in some cases.

To check if Python was successfully installed, run `python -V` again to see that your python executable echo the new version number.

### 2.2.2 Installing wxPython

#### 2.2.2.1 Python $\geq 2.3$

Once you have installed Python, and made it work it's time to install wxPython. If you have Python 2.3.X or 2.4.X installed, you can go directly to <http://wxpython.org/download.php> and download the wxPython libraries. Note that GLIB and GTK+ must be installed on your system, but this is also usually preinstalled on the different Linux distributions.

Scroll down to the Linux RPM section, read what that is written there. Choose only one of the wxPython Common packages, the one corresponding to your system, usually the ANSI version should cover your needs, and since the UNICODE version sometimes doesn't work with all systems, I recommend the ANSI version. In the wxPython Runtime section, you can choose several, but I recommend you to choose one, of the same type as your common, ANSI or UNICODE, depending on your choice. If you're running Debian you can get wxPython running the apt-get command as shown on the web page.

If none of the RPM packages listed are compatible with your system, you can build an RPM from the source as shown on the website. You are also recommended to download documentation and wxPython demos. If you do not manage to get wxPython up and running, I recommend to download a previous version of the wxPython library from SourceForge as described in 2.2.2.2, and possibly downgrade to Python 2.3 or 2.2.

#### 2.2.2.2 Python $< 2.3.X$

In this case you will have to download an older version of wxPython available from the SourceForge website: [https://sourceforge.net/project/showfiles.php?group\\_id=10718&package\\_id=10559](https://sourceforge.net/project/showfiles.php?group_id=10718&package_id=10559). Version

2.4.2.4 is recommended since it is tested that CdbVis is compatible with that version. If you choose 2.4.2.4 you download the wxPythonGTK-py2.2-2.4.2.4-1.i386.rpm, which is for Python 2.2. However the program does not officially support Python version lower than 2.3.

### 2.2.2.3 Known problems

When you have installed wxPython you will in most cases have to set up some environment variables yourself, or at least direct Python to where it can find the wxPython library/modules.

Since Python is usually installed in `/usr/local/lib/pythonX.X`, it is expected that new packages/libraries is installed in the subfolder named site-packages. But this is usually not the case with wxPython as the installation is usually directed to `/usr/lib/pythonX.X/site-packages/wx-<packagename>`.

So the system path is then directed to look for wxPython and other libraries in the subfolder of the Python installation, but can not find the modules there. There are 3 solutions to solve this problem:

Make a soft link from the subfolder of the Python installation to the place wxpython is. Go to `/usr/local/lib/pythonX.X/site-packages` and run the command:

```
ln -s /usr/lib/pythonX.X/site-packages/wx-<packagename> wx-<packagename>
```

There is a space between the first path and the name of the new link, and you substitute the wx-<packagename> with the name of the wxPython directory in your case.

Move the wxPython library to the `/usr/local/lib/pythonX.X/site-packages`

Change the PYTHONPATH and LD\_LIBRARYPATH to look for packages in the site-packages directory in `/usr/lib/pythonX.X/site-packages`, like this:

```
set PYTHONPATH=${PYTHONPATH}:/usr/lib/pythonX.X/site-packages
```

```
set LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/lib/pythonX.X/site-packages
```

You have to change pythonX.X to the name of the Python directory in your case.

For some reason, to change the system path (choice 3) seems sometimes to be ignored by Python. I recommend you to go for the first solution.

## 2.2.3 Installing CdbVis

This will be downloadable from the web (or CERN network) in a tar.gz (tgz) or zip file to be extracted to an arbitrary directory on the user's hard drive. **(To be added later)**. To extract the contents of the file to a location of your choice, run this command:

```
tar -xvzf <name_of_the_targz_file>.tar.gz <Directory_to_extract_to>
```

or

```
unzip <name_of_the_zip_file>.zip <Directory_to_extract_to>
```

## 2.2.4 Installing Oracle Instant Client 10g

This library is needed for the Oracle database connection used by the ConfDB Library. Installation instructions can be found here: <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/APIusage.htm>

## 2.2.5 Installing ConfDB library

This is the library used by CdbVis to communicate with the ConfDB. Download the latest Python Interface library for Linux from: <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/APIusage.htm>. All the files downloaded in the zip file should be added to the installation directory of CdbVis, and the path to this directory should be added to the LD\_LIBRARY\_PATH variable.

## 2.2.6 Testing Installation

If CdbVis is “not working” after you have installed all of its dependencies, here are some tests you should do to see what that fails. To make it easier to debug possible errors, we take one step at a time.

### 2.2.6.1 Test Python

Run the following command:

```
Python
```

You will enter the Python Shell, type:

```
print "hello world"
```

and press enter. If it writes hello world back to you, then Python works. Press Ctrl-D to exit the shell.

### 2.2.6.2 Test wxPython

Download the demo package (if you have not downloaded it yet) from <http://wxpython.org/download.php> if you downloaded the latest release of wxPython, or from the SourceForce place corresponding to your version of wxPython. Extract the tar.gz archive to an arbitrary place on your hard drive

```
tar -xvzf file.tar.gz destination_directory
```

go down the hierarchy :

```
cd <the_folder_you_extracted_the_demo_package_in>/wxPython-2.4.2.4/demo
```

and run the command:

```
python wxFrame.py
```

(if wxFrame.py does not exist, try executing one of the other \*.py files in the same directory), and a GUI-window should appear. If it does not, and a Python error is returned instead, the wxPython is not correctly installed (probably Python can not find the wxPython modules because the path to the wxPython directory is either not set or wrong).

### 2.2.6.3 Test CdbVis

Go to the directory where you installed CdbVis and run the command:

```
python main.py
```

If it runs, you have done most things correct so far, the ConfDB library files have been found as well. If it fails, it is probably because the ConfDB library files were not found, see through the installation of the ConfDB library as described above to see if you missed something.

### 2.2.6.4 Test ConfDB Connection

Once you have started CdbVis, click on the button which shows a lightning, or choose from the menu: File > Connect, to connect to the ConfDB. If it fails, the error will probably be reported either through CdbVis, or in the terminal window. Errors here should be directed to Lana Abadie, who can most likely tell you why the database connection failed.

## 2.3 Installing on Windows (NICE)

### 2.3.1 Installing Python

You can find the recommended stable versions of Python here: <http://www.python.org/download/>. Remember that there are some restrictions on which Python version that have to be installed for the different versions of wxPython. To check the version of Python you are running, run the following command on the command line:

```
python -V
```

If you get command not found, Python is not installed on your system. For Windows you will need Python version 2.3.X, due to compatibility issues with ConfDBLibrary and Python 2.4.X and 2.2.X (but it may work on 2.2.X if you are lucky).

It is possible to run two or several versions of Python in parallel, but not recommended unless you know what you're doing. You then download the Python executable from the website, and execute the \*.msi or \*.exe file to install it. Choose a directory to install it in, and remember this directory. You will have to add the path to the directory to the Path environment variable. This is done by right-click My Computer on your



desktop, choose properties, choose the advanced tab and click on the button that says "Environment Variables". In the system variable window, you look for the Path variable, choose to edit, and add the path to the directory where you installed Python at the back of the string. If the string that was already set for the path did not end with a ';' (semicolon), you will have to add that to the string before you add (append) your new path. Click Ok all the way back, to close the windows. You will have to close all open command line windows to make the change take effect (but in some cases you will have to log off and on again).

To check if Python was successfully installed, open a new command line window, and run `python -V` again to see that your python executable echoes the new version number. If the version number is incorrect it is most likely because you have another version of Python installed on your system that is executed instead. To deal with this; change the name of one of the Python executables (found in the Python root directory), and execute the command:

```
<name_of_my_python_executable> -V
```

To see that it is the correct version. Now you will have to use this name instead of Python when you are running python scripts.

## 2.3.2 Installing wxPython

### 2.3.2.1 Python == 2.3.X (only supported)

Once you have installed Python, and made it work it is time to install wxPython. If you have Python 2.3.X, you can go directly to <http://wxpython.org/download.php> and download the wxPython libraries. Note that if you're running < Windows 2000 you will probably have to update some libraries, read the documentation on the website and follow the instructions if this is the case.

You will have to download an ANSI version of wxPython, if you choose a UNICODE version; the database connection will not work (because strings are converted to UNICODE strings which the ConfDBLibrary does not accept).

You are also recommended to download the documentation and wxPython demos from the same webpage.

## 2.3.3 Installing CdbVis

This will be downloadable from the web in a zip file to be extracted to an arbitrary directory on the user's hard drive.

## 2.3.4 Installing Oracle Instant Client 10g

This library is needed for the Oracle database connection used by ConfDB library. Installation instructions for Windows can be found here: <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/APIusage.htm>

## 2.3.5 Installing ConfDB library

This is the library used by CdbVis to communicate with the ConfDB. Download the Python Interface library for Windows from: <http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/APIusage.htm>. The two \*.dll and the two \*.lib files should be extracted to the installation directory of CdbVis.

## 2.3.6 Testing Installation

If CdbVis is “not working” after you have installed all of its dependencies, here are some tests you should do to see what that fails. To make it easier to debug possible errors, we take one step at a time.

### 2.3.6.1 Test Python

Run the following command:

```
Python
```

You'll enter Python Shell, type:

```
print "hello world"
```

and press enter. If it echoes “hello world” back to you, then it works. Press Ctrl-Z to exit the shell.

### 2.3.6.2 Test wxPython

Download the demo package from <http://wxpython.org/download.php> and install it on your hard drive. Open a command line shell and go to <the\_folder\_you\_installed\_the\_demo\_package\_in>/wxPython-<wxPythonVersion>/demo, and run the command:

```
python wxFrame.py
```

(if wxFrame.py does not exist, try executing one of the other \*.py files in the same directory) and a GUI-window should appear. If it doesn't and an error is returned instead, the wxPython is not correctly installed.

### 2.3.6.3 Test CdbVis

Go to the directory where you installed CdbVis and run the command:

```
python main.py
```

If it runs, you have done things correct so far, the ConfDB library files have been found as well. If it fails, it is probably because the ConfDB library files were not found, see through the installation of the ConfDB library as described above to see if you missed something.

If \*.py files is associated with Python you can actually just double-click on main.py and it will start. (If the \*.py files have a icon of a python snake).

### 2.3.6.4 Test ConfDB Connection

Once you have started CdbVis, click on the button which shows a lightning, or choose from the menu: File > Connect, to connect to the ConfDB. If it fails, the error will probably be reported either through CdbVis, or in

the command line shell window. Errors here should be directed to Lana Abadie, who can most likely tell you why the database connection failed, and help you out.

## 3. Program features

The program is featuring two different approaches to the ConfDB:

Navigation mode: Get information about devices and connectivity between them, drawn in the program window using visual objects (representations of the devices in the ConfDB) on the screen, here called navigation.

Creation mode: Create/Modify/Delete information about devices and connectivity between them, both with visual objects and loading/parsing data files with information. This is the **default** mode.

### 3.1 Navigation

As described above, navigation is to get information from the ConfDB. Getting the results (devices and their connections) drawn as visual objects on the screen makes it easier to see and understand the connectivity between devices and subsystems in the real LHCb detector. There are several reasons for why one would like to navigate in this system:

To see the data flow of one kind of data (gas, voltage, data, etc.) from one end (inner part of the detector) to the other (outer part of the detector).

To read the different parameters that has been set for a given device, for a given run, or in general.

To track down errors to the source by following the data flow from where the error was found (seen).

More?

#### 3.1.1 Abstract Use-cases

Abstract level of description of what features the program provides.

##### 3.1.1.1 Connect to the database

The user connects to the ConfDB to get a persistent connection to the database which will last for the whole session unless the program terminates unexpectedly or the ConfDB becomes inaccessible.

##### 3.1.1.2 Disconnect from the database

The user disconnects from the database once he/she has finished her session, either indirectly by closing the application window, or directly by a specific disconnect command.

##### 3.1.1.3 Hierarchical navigation

The user can navigate in the LHCb detector in a hierarchical way, which means that he/she can go down the hierarchy in this order: LHCb detector, subsystem, device types (or link types), devices, ports.

##### 3.1.1.4 Subsystem dependent navigation

The user can navigate in the LHCb detector in a subsystem dependent way (different for each subsystem), to go down the different levels in each subsystem, f. ex. in the Muon subsystem: LHCb detector, muon station, station quadrant, muon chamber. There will a possibility to go both ways between the levels.

### 3.1.1.5 Object parameter information

One can view the parameter information of each object (device type, link type, device, link, port) stored in the ConfDB, by selecting it.

### 3.1.1.6 View only given link type

The user can choose (or choose not) to only view connections with a given link type.

### 3.1.1.7 Zoom

The user can zoom objects on the screen to make them bigger (zoom > 100%) or smaller (zoom < 100%).

### 3.1.1.8 Different view modes

The user can choose from the different views: path view, neighbor view, dynamic link view and subsystem view.

#### 3.1.1.8.1 Path view

If the user selects a device in this view, he/she will see one of the paths drawn on the screen as the device is a part of. The user can also choose to select and view the other paths this device is a part of.

#### 3.1.1.8.2 Neighbor view

In neighbor view the user can view the selected device and the neighbors which are directly connected to it.

#### 3.1.1.8.3 Dynamic link view

Dynamic link view works like neighbor view the first time you select a device after changed to this view, but you can also see the neighbors of the direct neighbors if you select the direct neighbors of the first device.

#### 3.1.1.8.4 N-hops device view

In this view the user can select a device and then choose from a pop-up window how many hops (neighbors of neighbors a.s.o.) to view out from the selected device.

### 3.1.1.9 Miniature view

Since the area of the screen where objects can be drawn on is bigger than the visible for the user, there will be a possibility for the user to see where all of the objects are placed on both the shown and hidden part of the window (which you will have to scroll to see), and easily move the visible part of the window to where the objects are.

### 3.1.1.10 Error logging

Log all errors (and informative messages) sent to the user in a text file, which is also available from inside the program.

## 3.2 Creation

As described above, navigation is to read information in the ConfDB, and in Creation Mode all the features for the Navigation mode are available in addition to some Creation Mode specific features. The user is able to create/modify/delete information to/in the ConfDB. The reasons for the creation mode are:

To easily do small changes or updates to single objects, in an intuitive way.

To create many objects (devices and connections between them) with similar structures in a fast and efficient way.

To easily replace a device with a new one, if it is broken.

To have the possibility to undo (and redo) creation/modification/deletion before it is stored in the ConfDB.

Generate and save SQL code for the all actions done on objects in CdbVis in a text file, to later make your own customized scripts against the ConfDB.

Read data information from file to create visual objects of it in CdbVis, and later store it in ConfDB.

### 3.2.1 Abstract Use-cases

#### 3.2.1.1 Same features as in navigation mode with the following limitation

The path view will work, but devices created in the given session and not yet stored in the database will not be viewed in path view.

#### 3.2.1.2 Create device type

The user can create a device type, which are common properties for a collection of devices of the given type.

#### 3.2.1.3 Create link type

The user can create a single link type or a composite link type, which are common properties for a collection of links of the given type. The composite link type is a link type consisting of several simple link types (can transfer different kinds of data).

#### 3.2.1.4 Create device and port(s)

The user can create a device of a given device type, and set the different properties for the new device. Every device also has several port objects related to each port of the device, and the ports have to be created when the device is created.

#### 3.2.1.5 Create multiple devices and ports

The user can create multiple similar devices of the same device type with the same port information for every device.

#### 3.2.1.6 Create connection (link)/Connect devices

A free link is a link that is connected to zero or none devices, cannot be created in CdbVis. A link can only be created when setting up the connection between two devices.

#### 3.2.1.7 Create a spare device and a spare port

Part of the equipment will be spare and use to replace broken modules. A spare device is identified by a serial number. It is located somewhere. It has an hw type which should be prefixed by the subsystem, if this latter is specific to one. For instance a spare for a hybrid can only be used by the VELO subsystem whereas a spare for a TELL 1 board can be used by any subsystem.

The port of a spare device should be created only if it has an MAC address and/or a bia (so only for DAQ devices). Otherwise don't do it.

### **3.2.1.8 Modify device type**

To modify is to change one or more properties of a device type.

### **3.2.1.9 Modify device (and port(s))**

To change one or more properties of a device, or one or more of the ports that is associated to that device. Ports can be created, modified and deleted for a given device.

### **3.2.1.10 Modify link type**

To change one or more properties of a link type, or to change from a single to a composite link type or vice versa.

### **3.2.1.11 Modify connection (link)**

To change one or more properties of a link, or to change the connectivity (which devices and/or ports it is connected to).

### **3.2.1.12 Delete device type**

It is possible to delete a device type that is stored or not in ConfDB.

### **3.2.1.13 Delete device**

It is possible to delete a device that is stored or not in ConfDB yet.

### **3.2.1.14 Delete connection (link)**

A connection (link) between two devices can be deleted by the user at any time.

### **3.2.1.15 Delete link type**

It is possible to delete a link type that is not yet stored in ConfDB yet. If it is stored in the ConfDB, it is not possible to delete it using CdbVis.

### **3.2.1.16 Duplicate structure (device(s), port(s), connection(s))**

The user can duplicate an already existing structure (one or several visual devices and connections between them), by selecting them, and choose to duplicate. The structure will be duplicated (all the devices (and ports) and links) to one or several identical structures. This way it is fast to duplicate often used structures. If a link is chosen, both devices it is connected to must also be selected, because links that is not connected to a device in each end is not allowed to exist.

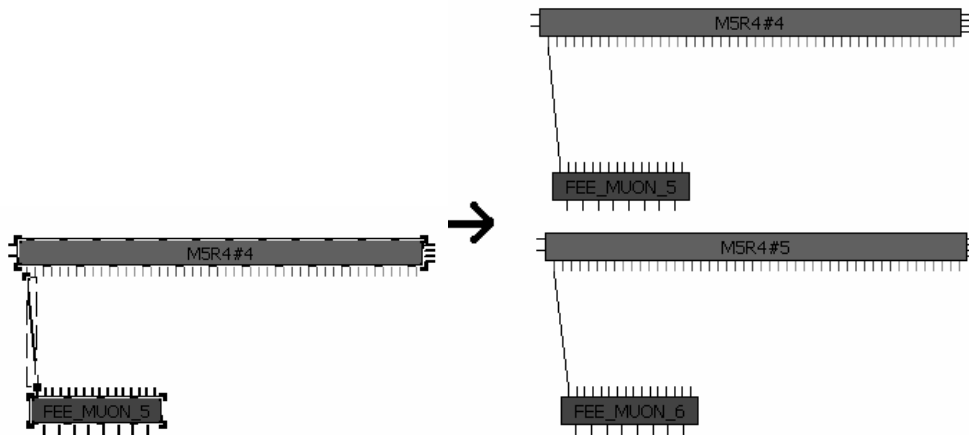


Figure 1 Here we choose to duplicate 2 devices and the connection between them, the result shows that we created an identical structure of what we duplicated.

### 3.2.1.17 Clone structure (device(s), port(s), connection(s))

Clone is similar to duplicate, except you can select a link without selecting one or both of the devices it is connected to. The devices that are connected to one or more of the links that are to be cloned, but are not to be cloned themselves, will have the new links attached to them the same way as the original link.

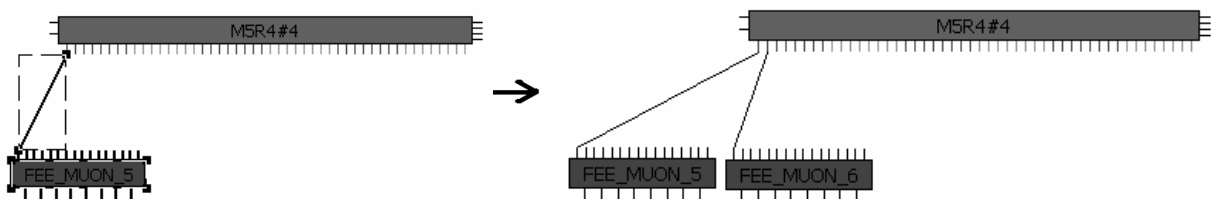


Figure 2 Clone a device (FEE\_MUON\_5) and its connection to M5R4#4. A new device is created with the same properties as the FEE\_MUON\_5, but connected to another port on M5R4#4.

### 3.2.1.18 Recovery feature

If the program terminates unexpectedly or the database connection fails, all objects are stored in a text file in a human readable format, and are imported the next time the user runs the program and changes to creation mode, if the user wants to. The user can then continue where he/she was interrupted last time. This is also the case if the user terminates the program without saving the changes to the database.

### 3.2.1.19 Mass insertion feature

The user can load a text file or xml file with a predefined format and syntax for the data in the file. The data in the file is device type, link type, device, link or port, a spare device, a spare port and the corresponding object will be created in CdbVis after successful loading of the data. After that, the user can store the data in the ConfDB.

### 3.2.1.20 Export changes to ConfDBLibrary Python Script file



The user is able to export the changes that he/she did in CdbVis to a python script file. The python script file can then later be modified manually by the user, and executed to commit the data in the ConfDB.

