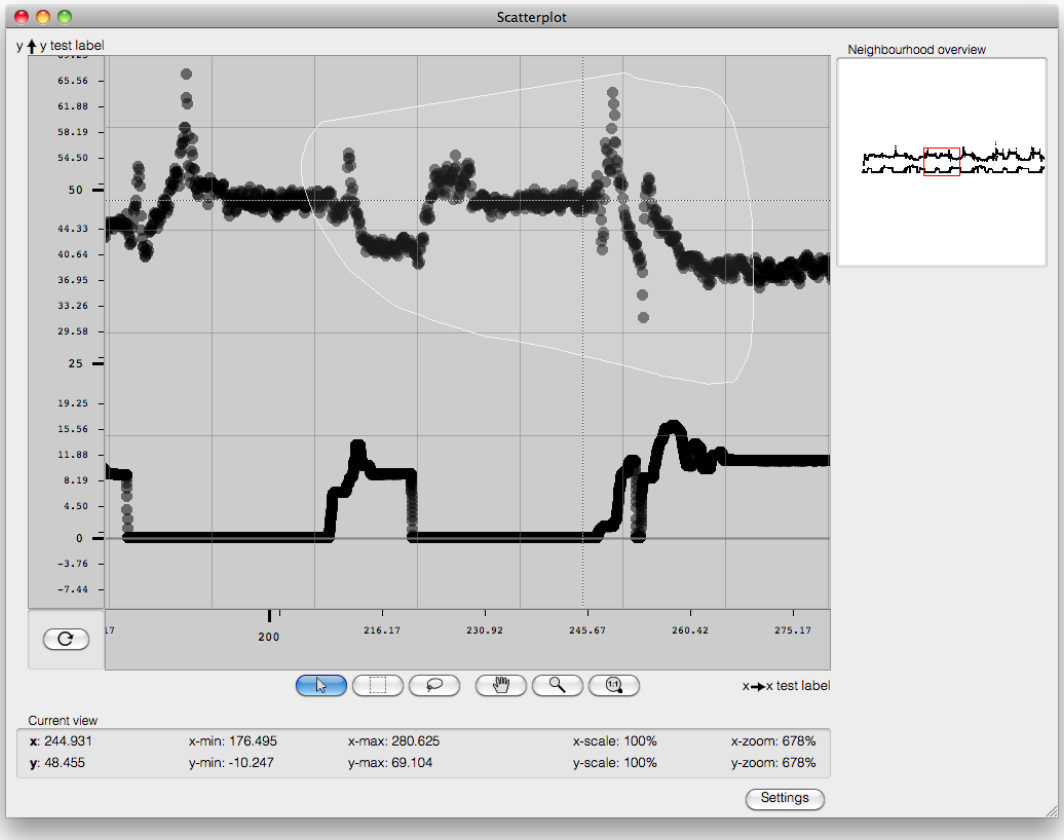
CHALMERS



CC Data Visualization Library:

Visualizing large amounts of scientific data through interactive graph widgets on ordinary workstations

Master of Science Thesis in the programmes
Interaction Design & Algorithms, Languages and Logic

MAX OCKLIND
ERIK WIKLUND

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, May 2012

CC Data Visualization Library:
Visualizing large amounts of scientific data through interactive graph widgets on ordinary workstations

MAX OCKLIND
ERIK WIKLUND

Examiner: Olof Torgersson

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
The GUI of the CCDVL library during a test run, showing a scatter plot graph of sample data with a lasso selection overlay. Sections 4.2.3 FRONTEND AND GUI ANALYSIS AND DESIGN *and* 4.3.3 FRONTEND AND GUI IMPLEMENTATION *as well as* APPENDIX B - MANUAL AND USER GUIDE *contains detailed descriptions of the GUI and frontend.*

Department of Computer Science and Engineering
Göteborg, Sweden May 2012

Abstract

A big problem today is that there are currently few or no solutions available for workstations that are able to provide both good interactivity and fast response times while visualizing large amounts of data. Using existing solutions often cause crashes or freezes due to the assumption that all data will fit in main memory and can be processed as such. The purpose of this thesis was thus to simplify analysis of large amounts of scientific data by creating a small modular and extensible cross-platform graphics library, intended to run on ordinary workstations, capable of handling such data and present it through highly interactive plot graph widgets. Focus was on basic functionality for two-dimensional graphs, namely scatter plot graphs and time series graphs.

The created library implements a memory manager that avoids keeping too much data in memory by controlling and partially storing and loading data to and from a temporary file and the main memory as needed. A clipmap-like structure is used to display the graph, and a cache of clipmap image tiles is used in combination with progressive updates to improve performance and responsiveness. Insufficient planning forced some of the functionality and requirements to be dropped, which left room for much improvement, as well as a lot of possible out of scope additions and extensions, such as better utilization of the cache, supporting groups of selected data points, logarithmic scaling, and improvements to the memory manager, renderer and GUI.

Comparing the results of a simple review and tests of some existing data visualization libraries it was found that the used renderer had average performance, although it had problems using very large or many clipmap image tiles. The resulting library may not be faster than many existing libraries, but it allows faster interaction with good tile configurations and stands out in that it also allows interaction during loading of data and graph updates, while normally avoid consuming all system resources. Its potential is promising, as it thus allows for faster extraction of interesting portions of the data, giving it many possible applications within several areas.

Keywords

Algorithms, Data points, Design, Human-computer interaction, Information visualization, Large data visualization, Memory management, Scatter plot, Time series plot, User interface

Acknowledgements

The authors would like to thank Olof Torgersson, the supervisor and examiner at Chalmers, for his support and feedback throughout the thesis work; Peter Karlsson and Alexander Busck at Combine for their encouragement and inspiring vision; Stefan Larsson at Volvo Car Corporation for his great insight and advice regarding graphs and plotting interfaces; and last, but not least, the authors would like to thank all family and friends for their support.

Table of contents

APPENDICES

List of Figures

List of Tables

Abbreviations

Anti-Grain Geometry (AGG): An open source image rendering library first distributed under the BSD license, and later versions under the GPL licence.

Apache Subversion (SVN): A software versioning and revision control system distributed under an open source license.

Application Programming Interface (API): A description of how applications may communicate, use, extend or borrow provided functionality.

Berkeley Software Distribution (BSD): An open source operating system that comes in a few different flavours, distributed under a nearly unrestricted software license.

Cathode Ray Tube (CRT): Older "thicker" monitors, usually called CRT-monitors, use a cathode ray tube, which is a vacuum tube containing electron guns, which in turn are used to manipulate an electron beam onto a fluorescent screen to displaying images.

CC Data Visualization Library (CCDVL): The library developed during this thesis, where CC refers to the file name extension of the C++ source code files.

Central Processing Unit (CPU): The CPU, or processor, performs computations and executes program instructions.

Dynamic Random-Access Memory (DRAM): One of the most common types of RAM used today, where each data bit is stored in a separate capacitor that is refreshed periodically to keep its charge, thus making it dynamic.

Fast, Light Toolkit (FLTK): An open source GUI toolkit released under the LGPL licence.

GIMP Toolkit (GTK+): A free GUI toolkit originally developed for GIMP.

GNU General Public Licence (GPL): A strict open source license, which GNU is distributed under.

GNU Image Manipulation Program (GIMP): A free image manipulation program.

GNU is Not UNIX (GNU) : An open source reimplementation of a UNIX like system, with a recursive abbreviation.

GNU Lesser General Public Licence (LGPL) :A slightly less strict version of the GPL license, often more suitable for software libraries.

Graphical User Interface (GUI): An interface used by users to interact with applications through displayed graphics; often windows, buttons, checkboxes, etc.

Hierarchical Data Format, version 5 (HDF5): A portable file format with no limit on the number or size of data objects in the collection, distributed under the BSD license.

Integrated Development Environment (IDE): An application facilitating software development, usually providing at least a source code editor, automated building tools and a debugger for the computer programmers to use.

Input/Output (I/O): Data communication (in and out), referring to using either a communication port or a data bus.

Liquid Crystal Display (LCD): A flat panel display, which uses the light modulation properties of liquid crystals to display images.

Not a Number (NaN): An error which can occur during floating point operations, involving infinite values, other NaN values and calculations with undefined results.

Portable Document Format (PDF): A platform independent file format for storing and formatting rich text documents.

Portable Operating System Interface for UNIX (POSIX): A UNIX like operating system API.

Postscript (PS): A format used to describe and store documents.