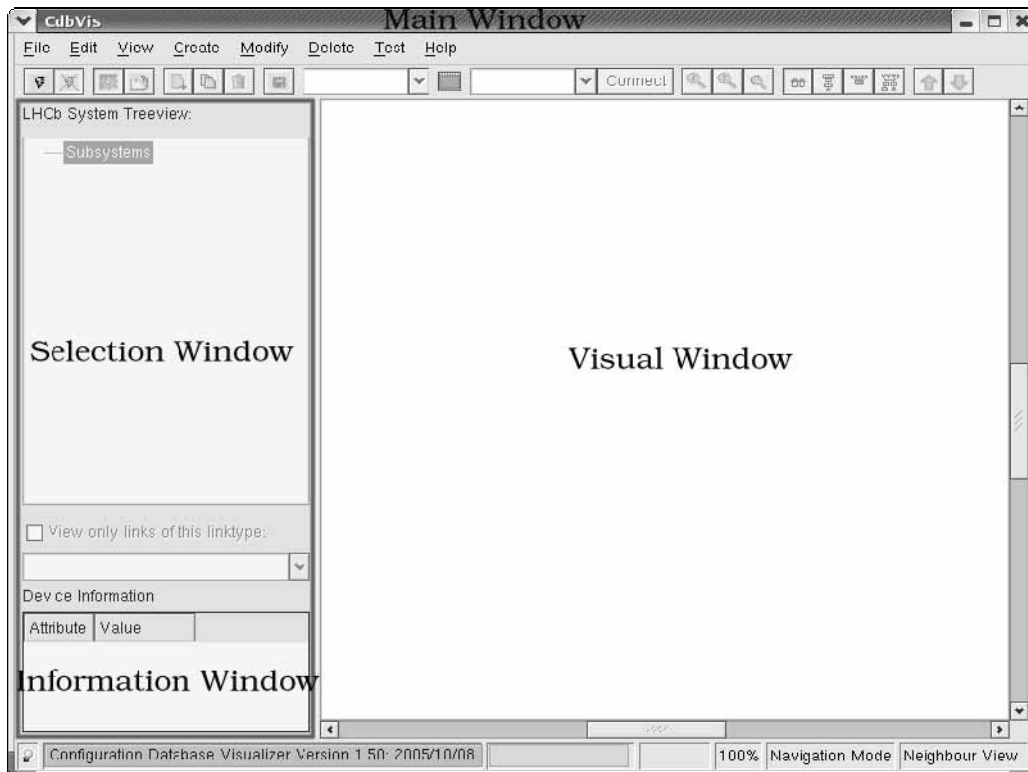


Figure 3 CdbVis program window (MainWindow), and the names of the windows within that one.

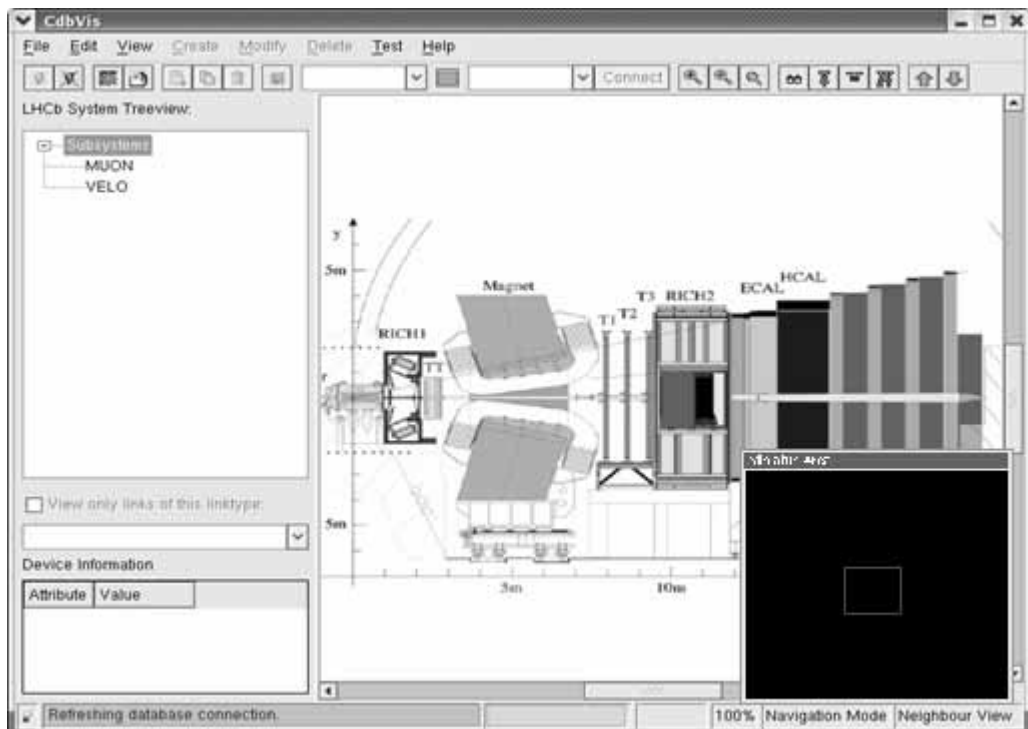


Figure 4 CdbVis program window right after it has been connected to the ConfDB. The miniature window at the bottom right is for use in the Device Level (read below).


## 4. Program functions - How to use the program.

### 4.1 Startup and shutdown


Open a new terminal/command-line window, go to the directory of the installation of CdbVis, and run the command: `python main.py` Or, you can simply double-click on `main.py` if `*.py` files are associated with the python interpreter/executable.

It may take some time before the program starts, but if you do not get any errors output to the terminal window, you will just have to wait.

Once the program has started, you will see a “static” window frame, without any active content, because the program is not yet connected to the ConfDB.

Whenever you want to connect to the ConfDB and make your program “active”, at least some options clickable, choose File > Connect from the menu, or click on the button with the  lightning: But if you may prefer to click Ctrl-C to connect to the database in Windows, because the window is not fully repainted on start-up, and needs to be moved to get a full repaint, and then be able to click on the connect button.

There is no timeout on the database connection, so it will try to connect to the ConfDB until it responds. This may sometimes take a while. If it continues to try to connect to the ConfDB for several minutes, you may have to terminate the program and try again (later).

Once the connection to the ConfDB is established several icons/buttons will become active (clickable), and you will also see that the tree view is filled with names of the different subsystems/sub detectors available in the ConfDB and a 2D illustration of the LHCb detector appears in the visual window. The program is set to run in the default mode, which is creation mode, which you will see because the hand icon is depressed: To change to navigation mode, you will have to click on the navigator helm icon next to  the hand helm icon.

You can change between navigation mode and creation mode whenever you want to, but if you have created something in creation mode, and not stored it in the database yet, you will either have to store it in the ConfDB or choose not to save when you are asked when changing to navigation mode.

When you have finished your work, you can click on the button which shows a lightning with a red cross over it to disconnect from the ConfDB, or just exit the program without disconnecting. The database connection will be gracefully disconnected in both cases.

### 4.2 Other user mode independent features

All errors and information that are reported to the user during a session is stored in a file called `CdbVis.log` in the same directory as the program. All errors and information for a session can be viewed by double-

clicking on the first column in the status bar (where the icon showing the type of message) or by choosing View > Log from the menu bar. The error log can be deleted after program termination without affecting anything.

If the program terminates, and the user has not saved in ConfDB what he/she created in the given session, all the objects created in the given session is saved in a file named "recover.dat" in a human-readable format. The objects in this file (which was created/modified/deleted by the user in the previous session) will be loaded into the memory after changing to creation mode, if the user wants to. The user can also choose to ignore this file, and clear the contents. The file can also be cleared manually in a text-editor etc without affecting anything.

## 4.3 Navigation mode

### 4.3.1 Tree view/hierarchical navigation

#### 4.3.1.1 Subsystems node and its child nodes

In the selection window to the left in the program window, you see the tree view control (shown to the right) that shows the expanded "Subsystems" node with the name of the different subsystems as child nodes (Muon, Velo, TFC, PS etc.), after you have connected to the ConfDB. You can choose only one of the subsystems at a time, and you will see that the name of the active subsystem (the one you choose) will appear in the fourth column in the status bar (at the bottom of the application window). If you choose a subsystem name in the tree view with a single-click, you will only set the system to the active system, and perhaps see a change in the visual window to the right, depending on the subsystem you chose. If you double-click on a subsystem name in the tree view however, you will in addition to what you do with the single-click, also expand the node you clicked, and see two new child nodes appear (in Windows you will probably have to click on the plus sign next to the node to expand it after a double-click), named Device types and Link types respectively.

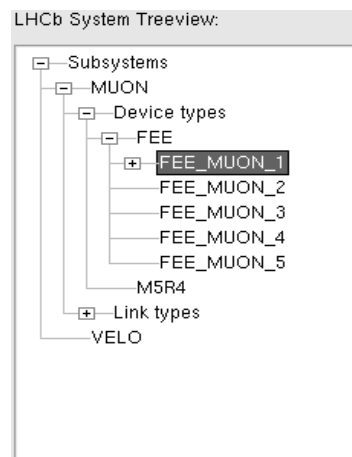


Figure 5 The tree view window in the selection window.

If you change subsystem by single -or double-clicking on a subsystem node, when another subsystem is already expanded, you will collapse the whole previously active subsystem, and immediately change the active subsystem to the new subsystem you chose.

You can expand a node, by either double-clicking it or clicking on the plus-sign to the left of it, if it is visible.

#### 4.3.1.2 Device types node

Single-clicking on this node will make the visual window cleared with a white color, because you have now definitely chosen to find the object you are looking for using the hierarchical view, and hence the subsystem specific view in the visual window is not necessary to display any more. If you double-click on the "Device types" node, it will be expanded, and you will see all device types created for/in the given subsystem as child nodes of the "Device types" node. If there are no child nodes, it means that no device types have been created for the given subsystem yet.

#### 4.3.1.3 Device type node(s)

If you just choose a device type by single-clicking, you will get all information stored in the database about the given device type in the information window (just beneath the selection window). If you double-click on it, it will expand and you will see all the devices created in the given subsystem of the given device type as child nodes of the given device type node. If no devices have been created for the given device type yet, the given device type node will not be expanded.

#### 4.3.1.4 Device node(s)

A single-click on a given device node will give you all information about it in the ConfDB shown in the information window, and the device will also be shown as selected in the visual window (the appearance of the device in the visual window depends on the view you have chosen). If you double-click on a device node, it will expand and you'll see all the names of the ports for the given device with the following node name syntax: <Device name> : <Port number> : <Port type> : <Port direction>. If any of the values is not set, it will just be a white space set instead, for the given value.

#### 4.3.1.5 Port node(s)

A single-click on a given port node will provide you with the information that is stored about it in the ConfDB in the information window. This is the final/lowest level in the hierarchy, so double-click will have the same effect as single-click.

#### 4.3.1.6 Link types node

Same affect as with device types node in 4.3.1.2, but here the "Link types" node will be expanded and you will see the link types in the subsystem, when double clicking on the "Link types" node.

#### 4.3.1.7 Link type node(s)

A single-click on a link type node will provide you with information about the link type in the information window. A double-click is the same as a single-click because this is the final/lowest level in the hierarchy.

#### 4.3.1.8 Spare hierarchy

In the subsystem hierarchy, there is "Spare devices". It is the node which permits to view the spare equipment. By double-clicking on Spare devices, we get the list of the spare hardware types. By double-clicking on spare hardware type, we get all the spare hardware of the selected type. By double-clicking on a spare device, we get all the ports of this spare (useful only for DAQ devices to get the MAC address and the bia).

The name of the spare hardware type indicates the subsystem it is part of as it is prefixed with by <subsystem\_name> and “\_”. It is the case for the Muon chambers. If it is common to all the subsystems, then it is not prefixed with any subsystem. It is the case for the TELL1 boards

### 4.3.2 Only view links/connections of a given link type and filter by spare parts/in place components

#### 4.3.2.1 Only view links/connections of a given link type

Beneath the tree view window, there is a text saying: “View only links of this link type:” to the right of a checkbox.

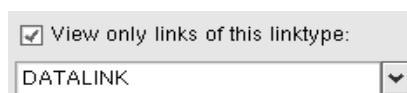


Figure 6 The Link selection in the selection window. In this screenshot it is set to only show links of the link type DATALINK

If the checkbox is *unchecked* you will always see all connections of a device in the visual window, meaning that the name of the link type in the combo box has no effect on the view. If the checkbox is *checked*, you will only see those connections of a device that are of the link type selected in the combo box below.

The effect of the change of the link type to view, and check/uncheck of the checkbox, will be set immediately. But if you choose to view links of a link type, and none of the links in the visual window are of the given link type, you will get a message telling you that the visual window would become empty if you choose what you do, and it is set back to the previous setting.

If you check the link type to view, and there are one or more links of that type in the visual window, all the other links (of other link types), and devices that are no longer attached to any other devices once the links of other link types have been removed, will also be removed from the visual window. You will end up with viewing only devices with connections with links of the given link type the view is restricted to. The usage of this link type is to set the link type at the beginning of a device choice.

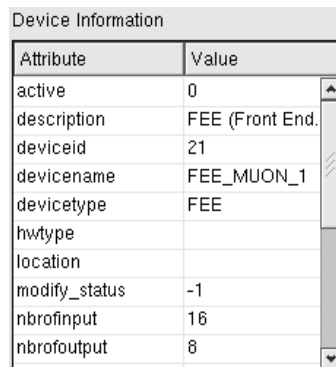
#### 4.3.2.2 Only view spare parts/parts in place

Beneath the tree view window, there is a drop-down box which has the title ‘Filter by spare parts’. By choosing a value in this drop-down box you can display only spare devices, only devices in place, or all devices. It defaults to displaying all devices.

### 4.3.3 Information window

As mentioned above (under 4.3.1), selecting an object that exists in the database (device type, link type, device, port, link) either in the tree view window or in the visual window, will display the information stored in

the ConfDB about the object in the information window. Note that not all information, and in some cases more information is shown than what, is stored in the ConfDB.



Attribute	Value
active	0
description	FEE (Front End.
deviceid	21
devicename	FEE_MUON_1
devicetype	FEE
hwtype	
location	
modify_status	-1
nbrofinput	16
nbrofoutput	8

Figure 7 The information window with information for a FEE device in the Muon subsystem

### 4.3.4 The Status bar



Figure 8 The status bar. The column with the light bulb is column 1.

The status bar at the bottom of the program window has seven (7) columns, each of them showing informative information about some settings etc. in the program.

#### 4.3.4.1 Message icon and text (column 1 and 2)

In the first column, the most-left column, there is an icon, telling you the severity of an error, or whether it is an error or just an informative message that is displayed in the second column. In the second column the text appears in blue if it is an informative message to the user (not an error), or in red if it is an error message to the user. If you double-click on the first column you will see a window pops-up with all messages that have been shown to you in the existing session. If you do not see the whole message in the second column, you can double-click on it and the whole message will be shown in a message box.

#### 4.3.4.2 Progress bar (column 3)

In the third column, there is a progress bar, and this one will show the status (how much is done, and how much is left of the process to finish) of a process of most lengthy processes in the program. This way it is easier to see if the program is doing something meaningful, or just idling. The cursor will also change to a busy cursor (f. ex. hourglass) while the process is in progress.

#### 4.3.4.3 Active subsystem (column 4)

The fourth column is showing the active subsystem, or is blank if there are no active subsystems. The active subsystem means that everything you do is restricted to the active subsystem. You can only have one active subsystem at a time.

#### 4.3.4.4 The Zoom Factor (column 5)

The zoom factor tells you the zoom that is used on the visual objects in the visual window. If the zoom is less than 100%, it means that the visual objects on the screen appear smaller than the original (default) size (100%), and if it is bigger than 100%, the visual objects appear bigger than the default size.

#### 4.3.4.5 User mode (column 6)

Lets you know whether you are in navigation mode (default) or in creation mode.

#### 4.3.4.6 View (column 7)

Tells you the current view chose for the visual objects in the visual window. The possible views are: Neighbor view, dynamic link view, path view and subsystem view.

### 4.3.5 The menu bar

#### 4.3.5.1 File menu

##### 4.3.5.1.1 Load settings...

CdbVis has a configuration file that is used to set different settings for the program. This is normally named settings.cfg (default), but if this file does not exist or the user wants to load a configuration file with different settings the user can choose an arbitrary file with a .cfg extension to use instead (as long as it is a syntactically correct configuration file for CdbVis).

##### 4.3.5.1.2 Connect

Creates a persistent connection to the ConfDB for the whole session (until the program terminates), or until the user chooses to disconnect. Access key: Ctrl-C

##### 4.3.5.1.3 Disconnect

Disconnect from the ConfDB (terminate the persistent connection). Access key: Ctrl-D

##### 4.3.5.1.4 Export as text

Export the contents of the selected sub-sytem into a text file (user can choose a file-name in pop-up window). This file contains device name, serial number, and physical location, and is intended to be used for a 'reality check' – i.e. ensure that the data in the database matches up with the physical devices.

##### 4.3.5.1.5 Exit

Terminate the program. Access key: Ctrl-Q

#### 4.3.5.2 Edit menu

See Creation mode.

#### 4.3.5.3 View menu

Only one of the views 4.3.5.3.1 -> 4.3.5.3.4 can be selected at a time.

##### 4.3.5.3.1 Neighbors



A view when only the selected device and its direct neighbors and the connections between them are shown (of chosen link type). Access key: Ctrl-N

#### 4.3.5.3.2 Paths

A view where you get a list of all paths a selected device is a part of. This list appears as a box that stays on top of the application window, and you choose paths from the list in this window. Access key: Ctrl-P

#### 4.3.5.3.3 Dynamic Link view

After the first device selection it appears to be the same as neighbor view, but when you double-click on one of its neighbors or another node, this node's neighbors is added to the visual window as well (including to the node itself if it was not already in the visual window). If you double-click on a device that is already in the visual window, and the neighbors of this device are visible, this device will be collapsed. That means that the direct neighbors of this device will be hidden. Thus; first double-click expands a device's connections, the second double-click collapses a device's connections. Access key: Ctrl-Y

#### 4.3.5.3.4 N-hops device view

In this view the user can select a device in the tree view and then choose from a pop-up window how many hops (neighbors of neighbors a.s.o.) to view out from the selected device. Access key: Ctrl-U.

Example:

Nr of hops	What is shown?
0	Only the selected device itself
1	The selected device itself and its directly connected neighbors (as in neighbor view)
2	The selected device itself, its directly connected neighbors, and the directly connected neighbors of its neighbors
3	...

#### 4.3.5.3.5 View miniature window

Opens or closes (depending on whether it is open or not) a miniature window, and places it somewhere near the bottom right corner of the main window. In this window you can see where the visual objects are placed within the visual window. The visual window includes also the area which you have to scroll to see, and by using the miniature window you can easily see where the objects are within the visual window, and move the red rectangle (while clicking inside it and holding down the left mouse button) to the area where the objects are. Access key: Ctrl-M

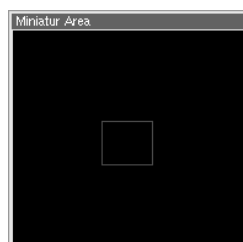


Figure 9 The miniature window which can be used when we are in level -1 (Device level)

#### 4.3.5.3.6 View Program Log

A window pops-up with all messages that have been shown to you in the existing session, and where the user can filter what messages he/she would like to have a closer look at.

#### 4.3.5.3.7 Autozoom enabled

This is a check-item on the menu. If it is checked, auto-zoom is on, if it is unchecked, auto-zoom is off. To understand how autozoom works, read 4.3.6.5.

#### 4.3.5.4 Create menu

See Creation mode

#### 4.3.5.5 Modify menu

See Creation mode

#### 4.3.5.6 Delete menu

See Creation mode

#### 4.3.5.7 Visual menu

##### 4.3.5.7.1 Zoom

###### 4.3.5.7.1.1 Zoom In

See 4.3.6.5.

###### 4.3.5.7.1.2 Zoom Out

See 4.3.6.6.

###### 4.3.5.7.1.3 Custom Zoom...

See 4.3.6.7.

##### 4.3.5.7.2 Selection

###### 4.3.5.7.2.1 Select All

Selects all visual objects in the visual window. Access key: Alt-A.

###### 4.3.5.7.2.2 Deselect All

Deselects all visual objects in the visual window. Access key: Alt-D.

#### 4.3.5.8 Test menu

Only for testing purpose, here you can add any code you want to test a function etc. At the moment it echoes the content of some object lists in the program (only for debugging).

### 4.3.5.9 Help menu

#### 4.3.5.9.1 About

Program credits.

### 4.3.6 The tool bar



Figure 10 The toolbar

Going from left to right; the name of the icon is given by a description of the bitmap you see on the icon.

#### 4.3.6.1 The lightning

Connect to ConfDB, see 4.3.5.1.2 Connect.

#### 4.3.6.2 The lightning with a red cross

Disconnect from ConfDB, see 4.3.5.1.3 Disconnect.