Random-Access Memory (RAM): The physical main memory of a computer, used by the operating system, processes and applications to store and manipulate temporary data. More specifically, DRAM is usually the type of memory used.

Solid-State Drive (SSD): A type of data storage devices, usually flash memory-based or DRAM-based, which contain no moving parts. They are faster than traditional mechanical storage devices, such as hard disk drives, in that they generally have better read and write speed and performance.

Subversion (SVN): See Apache Subversion (SVN) above.

Symmetric Multiprocessing (SMP): Two or more processors, or processor cores, share and access the same main memory, controlled by a single operating system. This allows the operating system to distribute tasks between the processors, or processor cores, and balance the workload more efficiently.

1 Introduction

This master's thesis was carried out during the late fall and winter of 2011 and finished in the spring of 2012 at Chalmers University of Technology and at Combine AB in Gothenburg, Sweden. Combine is a Swedish company doing work and consultation within the areas of control systems, software development, electronics and mechanics. Their main office is located in Gothenburg, and they have offices located in Lund, Sweden, and Beijing, People's Republic of China, as well (Combine, 2012).

1.1 Background

As technology advances and computers are able to process and store larger and larger amounts of data, there is an ever growing need to visualize and analyse all this exponentially increasing amount of data (Zverina, 2010). This is especially true for the vast amount of multidimensional scientific data generated through tests and measurements, which must usually be visualized one way or another before analysis can begin (Spence, 2007). FIGURE 1 contains a good metaphor that may give the reader an idea of the scale of data that has to be dealt with. Using visualization tools to produce advanced interactive diagrams, graphs or similar graphic visualizations have become common practice, often utilizing more powerful algorithms, data structures, computers or computing solutions, such as high-performance computers or supercomputers to be able to process and interact with the vast amount data (National Endowment for the Humanities, Office of Digital Humanities, 2012). See section 2.2 ALGORITHMIC PROBLEMS for more on algorithms and section 2.6 INFORMATION VISUALIZATION for more on visualizing data.

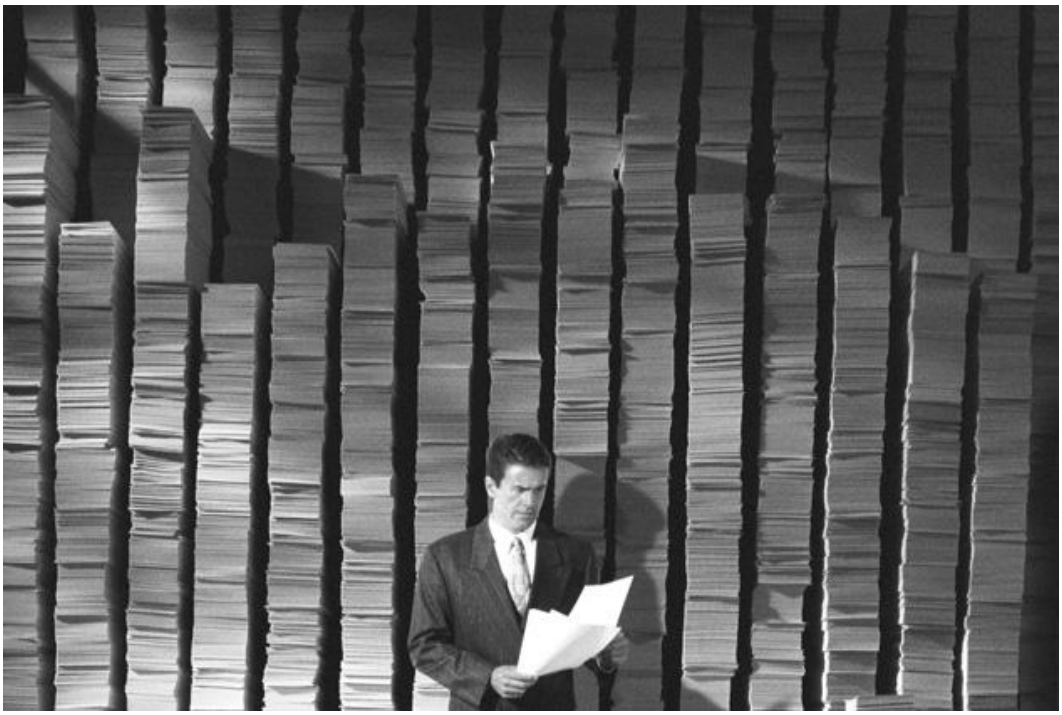


Image source: http://www.businessweek.com/articles/2012-04-23/the-big-deal-about-big-data

Figure 1. *An ineffective way to store and visualize large amounts of data to be analysed.*

1.2 Problem

Currently there are few solutions that still yield fast response times while visualizing large amounts of scientific data on ordinary (less powerful) workstation personal computers (National Endowment for the Humanities, Office of Digital Humanities, 2012). In this context large amounts of scientific data is considered to be millions of measurements or reference points stored in one or more files in the size range of up to a couple of gigabytes each. Such files often contain too much data or are often too large to be handled by existing plotting software running on workstations, causing these to crash due to the assumption that all data will fit in main memory, or seemingly freeze due to lack of responsiveness in the software user interface (Musumeci, 2002). A more detailed description of operating system memory management can be found in section 2.1 OPERATING SYSTEM MEMORY MANAGEMENT. Combine proposed creating a software library addressing these issues, as described in the purpose below.

1.3 Purpose

The purpose of this thesis is to simplify analysis of large amounts of scientific data by creating a small modular and extensible cross-platform graphics library, intended to run on ordinary workstations, capable of handling such data and present it through highly interactive plot graph widgets. The library API should also provide Python bindings to allow smoother integration with other Python-based software projects that are used and developed at Combine.

1.4 Delimitations

Only two-dimensional graphs, with a focus on scatter plot graphs and time series graphs, with the most basic functionality will be considered in this thesis due to the limited time frame. High-level programming languages, such as Java, will not be used to implement the API, due to the need of low level memory management. Further, an external GUI library will be used to save time and avoid reinventing the wheel while working on the interface. Finally, Combine requests that the proposed library will be distributed under the BSD license, and that all external libraries used by the implementation should only use less strict types of licenses, such as BSD or LGPL. This would allow Combine the freedom of use needed when working with the proposed library.

A notebook and a desktop computer will be used for the prestudy and library benchmarks, representing “typical” workstations. The low-end notebook, which will be used for most of the tests, has a 1.6 GHz single core processor, a 120 GB (5400 rpm) hard drive, 2 GB RAM and 2 GB swap. The more powerful desktop computer, which will be used to verify performance gains on better hardware, has a 3.4 GHz single core processor with hyperthreading, a 300 GB (7200 rpm) hard drive, 3 GB RAM and 1 GB swap.

2 Theory

This chapter contains theory relevant to this thesis; theory related to operating system memory management, algorithms, software engineering design patterns, concurrency and multithreading, human-computer interaction, information visualization and graphical user interfaces.

2.1 Operating system memory management

By definition, large amounts of data greater than the total amount of available physical memory (RAM) makes it impossible for an application to keep all this data available in memory. Applications can avoid this problem with memory management, simply by keeping smaller parts of the data in the memory, and by efficiently reusing parts of occupied memory that are no longer needed. Memory is normally managed by the programming language runtime, which in turn uses memory management routines from the operating system. Poor system performance for memory-intensive applications stem from the programing language runtime and how the operating system share system resources. Programming language runtimes usually have no idea what the memory is going to be used for and will consequently simply allocate the required amount. Naive memory-intensive applications relying only on this form of memory allocation will quickly use up system resources.

Modern operating systems try to share memory as a resource between all running applications (Silberschatz, 2008), but unfortunately this also contributes to poor system performance when running memory-intensive applications, as explained above (Musumeci, 2002). Applications are forced to share free memory more aggressively when it begins to run out, and the effects this has on other applications are not completely obvious. Naturally, allocated memory that is not used by an application anymore is safe to reuse, but an operating system cannot safely decide if allocated memory is no longer used, unless told so by the application that allocated it. This will lead to memory shortage when applications request more memory than what is currently known to be available. The solution to this is virtual memory and swapping of virtual memory blocks, which greatly simplifies memory sharing, as described in the next section.

2.1.1 Virtual memory, swap and memory leaks

Every application is given a private memory address space, known as virtual memory, which is not shared with other applications (Silberschatz, 2008). At first this may seem to contradict the concept of sharing memory, but only the data of the application is made private, not the physical memory on which it is stored. For convenience, virtual memory is divided into small pieces of equal size, called pages, each mapped to a frame in physical memory (Silberschatz, 2008). An operating system may move rarely used pages from their frames to a storage media, which is known as swapping, releasing used memory and thus allowing more memory than physically available to be allocated by applications. FIGURE 2 below shows an overview of this setup, using a hard disk drive as storage media.