#### 4.3.6.3 Helm navigator

Click on this icon button to change to navigation mode. If you have made changes in creation mode, you will be prompted to save the changes before you change the user mode, but you can choose whether you want to or not.



#### 4.3.6.4 Hand

Click on this icon button to change to creation mode.



#### 4.3.6.5 Magnifier with plus sign

Zoom in one level (the fixed levels are: 10,15,20,25,33,50,75,100,200 and 400 %). Access key: Alt-I.



#### 4.3.6.6 Magnifier with minus sign

Zoom out one level (See 4.3.6.5 above for the fixed levels). Access key: Alt-O.



#### 4.3.6.7 Magnifier

Custom zoom in or out, enter a value of at least 1 (in percent). Access key: Alt-Z.



#### 4.3.6.8 Two houses

Neighbor view, see 4.3.5.3.1 Neighbors.



#### 4.3.6.9 List of squares/rectangles

Path view, see 4.3.5.3.2 Paths.



#### 4.3.6.10 Blue square with red links

Dynamic link view, see 4.3.5.3.3 Dynamic Link view.



#### 4.3.6.11 Tree of nodes

Subsystem view, see 4.3.5.3.4 Subsystem view.



#### 4.3.6.12 Arrow pointing upwards

Using the subsystem dependent navigation in the visual window, you go down in levels from the top (f. ex. LHCb system, Muon station, Muon station quadrant, Muon chamber), with the up arrow, you can also go the other way once you gone down one or more levels, to go up again, on level on each click.

#### 4.3.6.13 Arrow pointing downwards



This is only enabled in some specific cases, as f. ex. when you are down to the device level in the Muon system, and you want to go further down one level to see the chamber as two objects (the two double gaps as single devices).

### 4.3.6 The visual window

The subsystem dependent navigation lets us have different levels of detail view for each subsystem, and the number of levels depends on each subsystem. Two levels however are always the available, the top level is Level 0, and the bottom (device) level is Level -1. The bottom (device) level is where you create/modify/delete the objects to be stored in the ConfDB.

#### 4.3.6.1 LHCb system level (Level 0)

Right after you have connected to the database you will be in this level (right after startup). You will see an illustration of the LHCb system/detector, where each subsystem is clickable and will lead you to the subsystem level of the subsystem you chose. The arrow turns into a pointing hand when you move the cursor over a clickable area (here: subsystem), and you will see the name of the subsystem you are moving the mouse cursor over in the top left corner of the visual window. Already in this level, the Muon system is

divided into several clickable regions, one for each Muon station, which will lead you down one level to the station you click on. If you use the up-arrow on the toolbar in this level, you will go to the Device level (Level -1), where you can create/modify/delete objects.

#### 4.3.6.2 Subsystem level (Level 1)

This level is subsystem specific for each subsystem. F. ex. for the Muon system, you will see the Muon station you clicked on in a more detailed view. The up-arrow will bring you to level 0 (LHCb system level).

#### 4.3.6.3 Level 2 -> n

The numbers and visual appearance of levels depends on the subsystem. The up arrow will bring you to level 1 if you are in level 2, to level 2 if you are in 3 and so on. If you are in level n (out of n possible levels), and you click on something in the visual window, this will normally bring you to level -1 (Device level).

#### 4.3.6.4 Device level (Level -1)

As already mentioned, the device level is the level where you manage the objects that are stored in the ConfDB. You can click on them to select them and see information about them in the information window. You can move the objects around by clicking on them and holding down the left mouse button while moving the mouse. You can select several objects by dragging out a selection area around the objects you want to select, or by clicking on one and one when holding down the Ctrl-button on your keyboard. You can resize the objects (independent of the zoom), by selecting it and click on one of the squares in one of the corners of an objects, and drag out.

Each device is displayed with a color associated to a device type (by the user), as a rectangle, with its name centered on it. The ports of a device are shown on one or several of the four sides of the rectangle. Selecting a link that is connected to two devices brings up the information about the link (connection), and the port number and port type of the ports the link is connected to on each of the two devices.

If you click on the up arrow you will go up one level, to level n. If you click on the down arrow you will either end up at LHCb subsystem level (if a more detailed level of the device is not offered), or you will end up in a level with a more detailed view of the device (f. ex. in the Muon subsystem, you will get a view where the chamber is split up into two double gaps, and you can click on wire pads/cathode pads on each double gap to view information about the physical channels etc).

#### 4.3.6.4 Auto-zoom

Auto-zoom is a feature that will do what it can to make the visual objects in the visual window fall within the visible portion of the visible window, when it is on. The auto-zoom is default set to on, but it can be disabled by clicking on the View > Autozoom enabled, and uncheck this menu item.

This function will always try to maximize the view, that is, it will try to make you view the devices in a zoom as big as possible. There are however some restrictions:

The autozoom will never try to zoom lower than 25% of original size. Because at that level of zoom, the objects are so small that it is up to the user to decide whether he wants to zoom more or not. This limit can however also be changed or removed by removing the 25% check in the `ResizeToFitContents()` function in `visWindow.py`.

Another restriction is that this function has no effect on the layout of the visual objects itself, it **only zooms**, and the function will therefore not always give you a satisfactory result. F.ex.; the nodes selected are always put to the center, and the neighbors of this node is layed out in a way defined by the layout. If the node is connected to a huge device (with hundreds of ports), this device will most likely be layed out over the edge of the visible visual window, and it will require a lot of zoom to view the whole huge device, maybe down to 25%. And in this case, it could have been solved by moving the node in question closer to the

center of the window, and we would probably not need less than 90% zoom.

The auto-zoom will automatically be disabled if you use one of the zoom-functions explicitly, and you will have to turn it back on from the View menu if you want to enable autozoom again.

*Note, known issue:* In some very rare cases, the autozoom sets the zoom to 100% when it should have been set to a zoom much much lower.

## 4.4 Creation mode

In creation mode you can do all the things you can do in navigation mode, in addition to creation/modification/deletion of objects. In this section only the creation mode specific functions are described.

### 4.4.1 The menu bar

#### 4.4.1.1 File menu

##### 4.4.1.1.1 Mass insert data from file

Here you can load a \*.dat, \*.txt or \*.xml file with a valid format and syntax to create data objects in CdbVis. The files are iterated through from start to end, but the objects are not validated to see if they have valid "foreign keys", that is valid references to other objects. Also, if a device is created in the data file, you will have to create all ports for that device as well, but there is no check on this, but a limitation in CdbVis as you can not add new ports after you have created a device.

Objects that are valid are: Device type, Link type, Device, Link and Port.

##### 4.4.1.1.1.1 Txt or Dat file

The \*.txt and \*.dat has the same file format; it is just different file extension.

```
Datatype;Device;s|system_name;MUON,DAQ;s|devicename;FEE_MUON_7;s|responsible;;s|node;False;i|  
save_status;CREATE;i|hwttype;;s|devicetype;FEE;s|deviceid;;s|serialnb;;s|  
modify_status;NO_STATUS;i|user_comments;;s|location;;s|function_name;;s|
```

Example of syntax of an object in the data file.

Every line in the data file corresponds to one object; that is, if enter (or /r/n or /n is used) it is interpreted as a new line, so this must not be done for an object.

Each line (which is a data object) is divided into attributes with given value and data type, so all attributes are separated with a | (pipe). Within an attribute, the different data for the attribute is separated with a ; (semi colon). The first data for an attribute is the attribute name, the next the value for the attribute and the last signifies the data type of the attribute. The data type of an attribute can be s (string), i (integer) or w (color attribute).

Each data object must be identified with the Datatype attribute, telling what type of object it is, it can be: DeviceType, LinkType, Device, Link or Port. The data type of this is always string (s). The rest of the attributes are normally data object independent. Below is shown tables with all attributes for each data object type and which that is required and optional.

It is recommended to only use the save\_status set to CREATE (or also DELETE for links), as MODIFY and RENAME is a bit advanced and easy to do mistakes with.

### DeviceType

```
Datatype;DeviceType;s|system_name;MUON,VELO;s|save_status;CREATE;i|
description;safdasfsdf;s|rgbcolor;(255, 0,
0);w|devicetypeid;;s|devicetype;FEE;s|old_name;FE;s|nbrofoutput;3;i|nbrofinput;4;i
|
```

Example of syntax of a device type.

Attribute name	Example value	Data type	Optional/Required	Comment
Datatype	DeviceType	s	REQUIRED	Identifies the device type object
system_name	MUON,VELO	s	REQUIRED	Name of sub systems separated with comma
save_status	CREATE	i	REQUIRED	CREATE, MODIFY or RENAME. Whether the device type is created, modified, deleted or renamed
description	...	s	OPTIONAL	Device type description
rgbcolor	(255, 0, 0)	w	REQUIRED	Color of the device type, choose an rgb color.
devicetypeid	1	i	OPT/REQ	Required if MODIFY or RENAME of a device type that is already stored in ConfDB. To refer to the correct device type.
devicetype	FEE	s	REQUIRED	Name of the device type, the new name here if renamed.
old_name	FE	s	OPTIONAL	Only if RENAME, the old name of the device type.
nbrofoutput	3	i	REQUIRED	Number of output ports
nbrofinput	4	i	REQUIRED	Number of input ports

### LinkType

```
Datatype;LinkType;s|save_status;CREATE;i|concan_links;DATA LINK,;s|link_name;Test
link;s|old_name;Test2;s|link_nbr;-1;i|changed_complinks;1;i|
```

Example of syntax of a link type.

Attribute name	Example value	Data type	Optional/Required	Comment
Datatype	LinkType	s	REQUIRED	Identifies the link type object
save_status	CREATE	i	REQUIRED	CREATE, MODIFY or RENAME. Whether the device type is created, modified, deleted or renamed
concan_links	DATA LINK,	s	OPT/REQ	If the link is a composite link type this is set, an a comma separated list is given of the links it consists of
link_name	Test link	s	REQUIRED	Name of the link type, the new name if RENAME.
old_name	Test2	s	OPTIONAL	Only if RENAME, the old name of the device type.
link_nbr	-1	i	REQ/OPT	The link type ID if already stored in the ConfDB
changed_complinks	1	i	REQ/OPT	If the link is composite link type, and the list of link types were changed



Attribute name	Example value	Data type	Optional/Required	Comment
Datatype	Device	s	REQUIRED	Identifies the device object
save_status	CREATE	i	REQUIRED	CREATE, MODIFY or RENAME. Whether the device is created, modified, deleted or renamed
devicename	FEE_MUON_7	s	REQUIRED	The name that identifies the device, the new name if it is renamed.
responsible	John Doe	s	OPTIONAL	Name of the person responsible for the device
node	False	i	REQUIRED	Whether the device is a node or not, valid values are: True, False, 0, 1
hwtype	...	s	OPTIONAL	The name of the hardware type
devicetype	FEE	s	REQUIRED	The device type of this device, the name of the device type as it is given
deviceid	-1	i	OPT/REQ	Required if MODIFY or RENAME device.
serialnb	...	s	REQUIRED	The unique serial number of the device
modify_status	NO_STATUS	i	OPT/REQ	If the device is being modified, it is required. Valid values then are: 0 (if the node, promiscuous mode or location value is being changed), 1 (if system is being changed)
system_name	MUON,DAQ	s	REQUIRED	Name of sub systems separated with comma
user_comments	...	s	OPTIONAL	Comments for device
location	...	s	OPTIONAL	Location of device
function_name	...	s	OPTIONAL	Function of the device such as DHCP, DNS

### Device

Prerequisite: The device type of this device must have been created.

```
Datatype;Device;s|system_name;MUON,DAQ;s|devicename;FEE_MUON_7;s|responsible;;s|node;False;i|
save_status;CREATE;i|hwtype;;s|devicetype;FEE;s|deviceid;;s|serialnb;;s|
modify_status;NO_STATUS;i|user_comments;;s|location;;s|function_name;;s|
```

Example of syntax of a device.

## Link

Prerequisite: The devices that this link is connected between must have been created, and the link type of this link must have been created.

```
Datatype;Link;s|linktype_changed;0;i|node_from;FEE_MUON_3;s|save_status;CREATE;i|port_typedto;AB;s
port_nbrfrom;0;s|linkid;TMP_LINK_1;s|node_to;M5R4#2;s|linkused_changed;0;i|bidir_changed;0;i|
lkused;1;i|port_typefrom;;s|bidirectional_link_used;0;i|link_type;DATA LINK;s|port_nbrto;0;s|
link_info;cross 5 pacth panels;s|
```

Example of syntax of a link.

Attribute name	Example value	Data type	Optional/Required	Comment
Datatype	Link	s	REQUIRED	Identifies the link object
save_status	CREATE	i	REQUIRED	CREATE, MODIFY or DELETE. Whether the link is created, modified or deleted.
linktype_change d	0	i	OPTIONAL	On modify, whether the link type has been changed
node_from	FEE_MUON_3	s	REQUIRED	The device the link goes from, device name
node_to	M5R4#2	s	REQUIRED	The device the link goes to, device name
port_typedto	AB	s	REQUIRED	The port type name of the device to, if any
port_typefrom		s	REQUIRED	The port type name of the device from, if any
port_nbrfrom	0	s	REQUIRED	The port number name of the device from
port_nbrto	0	s	REQUIRED	The port number name of the device to
linkid	TMP_LINK_1	S	REQUIRED	Unique temporary link name before added to the ConfDB, if in ConfDB, the link ID

---

linkused_change	0	i	REQ/OPT	If modify, whether link used attribute is changed or not
bidir_changed	0	i	REQ/OPT	If modify, whether bidirectional attribute is changed or not
lkused	1	i	REQUIRED	Whether link is used or not
bidirectional_link_used	0	i	REQUIRED	Whether link is bidirectional or not
link_type	DATALINK	s	REQUIRED	name given for the link type of the link.
link_info	Cross patch panels	s	REQUIRED	Information about the link. Put "none" if nothing special

Attribute name	Example value	Data type	Optional/Required	Comment
Datatype	Port	s	REQUIRED	Identifies the port object
save_status	CREATE	i	REQUIRED	CREATE, MODIFY or DELETE. Whether the port is created or modified.
devicename	FEE_MUON_8	s	REQUIRED	The device (name) this port is a part of.
portid	-1	i	OPTIONAL	The port id, if already stored in ConfDB
port_nbr	0	s	REQUIRED	The port number within this device
port_type	...	s	REQUIRED	The port type name of the port, if any
macaddress	none	s	REQUIRED	Mac address of the port, none if not any
ipaddress	None	s	REQUIRED	IP address, None if not any
administrative_status	0	i	REQUIRED	Administrative status of the port
pxi_booting	1	i	REQUIRED	PXI booting or not
bia	none	s	REQUIRED	Bia, none if not any
port_way	1	i	REQUIRED	Way of the port, 0 or 1
phy		s	REQUIRED	Phy value, if any
old_ip		s	OPT/REQ	If MODIFY and IP changed, set old ip here
ipname		s	REQUIRED	IPname, if any
subnet_info		s	OPT/REQ	Subnet address, if any
speed	12	i	REQUIRED	Speed of port
modify_status	1	i	OPT/REQ	If modify. 0 if port speed, phy or pxi changed, 1 if subnet or ip name changed, 2 if ip address changed.

## Port

Prerequisite: The devices that this link is connected between must have been created, and the link type of this link must have been created.

```
Datatype;Port;s|devicename;FEE_MUON_8;s|save_status;CREATE;i|portid;-1;i|port_nbr;0;s|
```

```
macaddress;none;s|ipaddress;None;s|administrative_status;0;i|pxi_booting;1;i|
```

```
bia;none;s|port_way;1;i|phy;;s|old_ip;;s|port_type;;s|ipname;;s|subnet_info;;s|  
modify_status;-1;i|speed;12;i|
```

Example of syntax of a port.

#### 4.4.1.1.1.2 XML file

The XML mass insertion is similar to txt and dat, except that the objects are in xml format. Still the same name of the attributes and valid attribute values and attribute data types.

```
<?xml version="1.0" encoding="UTF-8"?>  
<Dataobjects>  
  <Device>  
    <system_name>MUON,DAQ;s</system_name>  
    <devicename>FEE_MUON_8;s</devicename>  
    <node>False;i</node>  
    <save_status>CREATE;i</save_status>  
    <hwtype>;s</hwtype>  
    <devicetype>FEE;s</devicetype>  
    <deviceid>;s</deviceid>  
    <serialnb>;s</serialnb>  
    <modify_status>NO_STATUS;i</modify_status>  
    <user_comments>;s</user_comments>  
    <location>;s</location>  
  </Device>  
  <Device>  
    <system_name>MUON,DAQ;s</system_name>  
    <devicename>FEE_MUON_8;s</devicename>  
    <node>True;i</node>  
    <save_status>MODIFY;i</save_status>
```

```
<hwtype>gah;s</hwtype>

<devicetype>FEE;s</devicetype>

<deviceid>;s</deviceid>

<serialnb>;s</serialnb>

<modify_status>NO_STATUS;i</modify_status>

<user_comments>hohohoh, nice device;s</user_comments>

<location>;s</location>

</Device>

</Dataobjects>
```

Example of an XML file, with two devices pending to be stored; the first create the device, the second modifies it.

First reading how to use txt and dat files will make you understand the format and syntax of this xml file.

The mandatory syntax of this xml file is that the first line is: `<?xml version="1.0" encoding="UTF-8" ?>`

and the second and the last line is respectively `<Dataobjects>` and `</Dataobjects>`. The data objects are set up within those two element tags, with the same name as in the Datatype attribute for txt and dat files: DeviceType, LinkType, Device, Link or Port, as shown with Device in the example above. The attributes for a data object are given as child elements of the respective data object element, and the attributes are split in two with the value to the left of the semi colon (;) and the data type on the other side.

Use the XML example above as a template for your own data objects.

#### 4.4.1.1.2 Store objects in ConfDB

If you have created/modified/deleted any objects, ConfDB will be updated when you choose this option. There is a check before saving to see if there is anything to save, if a device has been created but later deleted, there is no point in first creating it and then deleting it, therefore are both deleted from the list of actions/queries to the ConfDB.

The objects will be stored in the order: device type, link type, device, port and link, so that we will not get any inconsistency or violate any constraints in the ConfDB.

If an error occur while updating the ConfDB for an object, we will still try to save as many objects as possible (because the object might have been violating a constraint etc.), and the objects that failed will be put aside, still being able for the user to change (so that they can be stored in ConfDB later). That is, if we are about to store 1000 objects, and 23 of them fail, the 23 that failed to be stored will be put in the list where they are still pending to be saved. You will also be informed about which object(s) that was not saved and what that went wrong, so that you can change it/them in order to save them to ConfDB.

#### 4.4.1.1.3 Generate ConfDBLibrary functions file

With this feature you can first create the devices, links etc. in CdbVis, and then create a python script file that will commit the changes you made to them to the ConfDB whenever it is executed.

If you f.ex. create a device in CdbVis, it is temporarily stored in the program's memory. If you choose to commit this device to the ConfDB, it is removed from the program's memory after it has been committed. But if you choose to store the changes (create information for the device you created) in a python script file, a line is added to a python script file with the command and all the attributes set needed to commit the device to the ConfDB (see below). This python script file can also be modified by the user. You can f. ex. copy this lonely line of insertion of a single device to many more, by just changing the name and serial number to be unique.

You must also remember to set database name, username and password in the beginning of the script yourself, or that you set this in the GenerateSQL() function in cdbVis.py.

To understand/know what data that is needed in the different parameters to the functions that is printed in this file, have a look at the documentation for the ConfDBLibrary here: [http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/download\\_libraries.htm](http://lhcb-online.web.cern.ch/lhcb-online/configurationdb/download_libraries.htm)

```
#####  
#      AUTOGENERATED PYTHON CODE      #  
#      BY CDBVIS                       #  
#####  
  
from confDBpython import *  
  
cfDB = CONFDB("databasename","username","password") #NB! Set databasename, username and pwd  
for ConfDB  
  
cfDB.DBConnexion()  
  
# Start of ConfDB Commit  
  
cfDB.InsertFunctionalDevice("MUON,DAQ","FEE_MUON_7","FEE",1,0,"","serial number","","","hohohoh,  
intresting",1)  
  
# End of ConfDB Commit  
  
cfDB.DBDeconnexion()
```

Example file for a created device stored in a python script file, to be commit to ConfDB when this script is

run.

#### 4.4.1.2 Edit menu

##### 4.4.1.2.1 Undo

If you have created, modified and/or deleted something in creation mode, and it turned out that it was not what you had in mind or of any reason do not want to have it that way, you are able to undo this. Then the undo option on the menu will not be grayed out any more, but it will be changed to undo + “description of action”. Choose to undo, and the action you undid will be reset. Then the action you did before the action that you undid will be the new option to undo and so on.

##### 4.4.1.2.2 Redo

If you have undone something, and you of some reason want to undo the undone, then you choose this option. When you click to undo an action, this action will be added to the redo option, if you changed your mind.