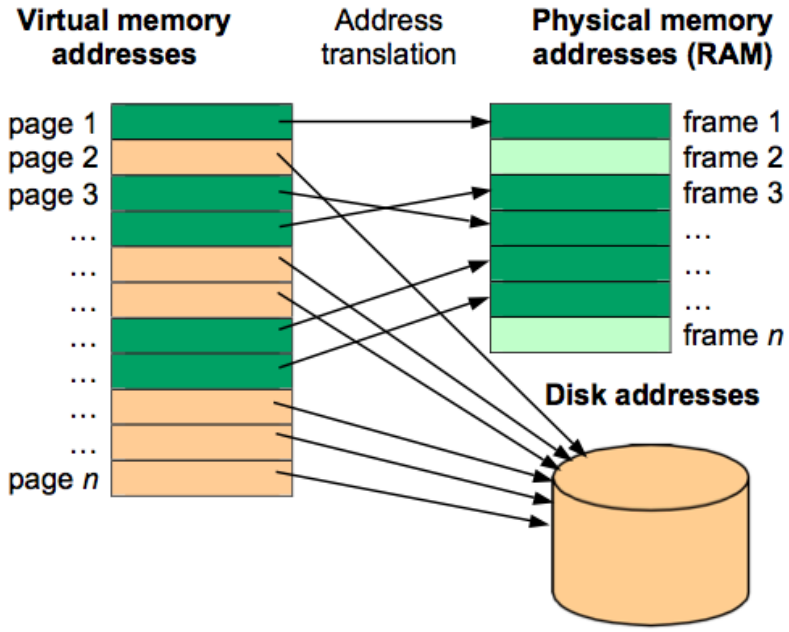


Figure 2. *An overview of the relationship between virtual memory, physical memory and disk memory. The light-brown pages (such as page 2) have been swapped to the disk, while the green pages (such as page 1) are still retained in the physical main memory. The pale-green frames (such as frame 2) represent free unused memory.*

A page fault occurs when an application attempts to access a swapped page, which also halts application execution. The page fault is caught and handled by the operating system, which proceeds to move the missing page back into a frame for use, allowing application execution to continue (Silberschatz, 2008). The primary cause of poor application performance comes with large amounts of swapping, due to the fact that accessing storage media is an order of magnitude slower than accessing physical memory (Pudov, 2011). Memory intensive applications often tend to access swapped pages and are forced to wait for longer periods of time until the required pages are loaded. Memory leaks may also contribute to poor application performance as larger or many memory leaks will effectively decrease the total amount of available memory over time. A memory leak occurs when an application do not reuse or release allocated memory during its runtime, but rather requests more memory from the operating system instead of reusing it. As memory shortage increases the number of swapped pages, the overall application performance will quickly decrease (Wiseman, 2009), also resulting in an unresponsive operating system that seems to freeze (Musumeci, 2002). This happens because the operating system tries to be fair, freeing up frames used by other applications to run a memory intensive application, until another page fault occurs, which will likely happen frequently during a memory shortage. The problem then becomes I/O overload due to swapping, while the CPU is forced to wait for pages to return to suitable frames to be able to access them. A more natural way of describing this problem is system wide resource starvation, where applications are denied access to their data stored in memory and must wait for it to become available.

2.2 Algorithmic problems

This section describes algorithms in general, algorithmic problems that are interesting in the context of this thesis, and their solution strategies.

2.2.1 Algorithms

An algorithm is a well-defined method for solving specific computational problems (Arora, 2009), and any instance of the same type of algorithmic problem can therefore be solved by the same algorithm. Kleinberg (2005) states that "*algorithmic problems form the heart of computer science, but rarely arrive as cleanly packaged, mathematically precise questions*" (Kleinberg, 2005, pp. xiii).

Algorithm efficiency is an important topic in addition to correctness, as it allows different algorithms and their implementations, which solve the same kind of problem, to be compared. Algorithmic problems are classified into categories, called complexity classes, based on how hard they are to solve, measuring required resources to solve a problem. However, quantifying algorithm efficiency requires some thought; as inefficient algorithms could be as fast as an efficient one, given powerful hardware and small inputs (Kleinberg, 2005). This in turn leads to the area of computational complexity theory, which focuses on measuring efficiency of computations (Arora, 2009).