#### 4.4.1.3 Create menu

##### 4.4.1.3.1 Create Device type

If a subsystem is active, you can choose to create a device type. A window pops up, and you can set the different properties for a device type:

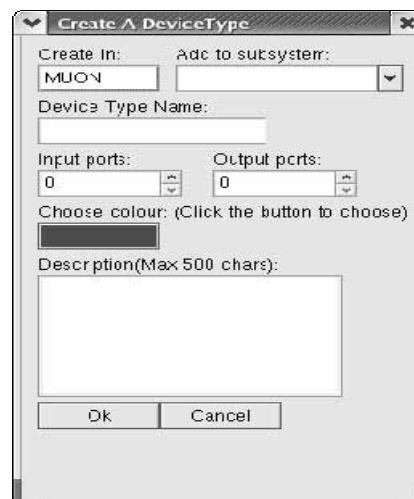


Figure 11 The Create Device Type dialog, here the Muon subsystem is the active system.

**Subsystem(s)** to create the device type in: First you can choose which subsystem(s) this device type should be accessible from (of course in all subsystems that have the same abstract device type with the same properties). But if you are not sure, you will only add the device type to the subsystem you are familiar with. You choose subsystem(s) to add the device type to with the combo box to the left of the text control. (REQUIRED)

**Device type name:** You will also have to assign a name to the new device type. (REQUIRED)

**Number of input ports:** The number of input ports can be set to a number between 0 (default) and 10 000 (REQUIRED)



**Number of output ports:** The number of output ports can be set to a number between 0 (default) and 10 000 (REQUIRED)

**Color of the device type:** To be able to easier distinguish the different devices from each other in the visual window, you assign the same color to all devices of a device type. If not changed, it will be colored red (default), or you choose a color by clicking on the colored button and choose from the color dialog that appears. (OPTIONAL)

**Description:** Give a description of the device type, maximum 500 characters (OPTIONAL)

When the device type is created (you clicked the OK button), it's immediately added to the list of device types you can choose to create a device from.

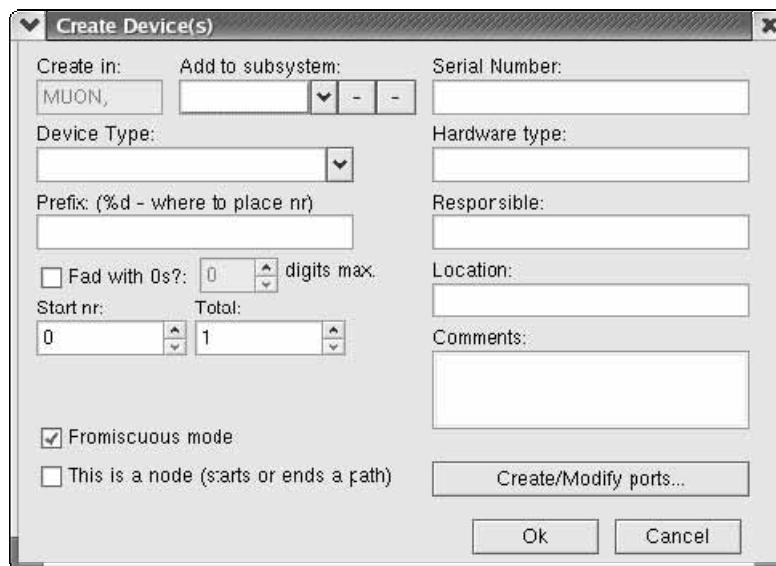


Figure 12 The Create Device Dialog, here the Muon subsystem is the active subsystem.

#### 4.4.1.3.2 Create Device(s)

If a subsystem is active you are able to create one or several devices, and the port information to every port of each device. When you create a device both the hardware device and the functional device is inserted in the ConfDB. The hardware device is the information for a specific independent physical device, while the functional device is the information about its connections, the location etc. associated with the location the device is installed in in the ConfDB.

A window will pop up where you have to fill in information about the device(s) you're about to create:

**Create in subsystem:** You choose which subsystem(s) the device(s) should be created in by choosing the subsystem in the combo box, click on the button with the plus sign to add the subsystem to the list of subsystem(s) it will be created in, or the button with the minus/negative sign to remove the subsystem from the list of subsystem(s) it will be created in. (REQUIRED)

**Select device type:** A device has to be one kind of a device type, and you choose the device type from the combo box. If there is already at least one device formerly created of the given device type, the program will try to guess the syntax of the devices. (REQUIRED)

**Choose prefix (device name syntax):** You can change the prefix if the program fails to guess after you selected the device type, but only you can tell if the program guessed correctly or not. If there are not any devices created of the given device type or the program failed in guessing the correct prefix, you will have to set it up yourself. What you do is to write the part of the device name that is common for every device of this device type, and then add '%d' (without the quotes), to the name where you want it to be replaced by

the device number/id. At the moment it is required that the %d is a part of the name, also if you only create 1 device, but you can remove the number next time you rename the device. (REQUIRED)

**Choose padding:** You can choose to pad the device id/number with leading zeroes. The number you choose in the spin box indicates the maximum number of digits for the id/number for a device. If the maximum number of devices is more than 1000 but less than 10 000, then you put 4 as maximum digits, if it is between 10 and 100, you put 2 and so on. If you chose 4 as maximum number of digits, 1 will be padded like this: 0001, and 6748 as 6748. (OPTIONAL)

**Start number and total number:** Start number indicates on which number/id you want the devices you create to start on (if f. ex. 5 devices is already created with the same syntax, and it started with id 0, then you will most likely want to continue with id (start number) 5. The total number is the total number of devices you want to create now; they will be numbered from the start number to start number + total number of devices. (REQUIRED)

**Promiscuous mode:** Check this one if the device will run in promiscuous mode (f. ex. PCs) (REQUIRED)

**Node:** If the device will be a node, you check this check box. A node means that a device is at either end of a path. (REQUIRED)

**Serial number:** UUID, bar code or some other kind of a unique id of a device. At the moment only one serial number can be given to the collection of devices that you create, therefore the serial number property has to be modified for every device after a mass creation. (TODO) (REQUIRED)

**Hardware type:** hardware type of the device. (OPTIONAL)

**Responsible:** Who that is responsible for the device. (OPTIONAL)

**Location:** Location of the device in the LHCb system. Location ID is not a common convention for every subsystem; they may have totally different formats. The location ID is usually unique for every device, therefore this property must also be changed after the mass creation(TODO)(OPTIONAL)

**Comments:** comments or description of a device. (OPTIONAL)

**Function:** function of the device, should be selected among one of the list

#### 4.4.1.3.2.1 Create/Modify/Delete Port(s)

The screenshot shows a 'Create Port(s)' dialog box with the following fields and options:

- Device Name(s): MSR4#5
- Port Number(s): 0-9,11-20,10,21;
- Port Type(s): A,B;AB;
- Port Way:  In  Out
- Port Speed: 0
- Phy: [dropdown menu]
- PXI Booting?:
- Adm. Status:
- MAC Address: [text field]
- IP Address: [text field]
- Subnet Address: [text field]
- IP Name: [text field]
- BIA: [text field]

Buttons: Add port(s), Edit, Remove, OK, Cancel

Device Name	Port Nr	Port Type	Port Way	Port Speed	PXI?
-------------	---------	-----------	----------	------------	------

Fill with port data from other device...

Figure 13 The Port Creation Dialog, opened from the Device creation dialog, you can see that in this case we are creating ports for a device named M5R4#5.

When you have set the different properties for a device(s) you can also set the port properties for the port(s) on the given device(s). You click on the button Create/Modify ports and a new window pops up. You can choose to either fill the information "manually" or copy port information from a previous device of the same device type by clicking on the button down to the left and choose a device to fill port information from. After that you can change the properties that differ, if any. If this is the first device of the given device type or you for any other reason do not want to get port information from another device, you do the following:

**Device name:** Here you will see the name of the device (or the range of devices) that you will create the port info for.

**Port numbers:** To make it more efficient to create many ports (especially if a device have many ports), you can add a range of port numbers by writing 0-9 in the text control, which will add 10 ports with the port number from 0 to 9. You can also add single ports like this: 0, 3, 5, 6, which will create 4 ports with the respective port numbers. And you can combine those like this: 0-9, 11, 13, 15-20. If the port number consists of letters or other characters you will have to add it like: A, B, C, D etc.

**Port type:** You can add several port types at the same time, as with port numbers you can add single port types separated by commas: AB, CD, A, B.

**Port Way:** Choose whether the port is used for data going to the device, or out from the device.

To add many ports in one go with different combinations of port numbers, port types and port way, you can set this up quite easily. All the port numbers given inside a semi colon in port number will be set up with every combination of port types within the same semi colon, and all these ports will be given the port way checked in port way checkbox. F. ex.: For an M5R4 Muon chamber we have the following ports:

Port number 0-23 of the port type AB, all with the port way: OUT  
Port number 0-23 of the port type CD, all with the port way: OUT  
Port number 0-4 of the port type ABCD, all with the port way: IN  
Port number 5 of the port type ABCD, all with the port way: OUT

This can be added fast and efficient like this:

1.

Port number: 0-23;5

Port type: AB,CD;ABCD

Port way: OUT

Here we say that we want all combinations of port number 0-23 and port type AB, and all combinations of port number 0-23 and port type CD, all of the of port way OUT before the first semi-colon (which is 48 combinations). After the semi colon we say that we want all combinations of 5 and ABCD (which is obviously only one)

When that is added you add this:

2.

Port number: 0-4

Port type: ABCD

Port way: IN

All combinations of 0-4 and ABCD which yields: 0 and ABCD and IN, 1 and ABCD and IN, ..., 4 and ABCD and IN.

When all the different port combinations is added, you can add specific properties for each port (they can all be set at the time you create the port combinations but there are probably some differences between the ports on several properties, at least MAC, IP etc if they are needed). To change settings for one port only, double click on the row of the port or select the row and choose edit. If you want to remove a port, you can select a row and click on the remove button. Remember that the port number, port type and port way cannot be changed (modified) once a device is created.

**Port speed:** Set the speed of the port. (OPTIONAL)

**Phy:** The following values are valid: " ", "SX", "T", "SL" (OPTIONAL)

**PXI booting:** Check if true. (OPTIONAL)

**Administrative status:** Check if true. (OPTIONAL)

**MAC address:** the MAC address in the format: 0X-0X-0X-0X-0X-0X (OPTIONAL, but REQUIRED in DAQ)

**IP address:** the IP address in the format: 000.000.000.000 (OPTIONAL, but REQUIRED in DAQ)

**Subnet address:** the subnet address in the format: 255.255.255.255 (OPTIONAL, but REQUIRED in DAQ)

**IP name:** (OPTIONAL, but REQUIRED in DAQ)

**Bia:** (OPTIONAL)

When you have finished filling out all the data, there will be a validation checking that the number of ports with port way OUT and port way IN corresponds to the number of in and out ports of the given device type, and also that the number of ports created corresponds to the number of ports of the given device type.

If all succeeds you will be able to create the number of devices you specified, all with the same information/properties except from the device name.

#### 4.4.1.2.3 Create Link Type

If a subsystem is active, you can choose to create a link type. A window pops up, where you can fill in data to create a link type. When you create a link type, it'll be accessible in all subsystem in the database.

Give a name to the new link type. (REQUIRED)

If the link type is composite (consists of several other link types) because it can transfer different kind of data, you will have to check the "Composite link type?" check box. Then you add link types from the combo box by clicking on the add button, and remove by clicking on the remove button.

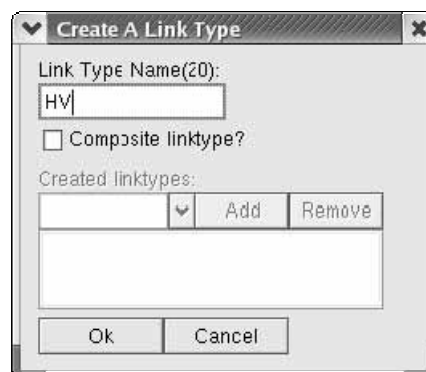


Figure 14 The create link type dialog, here we are creating a link type with the name HV.

#### 4.4.1.3.4 Create Connection (Link)

If a subsystem is active, you can choose to create a connection between two devices. A window pops up where you have to set the different properties for the connection. Note that you will have to create port objects for the ports you want to connect the link between, it is the name of this port that appears in the

select box underneath Port: which you see on the screenshot to the right. First you choose the Link Details tab:

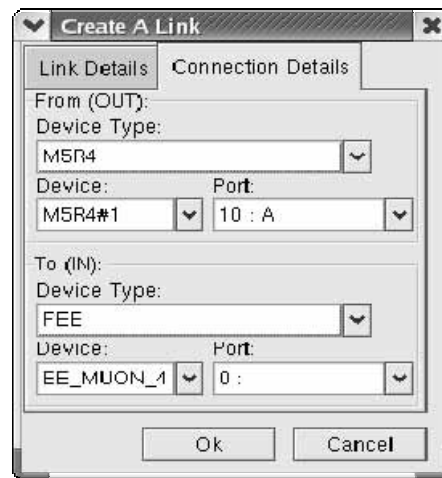


Figure 15 The create link dialog, where we set the devices and ports the link will be connected between

The link will be created in the system(s) where the two devices you choose to connect together are in (they do not have to be in the same subsystem)

You choose a **link type** for the given connection. (REQUIRED)

The link will be given a temporary name in this session, which is only changed when it is stored in the database (the name is then set to the link ID it is given in the ConfDB)

When you create a link it is default set to be **used**, and cannot be changed while creating the link.

You can choose to set the link as **one-directional or bi-directional** (whether data is transferred one or both ways)

In the **Connection Details tab** you set up which two devices to connect together. For both devices you choose first device type, next the device and finally the port number – port type combination. Only free/available port number-port type combinations are shown, and if none are free, you are not able to set up a connection to the given device.

#### 4.4.1.3.5 Create Port(s)

Choose to create or modify a device, and from there you click on the button that says add/modify ports, and you are able to add port information for the device. Have a look at create and modify device on how to do this.

#### 4.4.1.4 Modify menu

##### 4.4.1.4.1 Modify Device type

A device or a device type has to be active (selected) before you can choose this option. The device type of the currently selected device will be selected if no device types are selected and an almost identical window to the "create device type" will pop up:

All the properties that are set for the given device type will be shown, and most of them can be altered. You cannot change the subsystem(s) it is available in, and not the Device ID (which is shown if the device type have been stored in ConfDB), all the rest is possible to modify.

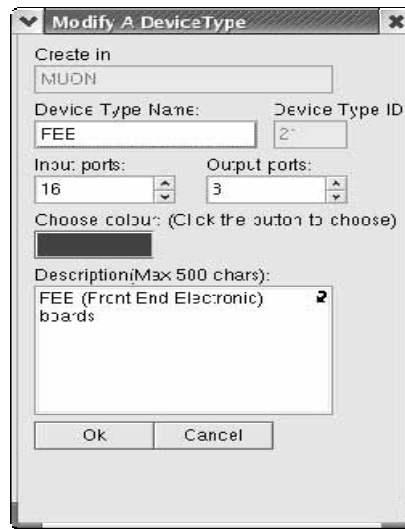


Figure 16 Modify a device type dialog, in this screenshot it is a FEE that is modified.

#### 4.4.1.4.2 Modify Device

If a device is active, you can choose this option, and a window almost identical to the “Create device” pops up:

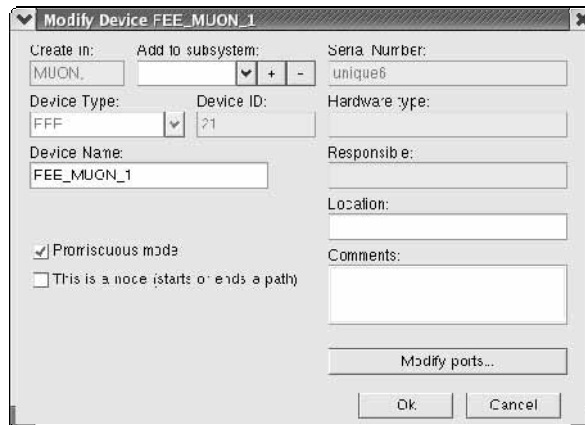


Figure 17 The modify device dialog, in this example an FEE board is modified, the disabled areas are attributes that can not be modified.

You can change the system it is available in, the device name, whether it runs in promiscuous mode or not, whether it is a node, its current location and comments if it is stored in the ConfDB.

However, if you have created the device, but not yet stored it in the ConfDB, you can also change serial number, hardware type and the person that is responsible for the device.

You can also alter the different properties for the ports of the device. If the device (and ports) is (are) stored in the ConfDB, you can change everything for a Port except port name, port type, port way, MAC address, administrative status and BIA. If it is not stored in ConfDB yet you can change everything except port name,

port type and port way.

When you choose to modify a device, you can always add more ports, modify the port information or delete ports. You choose to modify the device you want to create, modify or delete ports for, and then choose modify ports. You can click on the number of the row to select a row, and then click on edit to edit that row. If you change your mind, that you do not want to edit the row, you click on cancel edit, and the values in the text fields etc. are set back to default. To delete a port, you select a row, and click on Remove.

Remember that there are several functions/features that do not work as wanted if ports are not assigned and set for a device. E.g. to connect two devices together, there must be at least 1 free port on each device, and the port object must have been created for those two devices (for the given port they are being connected on). Also the clone feature needs that at there are a number of free ports on the device, and that the number of new connections to set up must be maximum the number of free port objects for the given device. It is always a good rule to create all the ports for a device, once you create a device, it is also more efficient.

#### 4.4.1.4.3 Modify Link type

If a link type is active, you can choose this option, and a window almost identical to the “Create link type” pops up:

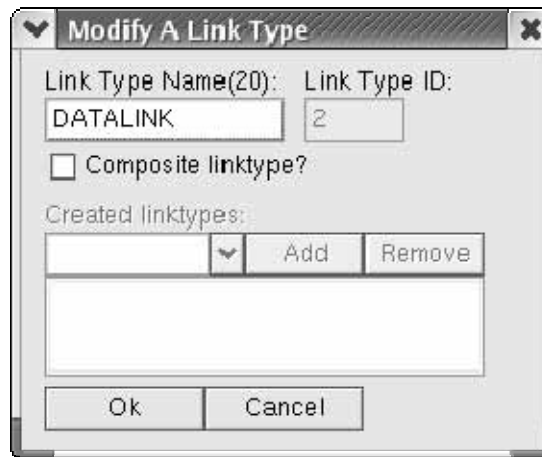


Figure 18 The Modify Link Type Dialog, in this example it is the data link link type that is modified.

You can change the name of the link type, and you can also change it from simple link type to composite link type, or from composite link type to simple link type.

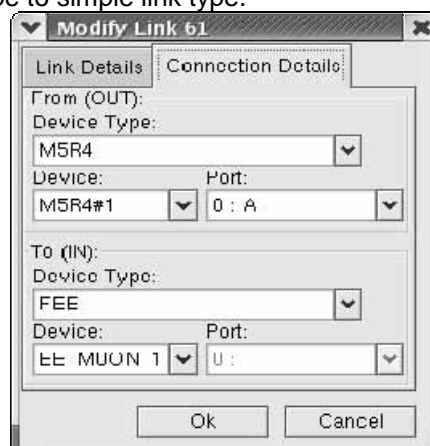


Figure 19 The modify link/connection dialog, here you can change the connected ports and devices and other attributes.

#### 4.4.1.4.4 Modify Link

If a link is active, you can choose this option, and a window almost identical to the "Create link/connection" pops up. All the properties for the link can be altered, except for the Link name/ID.

#### 4.4.1.4.5 Modify Port(s)

Choose to modify a device, and from there you can click on the modify ports button, to add, modify or delete ports for the given device. Have a look at Modify device, on how to do this.

#### 4.4.1.5 Delete menu

##### 4.4.1.5.1 Delete Device type

If a subsystem is active, you can choose this option, but you can only delete device types that are NOT yet stored in the ConfDB.

##### 4.4.1.5.2 Delete Device/Link

One or more devices and/or links can be deleted at the same time.

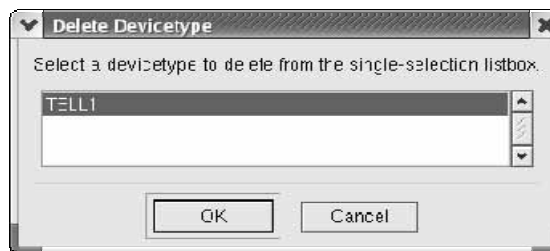


Figure 20 The Delete Device type dialog

A link can be deleted at any time, there are no restrictions, and also links that are already stored in the ConfDB can be deleted.

There are some rules on when one can delete a device.

A device that is already stored in the ConfDB can NEVER be deleted.

A device that is not stored in ConfDB, but is connected to one or more links that is not selected (and to be deleted), cannot be deleted. The reason why is that we do not allow links to exist as free or partially free links (a free link is a link that is not connected to any devices, a partially free link is a link connected to only one device).

A device that is not stored in ConfDB, and is connected to one or more links where all of them are selected, can be deleted.

A device that is not connected to any links can be deleted.

When you delete a device, all the ports associated with it are also deleted. (CdbVis is responsible for this)

When you delete a link, the two ports it is connected are NOT deleted.

Access key: Alt-R.

##### 4.4.1.5.3 Delete Link type



If a subsystem is active, you can choose this option, but you can only delete link types that are NOT yet stored in the ConfDB.

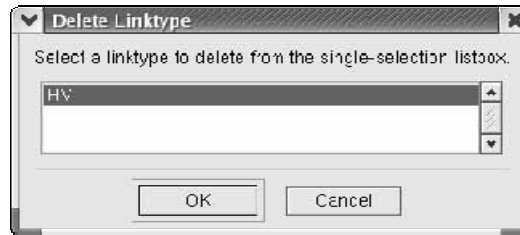


Figure 21 The delete link type dialog

#### 4.4.1.5.3 Delete Port(s)

Choose modify device, and by clicking the Modify ports button, you are able to delete one or more ports for a device. When you delete a port, the link connected to this port is also deleted (ConfDBLibrary is responsible for this, so you will not see the effect before after committing the pending data objects to the ConfDB).

### 4.4.1.6 Visual menu

#### 4.4.1.6.1 Object Manipulation

##### 4.4.1.6.1.1 Duplicate...

See 4.4.2.1

##### 4.4.1.6.1.2 Clone...

See 4.4.2.2

##### 4.4.1.6.1.3 Delete

See 4.4.2.3

## 4.4.2 The tool bar

Going from left to right, we describe the icons/buttons specific for creation mode only. The name of the icon is given by a description of the bitmap you see on the icon.

### 4.4.2.1 A sheet of paper with a plus sign – Duplicate



selection

Hold your mouse cursor over the icon and it will display the tool tip "Duplicate selection". You select the device(s) (and links) in the visual window that you want to duplicate. There are some rules to follow when duplicating:

A device can be selected without selecting any of its connections (links), for duplication.

If you select a link for duplication, you must also select BOTH devices it is connected to, to be duplicated. This is because we do not allow the existence of free links or partially free links (See terminology list for free links definition).

When you choose to duplicate devices and links, you will get an exact copy of what you duplicated, but with one exception; the device and link names will be different from the original ones. Also, the links that were duplicated will be connected to exactly the same ports on the new devices as they were on the original ones (the corresponding position).

Access key: Alt-U.

#### 4.4.2.2 Two sheets of paper - Clone



Hold your mouse cursor over the icon and it will display the tool tip "Clone selection". You select the device(s) (and links) in the visual window that you want to clone. Cloning and duplication is quite similar, but with cloning you can do the same as with duplication in addition to some more. The reason why this is split into two different actions is to hopefully make it easier for the user. With duplication, you only duplicate selected structures; nothing dynamic or "magic". With cloning you can clone, a single link, a device and none or one or more of its connections etc. Everything is possible to clone. There are some rules of what is the result when cloning:

If you clone a single device, you will just get a duplicate; same as with duplication.

If you clone a single link, you will not get a free link, but a new link connected to the same devices as the original link was connected to, but to the next free ports on the devices. If there are no free ports on any of the two devices, cloning is not possible.

If you select one device, and the links connected to it, a new device will be created with links connected to it and to free ports on the device(s) that were connected to the other end of the links.

As a summary: if you do not clone a device that is connected to links that you clone, the new links that are cloned will connect to the same device as the original links. If you also clone a device that is connected to a link, the new cloned link will connect to the cloned device.

Remember that you will need free ports on a device to clone, and you must have created port objects for those free ports already, if not the clone feature will not work as intended.

Access key: Alt-C



#### 4.4.2.3 Trash bin – Delete selected objects.

This is to delete devices or links in the visual window. See 4.4.1.5.2 Delete Device/Link.

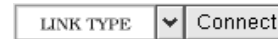
#### 4.4.2.4 Floppy disc - Store pending objects in ConfDB

Store pending objects in ConfDB. See 4.4.1.1.1 Store objects in ConfDB



#### 4.4.2.5 Combo box of device types and button showing a device template

You can choose a device type from the combo box, and click on the "device" button to the right of it (similar to the red button shown on the illustration above), and drag it over to the visual window. A "Create device" window will then pop-up, with some data filled out for the given device, but where you will have to fill in the rest. See 4.4.1.3.2 Create Device(s).



“Connect”

#### 4.4.2.6 Combo box of link types and a button with the title

To create a link, you choose the link type you want, and select two devices in the visual window, then you click on connect. Some of the required data areas will be filled out in the pop up window (f. ex. which devices to connect between, and a suggestion to which ports to use). For more see 4.4.1.3.4 Create Connection (Link).

### 4.4.3 The visual window

#### 4.4.3.1 Device level (Level -1)

This is the default level to work on objects in the visual window. Here you will see generic devices as rectangles with ports on the sides (1 – 4). The visual objects in this level are fully clickable (click-sensitive in the sense that a custom menu pops up when you right click on a visual object). Here is a short description of the features in the visual window:

##### 4.4.3.1.1 Pop-up menu

Right-click on an object (device or link) and a menu will pop up. You can select several devices and/or links while single-clicking them and holding down the Ctrl-button. The menu that appears, and the function chosen on the menu will affect all selected objects.

##### *Duplicate*

Duplicate the selected object(s). See 4.4.2.1 *A sheet of paper with a plus sign – Duplicate selection.*

##### *Delete*

Delete the selected object(s). See 4.4.1.5.2 *Delete Device/Link*

##### *Clone*

Clone the selected object(s). See 4.4.2.2 *Two sheets of paper - Clone selection*

##### 4.4.3.1.2 Visual features

##### *Select several objects*

Hold down the Ctrl-button on your keyboard while clicking on the objects that you want to select.

##### *Change link connection*

Click on a link to get it selected.

Click on the black square on one of the end points of the link, hold down the left mouse button, and drag the link over to another device.

Release the left mouse button over the device you want to change the link end point to.

The “Modify link” window will pop up with the new connection set up. Make changes if you want to, and click OK to make the change of the link connection take affect.

#### 4.4.3.2 Muon level 2 (Quadrant view of chambers)

If you click on a chamber position where a chamber is already installed you will get the standard view of the

given chamber that is installed in that position.

If you click on a chamber position where a chamber is not installed yet, you will get a message box asking you whether you want to create a new chamber with the given position set in the location property, or to set this property of an already existing chamber.

## 5. Known issues

### 5.1 Known bugs/problems

#### 5.1.1 General

The connection to the ConfDB does not have a time out, and will keep trying to connect until success or failure (if there is no response, it will just keep trying to connect)

Sometimes when changing between navigation mode and creation mode (probably due to a double-click combination or something), the button for the new mode is changed, but not the restrictions/features.

It seems that Python sometimes can execute functions in parallel due to several events occurring, even though the code is not set up to handle parallel execution. This may lead to unexpected behavior (f. ex. pressing Ctrl-Z (Undo) and Ctrl-R (Redo) fast after each other)

Even though undo/redo is configured to show the undid/redid objects in the visual window, it sometimes does not happen, but this is just a visual issue, as the objects availability is not changed. To be able to really see that the undo/redo worked, you should double click on the object, or an object connected to it. Because we work with wxPoint() objects which take integers as parameters, for positioning objects in the visual window, we lose the exact relative positioning when zooming out too much, and the layout when zooming back in (to normal scale) is somewhat bad. Can be changed if we use wxRealPoint() instead, but this is NOT just to replace wxPoint() string with wxRealPoint() because some functions are not compatible with wxRealPoint()

Creating many objects, or displaying many objects in the visual window at the same time (> ~1000) worsen the performance, the program will react significantly slower on events. This is due to both some lack in optimization and that Python is quite slow (compared to C).

Autozoom sometimes zooms to 100% when the zoom in fact should be much lower.

The guessing algorithm for guessing a device name given a device type does not work for device types that have different formats/syntaxes for the devices.

You are not able to modify several ports at the same time, even though it seems that the possibility is there. Path view is quite slow (due to quite a lot of processing in ConfDB)

The N-Hop device view should not be used to view more than maximum 3-4 hops, as it will be way to many devices to display. The layout for this view is not really good either.

The dynamic link view does not work properly for cyclic connections.

#### 5.1.2 Unix/Linux

The layout of widgets on the program window is not always good.

It can sometimes be difficult to choose an item from a combo box.

There are some memory leaks in wxPython (at least in the 2.4.2 version for Linux), which may cause the program to crash/terminate unexpectedly. This seems to occur quite random.

#### 5.1.3 Windows

Repainting of the program window is sometimes "skipped", especially repainting disabled widgets and before you connect to the database. I suggest that you always click Ctrl-C right after startup of the window

to connect to the database.

It is not possible to automatically expand the nodes in the tree view when double-clicking, they have to be expanded by clicking on the plus sign to the left of the node after double-click. The Subsystems node item is collapsed without a plus sign, but you can expand it again by a single-click just to the left of the node.

Sometimes the window dialogs end up behind the program window when they are shown, but remain active at the same time. The dialog window is then shown by choosing Alt + tab on your keyboard.

Sometimes when you double-click on a node in the tree view, it is either not expanded, or the one above or below is expanded.

The functions used to get the list of a device type, a device, a port are executed twice (at least) when double-clicking on a node. It is a problem as it reduces performances.

## 5.2 Missing features

- A feature that makes it easier/more efficient to replace a device.
- Moving between levels is not always intuitive or flexible enough.
- Possibility to add test boards to the ConfDB.
- Possibility to add microscopic board components, as well as viewing them.
- At the moment it is only possible to store and view functional devices.
- Let the user set up custom settings
- Be able to center to a device
- Be able to change the link type to view for a group of devices dynamically, so that you can switch between different connections between devices, on different cables right away.
- Create a layout manager that should manage the layout (...)
- Global settings file for ports (in AFS), in case the port indexing varies for the default. And that new port settings for how the ports are indexed for a device can be changed when adding a new device type.

---

## 6. Terminology/Definitions – User Documentation

The definitions given here apply for the use of the terms in this document, and may not apply for the terms used outside this document.

<b>ANSI</b>	American National Standards Institute; A coding scheme for representing string characters correctly; extended ASCII with somewhat support for ~2000 different characters.
<b>Cable</b>	Used to connect two devices together by the two ports the cable is connected to.
<b>CdbVis</b>	Configuration Database Visualizer; the name of the program this documentation is written for.
<b>Clickable</b>	Something that the user can click on, and that responds to the click.
<b>ConfDB</b>	LHCb Configuration Database, a database maintained in the LHCb online system.
<b>ConfDB Library</b>	The name of the library that is used by CdbVis to communicate with the ConfDB.
<b>Connected Link/Connection</b>	A link that is connected to two devices, one on each end point of the link.
<b>Data Object</b>	The device type, link type, device, link/connection and port objects have its own entity in the ConfDB, and the data (properties, attributes) that is needed to create/modify/delete one of those is encapsulated in a data object in the program.
<b>Device</b>	It is the different teams that work on each subsystem for the LHCb system to define what a device is in their subsystem. It is a sort of an independent (circuit) board with a well-defined task.
<b>Device Type</b>	A collection of devices of the same type that have some common properties and parameters.
<b>DLL</b>	Dynamic Link Library; a non-executable binary file with some common functionality that can be shared between many programs during runtime.
<b>Double-Click</b>	Two rapid clicks with the left mouse-button, where the event associated with the clicks is called on the second mouse button release.
<b>Environment Variable</b>	A variable set for the system (Operating System), that is used by programs that run in that system and need to know where to find (look for) executables or libraries it depends on.
<b>Free link</b>	A link that is not connected to any devices.
<b>Free port</b>	A port on a device where no link is connected to.

---

<b>GUI</b>	Graphical User Interface; that a program has a graphical program window and does not only run in the console.
<b>LHCb detector/system</b>	The whole detector and systems related to it (TFC, DAQ)
<b>Link</b>	The cable used to connect two ports on two devices together.
<b>Link Type</b>	A collection of links of the same type that have some common properties and parameters.
<b>Module</b>	A file with one or more related classes that must be included in another module to be visible for the classes in that module.
<b>Object</b>	Device type, link type, device, link and port are defined as objects; as they have their own entities in the ConfDB.
<b>Partially free link</b>	A link that is connected to a device in one of its ends.
<b>Port</b>	A part of a device where you can connect a link/cable to, so that data can be transferred to and/or from the device.
<b>Python</b>	The programming language CdbVis is written in; an interpreted, runtime compiling object oriented programming (script) language.
<b>Python Wrapper</b>	One or several files that make a code within a C++/C library (program) accessible for a Python module.
<b>Query</b>	To directly (use SQL) or indirectly (use a library that uses SQL) to create, modify or retrieve data in a database.
<b>RPM</b>	Red Hat Package Manager; easy to install/uninstall packages for a large range of Linux distributions.
<b>Single-Click</b>	A click with the left mouse-button, where the event associated with the click is called on mouse button release.
<b>SQL</b>	Structured Query Language; a computer language to create, modify and retrieve data from a database.
<b>Sub detector/subsystem</b>	A smaller part of the LHCb detector; f. ex. Muon, TFC, PS etc. (even though TFC is not really a part of the physical detector).
<b>Unicode</b>	A coding scheme for showing string characters correctly; a rather new technology that allows us to represent from 256 to more than a million characters, depending on the Unicode version.
<b>Visual Object</b>	A link or a device in the visual window; a link is drawn as a line and a device is drawn as a rectangle with lines sticking out of it on one or several sides of it.
<b>Widget</b>	A graphical control in the GUI program: frame, text box, check box, panel, window etc.
<b>wxPython</b>	The widget library used to be able to make a GUI interface of the program.



## 7. A developer's guide to CdbVis

The subchapters here are divided by the modules that are created for the program, which again is divided by the classes in the module. More detailed information/comments to be found in the modules themselves. This is not an introduction to Python nor wxPython, so I assume that the reader have knowledge (and experience) with those, where some Python or library specific is explained in detail.

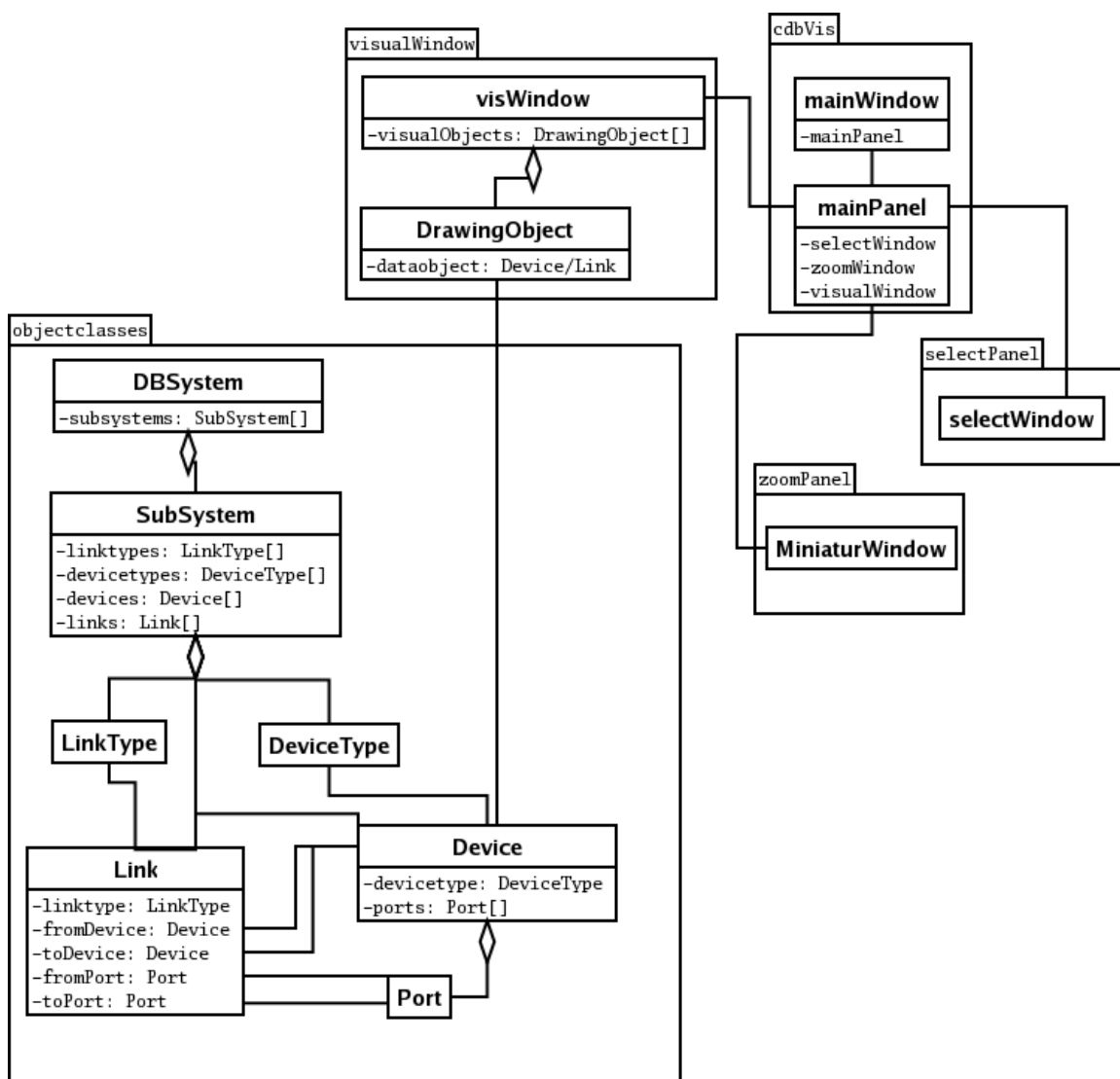


Figure 22 The class model above show the relations between the classes in the different modules (shown as packages), and the member variables that is responsible for the association/reference between the classes are shown.

## 7.1 cdbVis.py

### 7.1.1 Class mainWindow inherits wxFrame implements dbUpdate

#### 7.1.1.1 Inheritance and implementation

The mainWindow class is a GUI class, which means that it represents a part of what you see on the screen. It inherits the wxFrame interface (and methods), which represents a normal window you see on the screen, and it implements the dbUpdate interface. Since implementing interfaces is not really a part of Python it is "simulated" by inheriting the class (just to show that mainWindow implements specific versions of the methods defined in dbUpdate).

Visually, mainWindow is the big frame/window which you will consider as the whole program window.

#### 7.1.1.2 Constructor and initializing of member variables

We create the status bar, the tool bar and the menu bar for the main window.

##### *Status bar*

This is an enhanced version of the standard wxStatusBar(), since the standard one does not have support for other widgets than static text. Our enhanced version can have any type of widget associated to a column on the status bar.

##### *Tool bar*

We load \*.xpm bitmap images for the toolbar since this image type has in-built support in most common Operating Systems. We have two buttons to connect and disconnect to the ConfDB, since we discovered that it is not possible to change an image on a button after initialization on Linux.

##### *General*

A self.\_\_panel member variable of our class mainPanel is created. The different other GUI content is actually created in the mainPanel class. However, we still keep mainWindow as the main/top class, where the communication between the different classes (instances of them) takes place.

We have the self.\_\_cdb member variable to store the connection to the ConfDB in, and made it accessible through GetCdb() to the other classes.

The self.\_\_cfg stores the file object to the settings.cfg (or other another settings file that is being used).

We have self.systemlevels to know how many levels that are defined for each subsystem. You will have to change this if you add new levels to a subsystem. The default number is 1, which is Level 0 (LHCb system, not counted) and Level -1 (Device level).

self.\_\_active<XXX> is a variable to keep track of the current active object of a given type (device type, link type, system, device, link, port). The active object of a type is the one that is affected if we choose an operation for an object (modify, delete, duplicate etc).

##### *Important lists*

We have several lists to store information about objects to be stored in ConfDB:

*self.\_\_dirtyobjects*: is the list where we store all created/modified/deleted objects, and which we iterate through when we store changes in the ConfDB. See 7.1.1.3 *Dirty objects list*.

*self.\_\_renameobjects*: is the list where we store all renamed objects, and which we iterate through when we store changes in the ConfDB. See 7.1.1.3 *UpdateRenamedObject*.

In addition to *self.\_\_renameobjects* we have *self.\_\_oldnames* and *self.\_\_newnames* to store the names of renamed objects, so that we know when we f. ex. get a device from the ConfDB with a given name, whether it has been renamed to another name by the user (but not yet applied to the ConfDB), so that we do not treat the ConfDB device we retrieved as a different object than the renamed one. See 7.1.1.3 *UpdateRenamedObject*.

*self.\_\_undolist*: is a list with information about which objects that should be undone after an operation/action. The list is a list of lists, where each entry defines at which index in the *self.\_\_dirtyobjects* list the given operation started at, so that we can remove all objects after the given index if we want to undo the operation. In addition there is a description so that the user can see what he/she undoes. See 7.1.1.3 *How Undo Works*.

*self.\_\_visualundolist*: Where we store objects that have been undone, and that may be brought back to life when redoing. See 7.1.1.3 *How Undo Works*.

*self.\_\_redolist*: is the same as *self.\_\_undolist*, but for redoing objects that are in the *self.\_\_visualundolist*. See 7.1.1.3 *How Undo Works*.

*self.\_\_visualredolist*: Where we store objects that have been redone, and that may be brought back to life when undoing. See 7.1.1.3 *How Undo Works*.

### 7.1.1.3 Methods that need explanation

#### 7.1.1.3.1 How undo works

##### 7.1.1.3.1.1 Add new UNDO action

It is quite simple to add a new entry in the UNDO list. Before you create any new objects you call the function `GetFirstNewUndoIndex()`, which returns the index the next object that will be added to the dirty objects list will get. Then you will know that if you want to undo this operation, you will just tell the `DoUndo(...)` function that it should undo all objects created after this position in the list. After you called the `GetFirstNewUndoIndex()`, you can add the new objects to the dirty list. When you have successfully created the new objects, you call the function: `AddToUndo(integer_from_GetFirstNewUndoIndex(), "Description of what the user undoes if he chooses to undo")`. And voila, that is it.

Important; you should not add an UNDO entry if the objects are not created)

##### 7.1.1.3.1.2 UNDO/REDO an action

Undoing an action grabs the information piece from the last position of the undo list, iterates through the objects after the given UNDO position, and treats them the way they deserve:

If undoing a CREATE, the object is, if visible, removed from the tree view and the visual window.

If undoing a MODIFY, the object is, if visible, modified in the visual window, reversing the changes.

If undoing a DELETE, the object is, if needed, added to the tree view and to the visual window.

After it has iterated through all objects, the information about how to redo the undo action is added to the redo list, and objects backed up if needed.

For redo, it is just the opposite of undo, and undo information about the redo is added at the end of the undo list.

##### 7.1.1.3.2 Dirty objects list

The dirty objects list can be accessed from other modules/classes through some public available methods:

*AddToDirtyObjects(object,...)* adds a created/modified/deleted object to the dirty list as well as backing it up in the recovery file *recover.dat* in the *CdbVis* directory.

*RemoveFromDirtyObjects(object,...)* removes the last added instance of a given object from the dirty objects list.

*RemoveDirtyObjects(index\_from)* removes all objects from the dirty objects list after a given index

*GetDirtyObject(object,...)* returns the last added instance of a given object from the dirty objects list if found (if the last added instance of the given object was a delete of that object, we return as if we did not find the object).

*FindDirtyObject(object,...)* returns the last added instance of a given object from the dirty objects list if found, also if the last added instance was a delete of that object.

As already mentioned, the dirty objects list is set up reverse chronological. The first object in the dirty objects list is the oldest one, and the last object in the dirty objects list is the most recent one. And also, the "same" object, can be put several places in the list, but it is only the most recent "instance" that is the one we use and the one the user sees. So if we create an object at time X, we put it in the dirty objects list with the save status set to CREATE. Later at time X+1, we modify one or several properties of this object, and put it in the dirty objects list with the save status set to MODIFY. We may do several MODIFY after this as well, and for some objects even a DELETE as the last one. The last "instance" of an object in the dirty objects list contains the same unaltered data on the properties that has not been altered since the CREATE, but other properties may have been altered, and therefore different. Anyway, the last instance of an object in the dirty objects list will ALWAYS be up-to-date, with the most recent information for the object. So this explains also why UNDO/REDO works so well, once you remove the last instance of an object from the dirty objects list, it is the second last that will be the up-to-date one.

#### 7.1.1.3.3 Recover objects to/from file

Whenever we add a new object to the dirty object list or undo/redo this action, it is written to a file. This way the recovery file is like a history of what we have done in a session. The objects (lines) in the recovery file have information enough to recover an object, and it tells whether it was added because of UNDO, REDO or DESIGN (default).

We add all info from the recovery file to the dirty objects list on next start up if we did not store them in the ConfDB, and if the user answers yes to recover them from our message prompt.

#### 7.1.1.3.4 CleanUpDirtyObjects

This method figures out if there is anything to change in the ConfDB (create, modify or delete objects). For every delete object that is found in the dirty objects list, the methods looks for modify and create objects of the same object (type and name), and deletes/removed them from the list if they exist. Cause there is no reason to store changes in the database for an object that will be deleted anyway.

#### 7.1.1.3.5 OnSave

This method consists of several steps:

First we run the *CleanUpDirtyObjects(...)* method to see if there is anything to query to ConfDB, and to remove objects that will be deleted later on, as described under *CleanUpDirtyObjects* above.

Next we search through the dirty objects list and put instances of the same object together, f. ex. an object (given name and type) is modified 5 times, we only want to save this object with 1 modify query, and we can do that by only using the last instance of the object in the dirty objects list. Also, we put create and modify instances of the same object together, so that instead of 1 create and 5 modifies of a new object created in the given session, we only do the create, but with the most recent user-changed properties. As you can see, the objects can also have several modify statuses set, depending on which properties that have been changed, this is also taken into consideration so that 2 modify objects with different modify statuses does not overwrite each other.

Then we sort by the object type (device type, device, link etc.), save status and modify status, so that

objects that have the same ConfDB operations are sorted after each other in order, so that they can make use of database caching before they are committed to the ConfDB, which is faster than committing every single object/operation one at a time.

We then put the dirty objects and the rename objects in the same list (rename objects first), so that any objects that may have been renamed in our dirty objects list (and already stored in db), will be updated first, and modified/deleted correctly depending on the objects in the dirty objects list. Remember that all objects in the dirty objects list have the name after the last renaming, and are hence already updated, that is why the objects in the ConfDB must be updated before we iterate through the dirty objects list. Objects that are created in the dirty objects list hence have the most recent name, so no rename is needed for them.

Then it is time to iterate through the dirty objects (and rename objects). We have a [name, type, save status (modify status)] variable set for each object, to see if it is the last object of its type (with the given operation to be committed to ConfDB). If it is the last object of its type, we do a commit for the object (and the previous objects of the same type) in the ConfDB. We must also set whether an object is the first and/or last in a (possible) sequence of same types of objects. We save the object in every iteration, either in cache, or a commit. We do usually not get any errors before a commit, but with a commit there is a constraint check etc. for the given object/row to be stored in the ConfDB, and if it fails, all objects of that type fails. If that happens, we put them all in a recovery file, so that the user can try to save them later, after one or more modifications.

When we are finished iterating, we put all the objects we put in the recovery file earlier, back to the dirty objects list, and inform the user about the success/failure.

#### 7.1.1.3.6 Progress bar (Progress data and IdleHandler)

The progress bar (gauge) showing the relative time left of an activity/operation/process can easily be added for any operation. Before starting an operation, we must know how many iterations or parts (number of divisions of the progress bar), we will use for the operation, the more divisions, the more closely we can follow the progress of the operation, depending how long time each operation takes. There should perhaps be a visible change quite frequently, so that the user does not suspect the program to have stopped responding.

The method is defined as followed: `ProgressData(range, finished, step)`. When initializing the progress bar we set the range as the number of operations we want to divide the progress bar into. Finished is set to false, and step to 0 as we do not do any steps before we start the operations. Thus:

`ProgressData(length_of_operation, False, 0)`. After each operation have finished, we want to update the progress bar, we do that as follows: `ProgressData(0, False, 1)`, cause the range (already set) is ignored, and we are not finished yet, but we want the progress bar to increase 1 step. When all operations are done, we want to reset the progress bar, and do like this: `ProgressData(0, True, 0)`.

It is the `IdleHandler(...)` that is responsible for updating the progress bar GUI, and this happens whenever we call `wxYield(...)` to get some idle time from the CPU to take care of the progress bar. If we do not call `wxYield(...)` but only try to update the progress bar from the `ProgressData(...)` method, it will not be updated as the CPU will not let us spend any time to refresh the progress bar widget.

#### 7.1.1.3.7 ShowError

Every potential error (at least what I hope and tried to) in the code of the whole program (all modules), will run the `ShowError(...)` if it occurs. The informative messages to the user is also sent to this method, and every message, both information and error, is stored in a log file with the date and time for the start of this program session in the logs directory.

As the errors are not showed to the user with a new message box at every place in the code where an error can occur, but are showed with a message box or in the status bar (column 1 and 2) in the `ShowError(...)`, this can easily be changed to some other "error handling", and perhaps to except informative messages to be logged.

#### 7.1.1.3.8 UpdateRenamedObject

Whenever we rename an object in a modify operation, all previous instances of this object in the dirty objects list is renamed to the new name. If the object has been renamed earlier, and the object already exists in the ConfDB, a rename object exists in the renamed objects list, and must be changed with the new one. If it was not already in the renamed objects list we check whether the object exists in the ConfDB or not, if yes we put the rename object to the rename objects list. We have some rules for handling the renaming for the case when we have just renamed an object and what actions do we need to do:

<i>In dirty objects ?</i>	<i>In rename objects?</i>	<i>In ConfDB ?</i>	<i>Action</i>
yes	yes	yes	If the object was in the rename objects list, it implicates that it is also already stored in ConfDB. In this case we change names on all instances in the dirty objects list of the object to the new name, and we add a new object to the rename objects list with the previous name set to the previous name of the object that was already there (the name it has in ConfDB), and the name to the new name we renamed it to.
yes	no	no	We change the name of the previous instances of the object in the dirty objects list to the new one we renamed the object to.
yes	no	yes	We change the name of the previous instances of the object in the dirty objects list to the new one we renamed the object to, and we put a new rename object to the rename objects list with the previous name set to the name of the object before the we renamed it, and the name to the new name we renamed it to.

## 7.1.2 Class mainPanel inherits wxPanel implements dbUpdate

### 7.1.2.1 Inheritance and implementation

The class is a GUI class, which means that it represents a part of what you see on the screen. It inherits the wxPanel class, which represents a normal panel, which is the area inside a window, and it implements the dbUpdate interface.

Visually, mainPanel is the area in the mainWindow between the tool bar and the status bar.

### 7.1.2.2 Constructor and member variables

In this class we set up the two other main GUIs in the CdbVis: the visual window and the selection window. The two windows are split with a draggable sash, which is set up using SashLayoutWindow in this class. The miniature area window is also created in this class.

## 7.1.3 Class cdbVis inherits wxApp

### 7.1.3.1 Inheritance and implementation

This is the application class, which sets up the mainWindow instance to be shown as the top-most window.

## 7.2 cdbVisCore.py

This is the module where we put all the constants (or at least where all the constants should be).

### 7.2.1 Class dbUpdate

This is the interface class (or abstract class) which implements some methods for use in several other classes (those that inherit this class). It is the OnDbUpdate(...) which execute commands if database state changed and DbUpdate(...) that is used to inform other classes about the update.

## 7.3 objectclasses.py

This is the module where we have all the data objects (Subsystem, DeviceType, LinkType, Device, Link and Port), and the only module where we directly communicate with the ConfDB.

We have 3 global variables configfile, appmain and cfDB, and 2 global methods, GetConfig() and GetAppWin(). We need to set those variables to have access to the Configuration File, the Communication Object (mainWindow) and to the Configuration Database independent of object creation. Those global variables are initialized at startup and connection to ConfDB.

### 7.3.1 Class DBInfo implements dbUpdate

This is an abstract class with some methods that are implemented in every DBObject class (DeviceType, Device, LinkType, Link and Port). We also describe the common functionality for these classes here.

#### 7.3.1.1 Handling object properties

Most of the properties for an object are contained in a dictionary (a dictionary is a python hash array), this is all the properties that we need to have accessible when copying objects etc. We then simply use a function called SetDict() to copy every variable/property of an object to another.

Using a dictionary to store the properties makes many tasks easier. We do not have to know the name of the different variables, but we set a string constant for every property in the dictionary. The string constant is the name of the property in the ConfDB (if it exists in ConfDB, if not we just give it a name), so if the property (attribute) changes name in the ConfDB we also have to change name on the string constant in the CdbVisCore module.

Filling the dictionary with the data from the database is hence done quite simply in a for-loop, as we get both the attribute name and value from the ConfDB.

#### 7.3.1.2 Constructor

When we create an object, we fill in the unique properties in the constructor for the object in question, and also set the new-parameter to False or True. If new is set to False, we consider the object to already exist either in the ConfDB and/or pending list of dirty objects to be stored in the ConfDB. We will therefore call the

Update(...) function to fill the object's member variables with the values valid for the given object. If new is set to True, we consider the object to being created, and we can therefore not get information from previous instances of this object.

### 7.3.1.3 Set methods

Most of the set methods for the data objects contain a call to a validation function, to validate the data set to the object. If validation fails, a ValidationException exception is returned to the place in the code which called the set-method, and the validation exception must therefore be handled there. The set methods are only used when creating new objects, and it is hence quite easy to have one place to catch a Validation exception for all the set-methods we use, and handled there (instead of n IF structures).

### 7.3.1.4 Error handling

All data object classes that retrieves information from the database is designed to set a variable in its own class (self.\_\_errorMessage) if an error occurs, so that the error message can be retrieved by calling object.GetErrorMessage() where the object is created, to get error information. An error can be triggered if:

An error occurs in the ConfDB library, and it returns a Runtime Exception. An error may occur if a commit failed to store all info in the database, a select did not get any rows from the database etc.

A validation error occurs in the respective object. All variables that are set for an object is validated to see if it is a valid value, if not a ValidationException exception is thrown.

Some other exception error is thrown as a result of some error occurs.

All errors is caught at the end of a method, and returns False (if a real error) or in most data fetch cases an empty list [], if no rows were retrieved. The result of a method call is then checked where the method was called from, and action taken.

### 7.3.1.5 Create/Modify/Delete

When we have done something with an object (created, modified or deleted) we call the Create/Modify/Delete method for the respective object, which puts a reference to the object in question to the dirty objects list as well as putting the corresponding save status (flag) to it.

### 7.3.1.6 Save

When iterating through the dirty objects list in mainWindow class, we call the Save(...) method for every object, to do a create/update/delete query to the database. The Save(...) method has 3 Boolean parameters; first, many and commit. Those 3 parameters define which ConfDB Library function to execute for a given object. For most objects and operations in the ConfDB, two different functions are defined for an object: a function that immediately commits the object to the database and a function that puts the object in the cache and commits to the ConfDB when the last object of the given type with the same save status (and modify status) is called. If first=True and last=True (implies that many=False), we immediately commit the object to the database. If first=True and last=False (implies that many=True), we initializes cache, and puts the object there. The next object may have the values first=False, last=False (many=True), which adds the next to the cache. And then later first=False, last=True (many=True) commits all objects to the database.

### 7.3.1.7 GetObjectInfo

This method returns the dictionary of object properties to the calling code. This dictionary will be found in the most recent instance of the object in question. So if the instance is newly created, it returns its own dictionary, also if it is recently updated with data from the dirty objects list or the ConfDB. If none of the cases above is true, the instance will call the Update(...) method.

### 7.3.1.8 Update



This method returns the most recent dictionary (property values) for a given object. If we are working in creation mode, it will first look in the dirty objects list, if it is not found there we check in the ConfDB. If we are in navigation mode we do not look in the dirty objects list (because it is not used and therefore empty), but go straight ahead to look for the object in the ConfDB.

### 7.3.2 Class DBSystem implements dbUpdate

This is the main ConfDB class; it is the one responsible for connecting and disconnecting to ConfDB, to keep the connection in memory, to open and read some of the settings in the configuration file and to retrieve subsystems available.

### 7.3.3 Class SubSystem inherits DBInfo

This is the class for keeping information about a subsystem such as Muon, Velo, PS etc. We can create an object of this class to retrieve all device types or link types in a subsystem.

### 7.3.4 Class DeviceType inherits DBInfo

This is the class for keeping information about a device type such as M5R4, Tell1, FEE etc. We can create an instance of an object of this class to retrieve all devices of the given device type

### 7.3.5 Class Device inherits DBInfo

This is the class for keeping information about a device such as M5R4#2, Tell1\_0010 etc. We can create an instance of an object of this class to retrieve information about:

- Are all ports of this device free? (True if there are no connections to this device).
- Get free ports.
- Get the links that are connected to this device.
- Get the PortIDs of the ports on this device.
- Get the port objects of the ports on this device.
- Get the paths this device is a part of.
- Get the specific path info for a given path.

### 7.3.5 Class Link inherits DBInfo

This is the class for keeping information about a link with a given link ID (set in the ConfDB). We can create an instance of an object of this class to retrieve all information about a link.

### 7.3.6 Class LinkType inherits DBInfo

This is the class for keeping information about a link type such as Data link, Gas, HV, LV etc. We can create an instance of an object of this class to retrieve all information about a link type.

### 7.3.7 Class Port inherits DBInfo

This is the class for keeping information about a port with a given port ID (combination of device name, port number, port type and port way). We can create an instance of an object of this class to retrieve all information about a port.

## 7.4 visWindow.py

This is the module containing the classes that are responsible for the visual display in the visual window.

### 7.4.1 Class visWindow inherits wxScrolledWindow implements dbUpdate

#### 7.4.1.1 Inheritance and implementation

This class defines the GUI window/frame with white background located to the right on the mainPanel. It is where all the visual objects are drawn. It inherits the wxScrolledWindow which is a GUI class with automatically managed scrollbars and a DC (Device Context) associated with it, in which you can draw on.

#### 7.4.1.2 Constructor and member variables

If the window needs a refresh (caused by another window hiding it, and then refocused; the operating system is responsible for this), the paint event is called, which is set to invoke the OnPaintEvent() method. If the mouse cursor is moved, clicked, dragged etc. in the visual window, the OnMouseEvent() is invoked (except for double-click and right-click where we have to other methods to handles those). Key presses are handled in OnKeyEvent().

In level -1 (Device level): The self.contents list contains references to the visual objects that are shown in the visual window, and self.selection list contains references to the visual objects in self.contents (and the visual window) that are selected.

For the other levels except level 0, we have the self.macroobjects that stores the references to the drawn virtual objects in the visual window.

#### 7.4.1.3 Logic

##### 7.4.1.3.1 Zooming

We have several methods that indirectly control the zoom. We can set the zoom in methods in mainWindow class, which calls Zoom(...) in visWindow and redraws the content of the window with the new zoom factor. We have a global zoom method and variable, so that the current zoom value is accessible for instances of DrawingObject class.

The Zoom() method itself does not actually redraw the visual window contents itself, but iterates through the visual objects in the self.contents list and changes the size and position according to the new zoom value. The new size and position is only set for the nodes (not the links), and is calculated by comparing the previous zoom to the original size of the object (when zoom is 100%, default), and then the change the zoom to the new zoom value. The reason why the size and position to the links is not calculated is because they are fixed/stuck to the nodes, and has no "own" size or position, because this depends on where the nodes are. The position of the nodes is calculated by the distance from the upper left corner of the node to the center of the visual window (the center of the whole visual window, also the parts that you have to scroll to see, so the position is independent of the scrolling). If the previous zoom was 40% and the new zoom is 60%, we have a 0.6/0.4 increase of the size and distance from the center of the node.

In general, the size of a node is calculated by the number of input and output ports, but maximum and minimum widths are set.

At the end of the Zoom(...) method we force the paint event to redraw the visual window if needed.

#### 7.4.1.3.2 AddToContents

This method is related to the DoUndo(...) and DoRedo(...) methods in mainWindow, but may be used in other cases as well. It simply adds a node (or link) to the visual window and sets up the connections, if the links are already in the visual window, as defined in the port list of the node.

#### 7.4.1.3.3 AddNode

This method adds a node to the visual window. There are 2 different layout algorithms defined in here, one for path view and another for the rest (digression: the layout should be handled in a separate layout manager, if possible). The path view layout algorithm is quite simple, as each node is put beneath each other in the path. The other layout algorithm however does take a lot of variables and properties into account, and it does not work perfectly well under the current circumstances. The factors it takes into account are: suggested position, the position of its parent and total number of ports on the given side for its parent, and we check whether we have other nodes in the same Y layer (same horizontal positioning). This layout algorithm has to be changed as it does not really play well with the new changes in the visual window.

#### 7.4.1.3.4 OnMouseEvent

This method is, as you can see, divided in several Python switch-cases (read: if structures), one for each level and subsystem (and of course the two general ones for the level 0 (LHCb system level) and level -1 (device level)).

#### 7.4.1.3.5 Level 0 -> n

In level 0 we have created an image map, where we have set the coordinates and sizes defining the different subsystem (click) areas on the image, and stored it in a local list. We check the position of the mouse cursor, and change the cursor to a hand when moving over a clickable system. If we click we call the SetLevelSelections(...) to update the visWindow list variable with what we clicked on, and we change the level zoom to level 1, so that it can be handled in the OnPaintEvent(...) method.

It works similar in the other levels as well, but here we have created instances of the DrawingObject class of the type MACRO\_OBJECT, where some of them are clickable and some are not. All the instances of MACRO\_OBJECT that are created in the visual window in the current level are stored in the self.macroobjects list, so each time we move the cursor or click, we check whether there is a clickable MACRO\_OBJECT on the mouse cursor position. If the object under the mouse cursor is clickable we change the cursor on mouse move or change level on mouse click.

#### 7.4.1.3.6 Level -1

We detect mouse clicks and mouse moves in combinations and where on a visual object the user clicked, to figure out what the user wants. We then trigger one out of several actions: object selection, object resize, object move, multiple selections with Ctrl-key or selection area.

#### 7.4.1.3.7 ShowNeighbourLinks

If we are in dynamic link mode, a double-click on a node will result in either an expand or collapse of links to its neighbors (if any). In the code we have 3 different states for a dynamic linked node; COLLAPSED, EXPANDED and PARTIALLY COLLAPSED/EXPANDED. A node is collapsed if none of the links and the neighbors is shown. A node is expanded if all of the links and the neighbors are shown (given that the link type of the links is of a link type that the user has set to be shown). A node is partially collapsed or partially expanded if we have changed from another view (f. ex. neighbor view, path view) where only some of the links/neighbors are shown for a node.

We have defined that there can be many links between two neighbors, and a collapse/expand will hence hide/show all of the links.

#### 7.4.1.3.8 OnPaintEvent

As already mentioned, this method is responsible for drawing the contents to the visual window. It iterates through the visual objects in the self.contents list, and draws every each of them calling their own Draw(...) method, because it is the DrawingObject itself that stores the drawing information about itself.

It works quite similar for all levels, as all objects we create are instances of DrawingObject, except for the illustration in level 0 of the LHCb system which is an external PNG image file.

#### 7.4.1.3.9 DoDuplicate(Helper) and DoClone(Helper)

First of all, the helper methods are only some methods to have more control of the duplication and cloning. The DoDuplicate(...) and DoClone(...) are the main methods, which do all (most) of the work.

The differences between duplication and cloning already have been described earlier in this document. In the duplication helper method we do a check to see if anything is selected, and ask the user how many duplications he/she wants of the selected object(s). We then call the duplication method. We check whether all the links and the devices they are connected to have been selected (if any); because if a link is selected for duplication also the devices on each end of this link have to be selected for duplication as well.

Next we duplicate; we create new visual objects of what we selected and copy the data of the data objects to new data objects that we reference to in the visual objects, and we also give the new objects new names. Of devices we also copy the Port objects to new port objects, which we assign to the duplicated devices.

The clone helper also asks the user how many cloned structures he/she wants, and we do some checks before we go to the cloning machine (the core of the cloning method). The clone main method actually makes use of the duplication main method to duplicate the visual objects that are selected, before we do some magic with the duplicated object(s), which only the clone method can do (described earlier).

The checks we do in the clone main method are:

Do we have enough free ports available on the given devices to clone a given link?

Do we have any more ports of the same port type as the original one to clone a given link to?

If the first one fails, we stop and notify the user about the error. If the second fails, we try to continue, and the user will see if not all links were connected as he/she hoped, and can undo the action or fix the rest him/her-self manually.

#### 7.4.1.3.10 *CreateNode, CreateLink*

These are the methods that actually create the visual object in the visual window, node and link respectively.

#### 7.4.1.3.11 *ResizeToFitContents()*

This is the method that actually controls and does the work for auto zoom. It iterates through all the device/node objects in the visual window (in self.contents list), and checks whether it is outside the visible visual window. If it is, it calculates the zoom needed to make the whole objects fall within the visible visual window, but only if it is more far away than every other device objects we have checked so far.

We calculate the zoom like this:

We figure out the shortest and longest distance from the center of the window to the edges. The shortest are up and down from the center, and the longest is the half-diagonals. We also calculate the angle of the diagonal, we already know the angle for the shortest distance (0).

We also find out the start x and y for the visible visual window (upper left), and end x and y (bottom right). Then we iterate through the contents

For each object, we check whether it is outside start x and y, and end x and y, if it is, we need to find the distance.

For each object, we find the total distance from the center x and y, to the corner of the object that is most far away.

We then find the angle to the object from the center, and check this against the angle of the diagonal. If it is bigger, the object is more far away on one of the other sides, and we stop calculating.

If the angle is less than the diagonal, it is between 0 and the angle of the diagonal. We then find the distance from the center of the visible window to the edge of the visible window in the angle to the given object, and we divide this distance on the total distance to the object. Then we figure out how much of the distance to the object that we have within the visual window, and how much more we need.

If the relation was bigger than for any other object, this is the object that is most far away, that we have found so far, and we calculate the zoom needed, which is 100.0 divided by the distance\_factor.

When we have iterated through all the devices, we have found the greatest zoom factor needed, and we use this zoom factor to zoom in on the contents.

## 7.4.2 Class DrawingObject

This class defines the data/properties and drawing methods for a visual object in visWindow. Each visual object in visWindow is an instance of this class. The DrawingObject instances also have a reference to a data object (instance of either the Link or Device class in objectclasses.py) with the data set by the user.

### 7.4.2.1 Logic

We have 3 different types of DrawingObject (not subclasses, only a type variable defined in the class).

#### 7.4.2.1.1 *The NODE type*

Each node has a 2 dimensional port list, 1 list for ingoing ports and another one for outgoing ports. Both lists are filled with entries of None (NULL) when the DrawingObject is created, because each index in each list corresponds to a fixed port on a node. If a link is connected to a node, we set a reference to the DrawingObject instance of the link on the index in the port list which corresponds to the port the link is connected to. This index is "calculated" specific for every device type or we use a general one that works for general devices. The general one is defined as follows:

0 1 2 3 4 5 6 7 8 9 10

|---|---|---|---| IN

| NODE |

|---|---|---|---| OUT

0 1 2 3 4 5 6 7 8 9 10

The port number starts with index 0 in the bottom left corner, and continues in the top left corner with the next row. Ports are only on bottom or top side of the node (not on the right and left sides). If a device does not follow this port number system, you will have to define the port offsets to the index in the port lists yourself. This will probably be possible to set in the configuration file, but is so far only hard coded. FindCorrespondingPortIndex(...) is the method that takes care of the port number, port type, port way -> index in port lists conversion, and the place where it is so far only hard coded for M5R4 chamber in the Muon system. The general one can be changed to whatever that is the most common way of setting port numbers to a device.

Given the index of a port in the port list, we can find the position of that port with the GetPortPosition(...) method. This method must also be changed if new port numbering schemes for other device types is added to FindCorrespondingPortIndex(...).

## 7.5 selectPanel.py

First of all, this module contains the class that takes up the space to the left in the mainPanel. It also contains the class that shows and makes it possible to choose paths for a device in a pop-up window.

### 7.5.1 Class selectWindow inherits wxPanel, wxColumnSorterMixin implements dbUpdate

This class is most of all a GUI class, and is the panel with the tree view widget, a combo box widget and a list control widget. It inherits from a class called wxColumnSorterMixin, but this is actually to make the list items of the list control widget to be sorted alphabetically.

#### 7.5.1.1 Tree view Widget

When we do something with a node in the tree view widget (add, remove), we need to know the tree node IDs of the affected tree nodes. Because of this we keep track of all tree node IDs as we add them to the tree view in the member variable self.\_\_treeitemids {}. (A dictionary with tree node (name: tree node id) relations). The tree node name is the text a tree node has in the tree view, and the tree node ID is a unique ID that is generated and we get in return when we call a method to create a tree node in the tree view.

We also register a node type with each node that we add, and that is predefined constants telling us whether it is a device, link, device type, link type etc. When an event is then triggered for the tree view, we can get the type of the tree node, and call the right method (We have a method for each tree node type). Most of those functions call UpdateInfo(...) with the data object as parameter. In UpdateInfo(...) we call dataobject.GetObjectInfo() which is the same for every data object, to retrieve the dictionary of properties. We then iterate through the dictionary and add each property and value to the list control.

#### 7.5.1.2 Path

This is the class responsible for the GUI window that pops-up when you choose to view a device in path view. The Path window is created when calling UpdatePath(...) if it was not shown already. The PathSelect class is a class inside the selectWindow class and defines the path variables and methods, and the PathSelect class inherits from wxMiniFrame(), a frame/window with a small title bar, and shown as a pop-up window when path view is selected for a device. One of the parameters to the constructor is a callback function that is to be called when a path is selected in the path pop-up window.

## 7.6 areaview.py

### 7.6.1 Class sysWin inherits wxMiniFrame implements dbUpdate

This class defines the GUI window/frame that is shown if "View miniature window" is checked on the View menu. It is a window that shows all the content and where it is positioned in the visual window.

#### 7.6.1.1 Class systemWindow inherits wxPanel implements dbUpdate

This is the panel inside the window, which we draw the graphics on (more specific on the DC associated with the window). We redraw the contents on this panel whenever the PaintEvent is called for the visWindow object.

Whenever we click and hold down the mouse button while the mouse cursor is inside the red rectangle (illustrating the visible part of the visual window), we can drag the rectangle around to select other areas of the visual window to be visible. This is handled in OnMouseMove(...)

In DrawSystem(...) we get the size of the whole visual window and compare it to the size of systemWindow to set the size of the red rectangle showing the visible part of the visual window after getting the size and position of the visible visual window from visWindow. A good and useful update to this class would be the possibility to drag out a selection around some objects and make the selection exactly fit inside the visible part of the visual window (with use of zoom). We also retrieve all objects from the self.contents list in visWindow, and draw them in this window with the same relative positioning as in the visual window, but as yellow rectangles. We set the minimum height and width to some fixed size, so that we can see where the objects are positioned in the visual window no matter what the zoom factor is. The window is redrawn on resize, to make the relative positioning of the objects to the window size be the same.

The RectangleArea class is a class for the red rectangle in the systemWindow class. It makes it easier to update the properties of the rectangle.

## 7.7 createdevice.py

### 7.7.1 Class CreateDeviceWindow inherits wxDialog

This is the only class in this module, and defines the dialog window shown when we create or modify a device.

We have a variable defining whether we create or modify a device. If we create we may set some predefined properties in the constructor, where some of them are set from the parameters set when creating an instance of this class. If we modify a device we send an object of the given (active) device in the constructor parameter list, and set all input areas in the window to contain the previously set values for that device.

The layout of the window differs a bit between create and modify as well. When creating a device we can set prefix, padding, number of devices and start number, and when modifying they are replaced by device id and device name.

When we click on the ok button we perform some checks to see: if the required data fields have a value, if the user created the ports for the device(s) and that the number of ports corresponds to the number of ports set for the device type of the device, that the prefix is valid etc. Then we call the SaveObjects(...) which creates a device object for each device we want to create, assigns ports to it and sets its properties. Any validation error will be caught, the processing interrupted and reset and an error message shown to the user.

As already mentioned, but worth to stress, the user has to create the correct number of in – and out ports for a device when it is being created.

#### 7.7.1.1 Our device name guessing algorithm

When we choose a device type for a device, and there are already created devices of that device type, we try to find the prefix for the given device type. If no devices are yet created for the device type the user will have to choose a prefix. If the algorithm fails to find the correct prefix, either by informing the user that it failed or the user sees that it failed, the user can change the prefix the way he/she wants to have it.

At the moment, the user has to provide a '%d' in the prefix, even though the device name does not contain a number. This should probably be changed (TODO). However, if the user chooses to rename the device afterwards, he/she can remove the number given to the device.

The algorithm is quite primitive, but will work in many cases. At this moment it is unsure how the different subsystem devices will be named (for me), but I think it will be something like: SUBSYSTEMNAME\_DEVICETYPE\_NAME\_XX where XX is a number with or without padding. I assume that the number will be set at the end of the string, and that it is a string character in the position before the number (a character that cannot be converted to integer).

The algorithm works therefore like this:

A device name is found for the device type chosen, and we find the length of the string to be n.

Take the character at position n and see if it can be converted to integer.

If the character can be converted to integer, put it at the beginning of a temporary string, set n to n – 1, and go to 2.

If the character cannot be converted to an integer, and our temporary integer string is of length 0, set n to n – 1 and go to 2.

If the character cannot be converted to an integer, and our temporary integer string is of length > 0, break the loop, continue to 6.

Find the number of zeros padded at the beginning of the number, and set the padding.

Find the devices for the device type chosen, sort them first by string length (the greatest number so far has the longest string), and then alphabetically. Find the highest number for a device, and set the start number to one higher.

It is the same algorithm that is used when we duplicate and clone devices, to automatically assign a name to it, but with one variation; then we have to set a device name for a device, and we use a temporary name if we do not succeed to find a prefix.



## 7.8 createdevicetype.py

### 7.8.1 Class CreateDeviceTypeWindow inherits wxDialog

Is the only class in this module, and defines the dialog window shown when we create or modify a device type.

Nothing in particular to outline here, but you can see that a rename object is created and put in the list of renamed objects in mainWindow if a device type is renamed.

## 7.9 createlink.py

### 7.9.1 Class CreateLinkWindow inherits wxDialog

Is the only class in this module, and defines the dialog window shown when we create or modify a link/connection.

## 7.10 createlinktype.py

### 7.10.1 Class CreateLinkTypeWindow inherits wxDialog

Is the only class in this module, and defines the dialog window shown when we create or modify a linktype.

## 7.11 createports.py

### 7.11.1 Class CreatePortsWindow inherits wxDialog

Defines the dialog window shown when we create or modify one or several ports. It makes use of a customized DataGrid class defined first in the module: PortDataGrid inherits wxGrid.

The combinations of port number, port type and port way to create several ports in one go is described in the user documentation.

If the user modifies one or several ports for a device, those will be marked as modified (is actually a cell in the DataGrid). Since we have 3 different modify statuses for a port object (because there are 3 different

ConfDB library functions for update, depending on what properties that was updated), the modify status of a port is a string (also visible in a cell in the DataGrid), consisting of the different modify statuses that will be set for a port. Every time we change a port, we add the numbers for what modification status it got. A port can have from 1 to 3 different modification statuses if it is modified, but the string of modification statuses can be even longer, as we just add new modification statuses to the string if we modify the port. When we then create port objects of the ports to add them to the dirty objects list, we create one port object for each modification status that is set, by checking the different modification statuses set in the modification flag property of the port.

## 7.12 EnhancedStatusBar.py

### 7.12.1 Class EnhancedStatusBarItem inherits object

Every column in the status bar is an object of this class. The widget and position (status bar column) of it in the status bar is saved in this object.

### 7.12.2 Class EnhancedStatusBar inherits wxStatusBar

This class is an enhanced class of the standard status bar class. The difference is that this status bar class supports widgets on the status bar. However, the widgets are not always positioned correctly in every window manager/operating system (but this is just a visual minor negligible problem).

## 7.13 helper.py

This is just a module with some helper functions used in several other modules to:

Flatten a list (make a multi-dimensional list to a one-dimensional list)

Compare two data objects (obsolete, have to be changed in order to work for the new versions of the data objects)

### 7.13.1 Class TextDropTarget inherits wxTextDropTarget

Is the class we use to handle the dragged and dropped targets in cdbVis. So far it is only used for dragging a device from the tool bar to the visual window, when creating a device. This class can however be used for any drag and drop operations, but the data type of the object that is dragged and dropped must be specified (this is a string defined by you). The different data objects is then handled in the OnDropText(...) method.

In order to set new drag and drop source and targets, the widgets involved have to be set to source and targets respectively. You can see the code for the drag and drop device in mainWindow (OnDragInit(...) method which is called when the user left-clicks on the graphics button of the device in mainWindow tool bar, to initialize the drag.). The visWindow is set up as a drop target in the mainPanel constructor. Thus, to sum it all up:

If we click with the left mouse button on the graphics device button in the tool bar we call the method `OnDragInit(...)`. `OnDragInit(...)` gets the name of the device type from the device type combo box, sets this value in a `PyTextDataObject`, set the dragging widget to be the graphics device button and assigns the `PyTextDataObject` to the widget we are dragging. We have already set the `visWindow` object to a valid drop target (in the constructor of `mainPanel`), so when we drop the widget in `visWindow` the `OnDropText(...)` method is invoked for the `PyTextDataObject`. In this method we create an object for the `CreateDeviceWindow` etc.

## 7.14 logwindow.py

### 7.14.1 Class `LogWindow` inherits `wxDialog`

This is the class that defines the log message window that is shown if we want to view all logged messages (informative and errors) for the user. The messages have already been stored to the log file defined for the given session, so this file is opened and all messages is retrieved and displayed to the user in a grid window.

## 7.15 myexceptions.py

In this module we define each class of the custom exceptions; database exceptions and validation exceptions.

## 7.16 validation.py

This is the module where we defined the different validation functions which are included in `objectclasses.py` module. We have:

`ValidateString`, which can check if the string is following the rules for maximum length, minimum length and a regular expression mask to follow if needed.

`ValidateNumber`, which can check if the number is a number and that it is following the rules for minimum value, maximum value and whether a floating value is allowed.

`ValidateSystem`, which can check if the subsystem name(s) specified is a valid subsystem name or a list of valid subsystem names.

If validation fails, a validation exception is raised.

## 7.17 Other files in the `CdbVis` directory

*appMain.py*: Was a python module I started to work on to make it possible to have several instances of the `cdbVis` running in a window at the same time, with a tab for each instance. It is possible to add this change

to the program, but since the menu bar, status bar and the tool bar is set to be the same for every instance, there is some coding needed to change these widgets for every instance. I started to work on this module because Beat Jost expressed a wish that he wanted to have this functionality. However, it probably not needed, as the user can start a new instance of CdbVis if he wants to, and have several run at the same time. There is however a limitation to 25 concurrent connections to the database (...).

*backup.xml, xml\_parse.py and xml\_write.py*: A test xml file and python test files to learn how to parse (read) and write xml files with the xml module in Python. Maybe needed for later use? (mass insertion of objects from xml files, as well as text files).

*createtables.py*: Just a python file to drop and create all tables in the ConfDB.

*datagridsave.py*: Once upon a time there was a plan to show the user what operations that was pending to be executed (sql queries) for the user before the data objects were created/updated/deleted in the ConfDB. Probably not necessary.

*layoutmanager.py*: Wanted to write a general layout manager for the objects in the visual window, but implementation halted. It's probably the best idea to put the layout information in one place, but it's a bit difficult to find a proper and good layout algorithm for the visual objects.

*recover.dat and recover\_tmp.dat*: Files used for saving objects to files once they are altered in CdbVis, in case of program crash etc, so that the objects can be restored on next startup. It is in a human-readable format (name of property followed by value), one object on each line.

*Logs/*: This directory contains all the logs for CdbVis. A new log file is created every time CdbVis runs.

*settings.cfg*: The configurations file for CdbVis which uses an \*.ini file format.

## 7.16 How to customize the tool for your use

### 7.16.1 Add a new subsystem

In the file *cdbVisCore.py* (the file with all the constants), there is a variable called *allvalidsystems* which is a list with the names of all the valid subsystems. You will have to add the name of your subsystem in this list.

At the bottom of the constructor of the *mainWindow* class in *cdbVis.py* there is a variable called *self.systemlevels*. This variable is there to control the number of zoom-levels in each subsystem. When you first add a subsystem, there is only one zoomlevel; the device level (which is the same for every subsystem). The default to add here is then: *"newsystemname":1*.

In *visWindow.py* you will have to alter the illustration of the LHCb detector. This illustration is a PNG image, and you can add new boxes etc. for new subsystems using an image drawing program. Next you will have to define the region of the box that you added to the illustration, so that it can be known when the mouse cursor moves over it. In the *OnMouseEvent(...)* function in the *visWindow* module the coordinates are given under *if (self.levelzoom == 0) and ...* in the list called *w\_and\_h*. Here you see all the coordinates and information for the already added subsystems as lists in a list. You will have to add a new (sub)list to this list with the given format: *[width\_of\_subsystem, height\_of\_subsystem, x\_position\_from\_the\_left\_of\_the\_visual\_window, y\_position\_from\_the\_center\_of\_the\_visual\_window, macro\_id\_for\_the\_subsystem\_if\_several\_with\_same\_name, name\_of\_subsystem\_as\_given\_in\_allvalidsystems\_list\_in\_cdbVisCore]*. You can see from the coordinates given for the other subsystems where your

subsystem is located in the visual window. If you have problems, draw a rectangle in the given position and given size in the OnPaintEvent(...) method to aim.

## 7.16.2 Do not show uninterested subsystems in the tree view

There are a lot of subsystems available in the ConfDB, at the moment 17 in total. This takes up a lot of space in the tree view control in the program, especially if one are not interested in seeing all those subsystems that oneself do not know much about or is not interested in seeing them. You can then do the following: Open the cdbVisCore.py file and find the variable named *do\_not\_show\_systems*, which is a list. Here you just add a string for the subsystemname as given in *allvalidsystems* that you do not want to see there (or remove from the list if you want to see it). This will not prevent you from creating devices in the given subsystem, and it is still accessible through the LHCb detector illustration view.

## 7.16.3 Adding new zoom levels for a subsystem

As you at least have seen for the Muon subsystem, you can click on any of the Muon stations on the LHCb detector illustration on start-up, and navigate to the next level; a more detailed view of the station you clicked on where you can see the quadrants and the beam shield pipe in the center. You can also create similar views or zoom-levels as I have decided to call them, for your subsystem. I will explain how, but I assume that you are familiar with Python (or similar programming languages at least) to be able to understand the code and to do changes/additions of your own. I will show an example with 2 zoom-levels when not counting the LHCb detector illustration or device view (which is common for all subsystems).

There are two different approaches to create a new zoom-level; you will have to create the objects to display in the zoom-level in the OnPaintEvent(...), and you will have to write code to detect if any of your devices was clicked in the OnMouseEvent(...). All this is done in visWindow.py, but some small changes are needed in cdbVis.py as well.

### 7.16.3.1 Creating the zoom-level objects

First I will explain how you create the visual objects in your zoom-level. Find the OnPaintEvent(...) method in visWindow.py and look for *elif self.levelzoom == 1 and ...*, this is the next zoom level after the LHCb illustration zoom level (main zoom level), and the same zoom level that contains the Muon station view for the Muon subsystem. You will have to create a *elif self.levelzoom == 1 and self.main.GetActiveSystem() == "MY\_SUBSYSTEMNAME"*: where MY\_SUBSYSTEMNAME is the name of the subsystem you are creating a new zoom level for; in this case zoom-level 1 (You cannot create zoom-level 2 before you have created 1, because you have to go "through" 1 to get to zoom-level 2). Inside this elif-statement, you will have to provide the code to draw the objects for your zoom-level (1) and subsystem. I'll explain step-by-step how to do it:

1. Add the following code in the beginning, right after the statement:

```
corner_x,corner_y = self.CalcUnscrolledPosition(0,0) # upper left corner of the viswin
viswin_w,viswin_h = self.GetSizeTuple() # get width and height of visible visual window
center_x = corner_x + viswin_w/2 # x position for center of visible visual window
center_y = corner_y + viswin_h/2 # y position for center of visible visual window
```

```
if self.macroobjects == []: # if no macroobjects created, we have to create
```

First we check whether we have to recreate the contents of the window or not; we have to if there are no macro objects in the list to draw. Next we get some size and position information from the visual window. This way you will have the size of the visible visual window (width and height), the position of the upper-left corner of the visual window, and the position of the center of the visual window. This way it is easier for you to add the objects to the screen relatively to the center of the screen (which is usually what you want), because then you are sure that it will be shown and visible. Remember that zooming (you know %-zoom) and panning is disabled in every zoom-level except the device-level.

2. If you need to get information about what that was clicked on in the LHCb detector illustration except for the subsystem, you will have to retrieve that information from our "history" list:

```
id_what_was_clicked = self.levelselections["MY_SUBSYSTEM"][0]
```

You will then get the MacroID of what that was clicked if you have defined one for the image-mapped-subsystem.

3. Next step is to draw the objects/drawings you want, to make a "similar" detailed view of your system to the real one. What you do is to create a DrawingObject for each object, with the given drawing information; I will explain every parameter here:

```
my_new_object = DrawingObject(type, object, position, size, penColour, fillColour,  
lineSize, text, startPt, endPt, zoomFactor, clickable, originalsize, macroid,  
textvalign, texthalign)  
  
self.macroobjects.append(my_new_object)
```

Parametername	Explanation
Type	Always: obj_VIRTUALMACRO
Object	Always: None
Position	The position of the object in the visual window in a wxPoint object. Ex: wxPoint(x,y)
Size	The size of the object in the visual window in a wxSize object. Ex: wxSize(w,h)
Pencolour	A wxColour object defining the colour of the border of the object. Ex: wxColour(255,0,0) or wxRed
Fillcolour	A wxColour object defining the colour of the fill of the object. Ex wxColour(0,0,255) or wxBlue

Linesize	The size of the border of the object, default is 1. Valid data: All integers
Text	If the object should have any text written on it, set it here. Valid data: all text strings
Startpt	Ignore, only for links
Endpt	Ignore, only for links
Zoomfactor	Ignore, only for links and devices
Clickable	Whether the object can be click on or not (mouse sensitive), default: True. Valid data: True, False
Originalsize	Ignore, only for links and devices
Macroid	The unique ID for this object in the current zoom level, so that we can find out which object that was clicked in the OnMouseEvent(...) function. Valid data: all integers.
Textvalign	How the text should be aligned vertically on the object. Valid data: text_CENTER, text_LEFT, text_RIGHT.
Texthalign	How the text should be aligned horizontally on the object. Valid data: text_CENTER, text_TOP, text_BOTTOM.
Example:	<pre>DrawingObject(obj_VIRTUALMACRO, None, position=wxPoint(x,y), size=wxSize(w,h), penColour=wxBLACK, fillColour=wxColour(200,100,100), lineSize=5, clickable=True, macroid=1, textvalign=text_LEFT, texthalign=text_TOP)</pre>

Most of the parameters are quite easy to find and set. The exception is the position and size parameters, but the easiest way is to try-and-fail, but that you have some knowledge how to try-and-fail the proper way.

The size of the visible visual window in the standard program view is 552 pixels wide and 508 pixels high. So whatever object you create should not exceed this size, and neither appear outside this visible window (positioning maximum  $552/2=276$  pixels to the left and right of the center and  $508/2=254$  pixels to the top and bottom of the center). Using these figures you could estimate how big your object should be, f. ex. 1/10 of the window height and width would be 55x50 pixels.

The positioning of the drawing object can also be guessed using the size of the visible visual window together with the knowledge that you have the position of the center of the window stored in center\_x and center\_y variables. Figures less than center\_x is to the left of the center, and figures bigger is to the right of the center. Figures less than center\_y is below the center and figures greater than center\_y is above the center.

5. Then you will have to provide the code for, if the self.macroobjects list is not empty, ie. The macroobjects for the given zoom-level have been created, and are just needed to be drawn. Then you do this:

```
else:

    for macroobject in self.macroobjects:

        macroobject.Draw(dc,draw_NORMAL,self.zoom >= 75)

        headline = str("text") # the text to set

        headline_w,headline_h = dc.GetTextExtent(headline) #get size of text, width and
height

        dc.DrawText(headline,center_x-headline_w/2,center_y-viswin_w/2+headline_h) # draw
the text...
```

Remember that you will have to draw the objects you have created in order from bottom to top; objects that are going to lie on top of other objects will have to be drawn last in order to be visible. If you want to draw text somewhere, f. ex. as a header, this will have to be drawn last, like this:

```
headline = str("text") # the text to set

headline_w,headline_h = dc.GetTextExtent(headline) #get size of text, width and height

dc.DrawText(headline,center_x-headline_w/2,center_y-viswin_w/2+headline_h) # draw the
text, start position given for upper-left corner for the text extent, this text is drawn
at the top of the visual window, centered.
```

6. The code we have for zoom level 1 is then:

```
corner_x,corner_y = self.CalcUnscrolledPosition(0,0) # upper left corner of the viswin
viswin_w,viswin_h = self.GetSizeTuple() # get width and height of visible visual window
center_x = corner_x + viswin_w/2 # x position for center of visible visual window
center_y = corner_y + viswin_h/2 # y position for center of visible visual window

if self.macroobjects == []: # if no macroobjects created, we have to create

    # id_what_was_clicked = self.levelselections["MY_SUBSYSTEM"][0]

    my_new_object = DrawingObject(type, object, position, size, penColour,
```



```
fillColour,

    lineSize, text, startPt, endPt, zoomFactor, clickable, originalsize, macroid,
    textvalign, texthalign)

    self.macroobjects.append(my_new_object)

    self.OnPaintEvent(None) # Explicitly call paint event to draw macro objects
we created

    # it is the code below, in the else-statement that will be executed then

else:

    for macroobject in self.macroobjects:

        macroobject.Draw(dc,draw_NORMAL,self.zoom >= 75)

        headline = str("text") # the text to set

        headline_w,headline_h = dc.GetTextExtent(headline) #get size of text, width and
height

        dc.DrawText(headline,center_x-headline_w/2,center_y-viswin_w/2+headline_h) # draw
the text...
```

### 7.16.3.2 Detecting click on zoom-level objects

For this you will have to go to the `OnMouseEvent(...)` method in the `visWindow.py`. Find the `elif`-statement that says `elif self.levelzoom == 1 and self.main.GetActiveSystem() == "MUON"`. You will have to add your `elif`-statement at the same level as this one, and then you set the same `elif`-statement except that the name of the subsystem is substituted with your name of the subsystem.

1. Here's the code you will need to put under the `elif`-statement of yours.

```
    if event.LeftUp(): #if left mouse-button released, check if user did that over
clickable obj

        mousePt = self._GetEventCoordinates(event) # get mouse click
coordinates

        obj = self._GetObjectAt(mousePt,True) # see if the user clicked on a
macro obj

        if (obj != None): # if the user really clicked on a macro obj
            self.macroobjects = [] # we reset the global macro objects
list

            # You set the current selection that was done: Get the macro
id of the

            # object that was clicked, name of the subsystem (active
```

```
subsystem) and

                                # the level to set it for, which is 2 (since we came from zoom
level 1

                                # and are now going to zoom level 2)
                                self.SetLevelSelections(str(obj.GetMacroID()), "YOUR SUBSYSTEM
NAME", 2)

                                self.SetLevelZoom(2) # make level change take effect, zoom
level 2

                                self.SetCursor(wxStockCursor(wxCURSOR_ARROW)) # change to
normal cursor

                                elif event.Moving(): # if mouse cursor moves around in the visual window
                                mousePt = self._GetEventCoordinates(event) # mouse coord.
                                obj = self._GetObjectAt(mousePt, True) # mouse over clickable object?
                                if (obj != None): # if over clickable object
                                    self.SetCursor(wxStockCursor(wxCURSOR_HAND)) # change to hand
cursor

                                else:
                                    self.SetCursor(wxStockCursor(wxCURSOR_ARROW)) # else, normal
cursor

                                return # we will have to return, as we are finished investigating mouse
clicks in this level
```

But if there are no zoom levels for the current system (except the lowest, which is the device level), do not alter anything. If you do not need to add more code than here, then you can use the default one for every subsystem, but if you have to do something different, maybe you can set an if-statement in the code checking the active subsystem, or add a different elif-statement just above this one, because the default one takes all.

If you have the device level (the level where you view all devices the same way) just below this zoom-level, then you will have to change the *self.SetLevelZoom(2)* to *self.SetLevelZoom(-1)*.

### 7.16.3.3 Other changes that are needed to be done

In *cdbVis.py* you will have to find the *self.systemlevels* variable, where it is set (the last thing that is done in the constructor). You will have to find the name of your subsystem in that list, and increment the number to the right of it by 1. That is, if you have a subsystem with only the LHCb detector, and the device level (default), then the number is 1, if you have created zoom level 1, then you put 2, if you have created zoom level 2, you put 3, and so on.

If you want to use the go-down-one-level toolbar button you will have to do the changes in the *OnDownOneLevel(...)* function in *cdbVis.py*. This button is only enabled at the second-to-last level, and a click on that one should go down to the device level. Possibly the code already written for the Muon system can be used, if you provide the name of the device to view in the device level in the *levelselections* list, as is done with the muon system.

*OnUpOneLevel(...)* can be used at most levels, given that you have already gone down the levels. If everything else is donw correct, this function should work, but not from device level and up if not explicitly set.

### 7.16.3.4 More zoom-levels

If you want to add even more zoom-levels (next step would be zoom-level 2, which corresponds to the detailed Muon quadrant view), you will basically do the same stuff as for adding zoom-level 1 with some minor changes.

First, in `OnPaintEvent(...)` you get the selection done in the previous zoom level; `my_choice = self.GetLevelSelections()`. You get the size and position of the window as for zoom level 1, you do the same check to see whether `self.macroobjects == []` or not, creates the objects if necessary, or draws them from the macro objects list if already created.

In `OnMouseEvent(...)`, you do basically the same as you did for zoom level 1, set the macroid of what the user clicked on with the `SetLevelSelections(...)` function, and you set the new level zoom (-1 if the previous was the last level, and the next is the device level).

You will also accordingly have to do changes in the functions in `CdbVis.py` mentioned above if necessary, and at least increment the number of levels for the given subsystem with 1.

### 7.16.3.5 Device types and ports

In the ConfDB you can assign how many ports a device type has, and you can assign ports to a device. The ports of a device have a "name" (id) combining deviceid or device name, portnbr, porttype and portway. It is up to the user to set the names of the ports, and the port name format can be quite different for the different device types (we assume that the port name of devices of the same device type is the same...). In addition; the portnbr can be any string, the same with the porttype, and the portway can be either 1 or 2.

Where one starts to index the ports on a device can also vary; left-bottom, right-top, left-top or right-bottom, and whether one has independent or continuing indexing on the top and bottom of the device type.

It may seem a bit odd that one must have control over this, but the thing is that the program (algorithm) would not know where the port with porttype A and portnbr 1 and portway 1 (in) is on the device if it is not explicitly set somewhere. However, most devices seem to follow a pattern with the same setup of ports, and they are set up and shown as default. But some other devices do not follow this pattern, and must be set explicitly.

The default pattern can be used, and nothing will be needed to be customized if the device type has port names like this (the port type is blank (""), the port number with a number that starts at 0, and input ports on the top and output ports on the bottom):

0 1 2 3 4 5 6 7

|\_\_\_\_\_|

0 1 2 3 4 5 6 7

or

7 6 5 4 3 2 1 0

|\_\_\_\_\_|

7 6 5 4 3 2 1 0

unless it is very important that the links' visual connection (in the visual window) show the correct side the link is connected to. If however the port indexing for your device is different from the ones given above, you

will have to explicitly write code that tells the program where the ports with the given port number and port type can be found on a device.

### 7.16.3.6 Adding a new device type and specific port setup to CdbVis

#### *Drawing the ports on the device*

You will have to go to the `_PrivateDraw(...)` function in the `DrawingObject` class in `visWindow.py` to define where the different ports are located on the given device type. At the moment when this is written, only the "M5R4" and "M4R4" have been added, as well as the default one.

Find the line that states: `if self.GetObject().GetType() in ["M5R4","M4R4"]:`, your `elif`-statement will have to be at the same level as this one, preferably after this one. You write then:

```
if self.GetObject().GetType() in ["MY_DEVICE_TYPE"]: # replace MY_DEVICE_TYPE with the
name of the device type
```

Let's say that the device type you are going to set up explicitly port setup instructions for has the given port setup:

1A 2A 3A 4A 5A 6A 7A

1C|\_\_\_\_\_| 2C

1B 2B 3B 4B 5B 6B 7B

That is: 1-7 in port numbers on the top (all input), and they have all port type A. 1-7 in port numbers on the bottom (all output), and they have all port type B. 1C on the left side, is input, 1 is port number, C is port type. 2C on the right side, is output, 2 is port number C is port type.

You will then have to tell where all these ports are located on the device type, f.ex. like this:

```
i = 0
while i < 7: # first I draw the ports on the top and on the bottom
    xPos = position.x + self.size.width*i/8 # calculate the x position for current port

    yPosTopStart = position.y # y position start for top
    yPosTopEnd = yPosTopStart - self.linkLength # y position stop for top

    yPosBottomStart = position.y + self.size.height # y position start for bottom
    yPosBottomEnd = position.y + yPosBottomStart + self.linkLength # y position stop for
bottom

    dc.DrawLine(xPos, yPosTopStart, xPos, yPosTopEnd) # draw port on top
```

```
dc.DrawLine(xPos, yPosBottomStart, xPos, yPosBottomEnd) # draw port on bottom

i += 1

# then we draw for those two on the sides

dc.DrawLine(position.x, position.y + self.size.height/2, position.x - self.linkLength,
position.y + self.size.height/2) # draw the port on the left

dc.DrawLine(position.x + self.size.width, position.y + self.size.height/2, position.x +
self.size.width + self.linkLength, position.y + self.size.height/2) # draw the port on
the right
```

The position.x is the position of the device, given for its upper-left corner, position.y the same. Self.linkLength is the length of the line that visualizes the port out from the device. Self.size.width and self.size.height is the width and height for the device in question.

Now, that was the easy part. This could also have been done with 16 manual dc.DrawLine(...), but since we saw a pattern for the ports on the top and the bottom, we managed to do it with less code in a while loop.

#### *Find Corresponding Port Index for a given port id (nbr, type and way)*

We have already identified each port with a port nbr, port type and port way (although port type often is not set), and for this specific port we want to find a free place to put it in the port list to a device as well as give it an internal port index. This is all done in FindCorrespondingPortIndex(...) in DrawingObject class in visWindow.py module.

The function looks first for a free entry index to store the port (and later the link) for the port id that was sent to this function. If a free port entry was found, we continue processing. Next we will give each port an index on the device, which we can refer to in another function to get the port position of a specific port. The index does not have to be unique, but the index and the port direction have to be unique, and identify the given port. We can say that the port index is a merging of the port number and the port type to a new index (id/number). F. ex. what that is done for the standard default port indexing (0-n ports on each side, top and bottom, with 0-indexing and no port type), is that the port number that is given is also the port index. This works well, since the port type is not set, and two ports with the same index can be distinguished by looking at the port way.

We continue to use:

```
1A 2A 3A 4A 5A 6A 7A
1C|_____| 2C
1B 2B 3B 4B 5B 6B 7B
```

as an example. We can decide to either have a unique port index for each port (which is often the simplest), or we can have the same port indexes used for input and output ports, because the port direction is different. I will use port index 1 to 16 for all the ports here (16 in total). The code will then look like this:

```
elif self.GetObject().GetType() == "MY_DEVICE_TYPE": # replace MY_DEVICE_TYPE with the
devtypename

    if porttype.lower() in ["a","b"]:

        if porttype.lower() == "a":

            # indexing starts with a ports

            offset = int(portnbr) # ex. port 1a -> 1, 2a -> 2 ... 7a -> 7

        else: # porttype b

            offset = int(portnbr) + 7 # ex. port 1b -> 8, 2b -> 9 ... 7b -> 14

        index = int(portnbr) + offset

    elif porttype.lower() == "c":

        offset = 14

        index = int(portnbr) + offset # ex. port 1c -> 15, 2c -> 16

    else:

        print "Port type error"

        return -1,-1
```

This code is then inserted just below the *if self.GetObject().GetType() == "M5R4"*;, at the same level.

#### *Finding (Port) position for a given port id (nbr, type and way)*

So, now we have told the program how and where to draw the ports, and how to index them. The final thing we will have to code is to let a link know the position to connect its link to on a device for a specific port (given port index and port direction). Then we will have to keep in mind what indexes we gave the different ports in the previous function: it was port index 1-7 for 1a to 7a (all input), port index 8-14 for 1b to 7b (all output), and 15 for 1c input and 16 for 2c output. We will also have to keep in mind the positions we gave the links on the device in the *\_PrivateDraw(...)* method. We combine those mentioned, and return the position (x and y coordinates in a *wxPoint(...)* object) of the port in question.

Look for *if self.GetObject().GetType() == "M5R4"*: in the *GetPortPosition(...)* method, and add your code just below, at the same (intendation) level.

```
elif self.GetObject().GetType() == "MY_DEVICE_TYPE": # replace MY_DEVICE_TYPE with
devtypename

    if index >= 1: # we started our port indexing from 1

        if index <= 7: # first it's the input ports on the top of the device

            # if zoom is less than 50% only 1 link should be shown on each side (because
```

---

```
if too

    # many are shown it is a mess, so we position it to the center of the side
    if GetZoomFactor() < 50:
        xPos = self.position.x + self.size.width/2
        yPos = self.position.y
    else: # normal positioning of ports

        # ports from left to right (0 to left, 7 to right), in 8 parts (index +
1)
        xPos = self.position.x + self.size.width*(index/8)
        yPos = self.position.y + self.linkLength # connect to the port standing
up from dev

    elif index <= 14: # next section; output ports on the bottom
        if GetZoomFactor() < 50:
            xPos = self.position.x + self.size.width/2
            yPos = self.position.y + self.size.height
        else:
            xPos = self.position.x + self.size.width*(index/15) # split in index + 1
parts
            yPos = self.position.y + self.size.height + self.linkLength

    elif index == 15: # the port on the left side
        if GetZoomFactor() < 50:
            xPos = self.position.x
            yPos = self.position.y + self.height/2
        else:
            # remove the +1 to see that the link is not really attached to the port,
so needed
            xPos = self.position.x + self.linkLength + 1
            yPos = self.position.y + self.height/2

    elif index == 16:
        if GetZoomFactor() < 50:
```

```
        xPos = self.position.x + self.width

        yPos = self.position.y + self.height/2

    else:

        xPos = self.position.x + self.width + self.linkLength - 1 # same as
above, -1

        yPos = self.position.y + self.height/2

    else:

        print "Port Index error"

        return -1,-1
```

When you have added this code for your device type, you are ready to test it out. There is a chance that you have done something wrong, and that would result in either a program error, or that the visual display of the ports and/or links connected to it is not really what you expected.

## 7.17 CVS

To get the files from the LHCb repository to be able to modify you will have to need write access (only for developers of course). The URL to the branch for CdbVis is: <http://isscvcs.cern.ch/cgi-bin/cvsweb.cgi/ConfDB/cdbVis/?cvsroot=lhcb>