2.2.1.1 Algorithm complexity analysis

When considering the resources, and the amount needed, which are required to solve a problem, time is often the first resource that comes to mind, at least in the context of algorithm efficiency. But relying on time alone is not very dependable; simply running other applications in the background could skew the results, which are hardware and input dependent as well. However, running-time can serve as a complement to regular time complexity analysis (Collins, 2005). There are three major points of interest when analysing an algorithm:

- **Correctness:** an algorithm must yield an acceptable result for all inputs.
- **Time complexity:** the number of steps needed to solve a problem.
- **Space complexity:** the amount memory needed to solve a problem.

The degree of importance for each point may vary, depending on the requirements of an algorithm. Consequently, there often exist numerous algorithms based on different designs to solve the same problem.

For correctness, a proof showing that it is not possible for an algorithm to be incorrect is required. However, it is not always the case that an optimal answer for a problem is needed; a good enough result or approximation could be just as interesting for problems that are difficult to solve exactly. Algorithms that are designed to sacrifice correctness

for speed, are called approximation algorithms, and their correctness is shown using a bound guarantee (Kleinberg, 2005), with a lower optimal answer and an upper, worst answer.

Time complexity does not simply measure the running-time of a few instances of algorithms solving problems. As mentioned previously, such tests are not very useful due to interference from applications running in the background, as well as operating system and hardware limitations (Collins, 2005). Instead, a more general approach is used, where one counts the number of basic computational steps needed by an algorithm to find an answer. A formula is derived from an analysis of a worst case scenario using only the problem input size as a variable. If the chosen input is considered to give the largest possible number of steps, the function obtained then yields an upper bound of the number of operations needed for an algorithm to complete. Using big-O notation, functions are simplified and stripped from constants and less dominant expressions, leaving out a worst case asymptotic behaviour depending on input size (Kleinberg, 2005). This time complexity analysis is crude, as large constants and extremely small factors are lost. A definition of big-O based on the theory provided by (Kleinberg, 2005), including an example follows.

Let $T(n)$ be any function, but for clarity, say that it yields an algorithm's worst running time for a problem of size n.

Let $f(n)$ be a function.

$T(n)$ is of order $f(n)$ if $T(n) \leq C * f(n)$, for all $n \geq K$, where constants $C > 0$ and $K \geq 0$.

$T(n)$ is of order $f(n)$ is then written as $T(n)$ is $O(f(n))$.

Example

Let $T(n) = 2 * n^2 + n + 8$, then $T(n)$ is $O(n^2)$.

Proof

$2 * n^2 \leq 2 * n^2$, $n \leq n^2$ and $8 \leq 8 * n^2$ for $n \geq 1$.

Thus $T(n) \leq 2 * n^2 + n^2 + 8 * n^2 \Leftrightarrow T(n) \leq 11 * n^2$, for $n \geq 1$.

Choosing $C = 11$ and $K = 1$, then $T(n) \leq C * f(n)$ holds for $f(n) = n^2$ and all $n \geq K$.

An interesting implication from the definition is that $f(n)$ can overestimate the order of $T(n)$, if $T(n)$ is $O(n^2)$ then it also is $O(n^3)$ (Kleinberg, 2005). Further proofs for big-O will not be provided here, since it should be clear from the proof above that using the fastest growing term without constants is enough.

Space complexity analysis is similar to time complexity analysis, but instead of counting computational steps, the number of variables used is counted (Collins, 2005). For example, an array of length n would count as n variables (Collins, 2005). There are also dynamic programming algorithms that sacrifice memory in favour of speed by memorizing results for a recursive function (Kleinberg, 2005).One of the most clear examples where this technique improve performance is when computing the ith Fibonacci number, simply by saving the two previous numbers so that the next number, which by definition depend on the two previous numbers, can be quickly computed instead of inefficiently recomputing all previous recursive steps over and over.

2.2.1.2 Polynomial problems

The polynomial class P can be defined by using a parameterized class TIME($f(n)$), proposed by Papadimitriou (1993). If TIME($f(n)$), or DTIME($f(n)$) for deterministic time, is the complexity class containing problems that can be solved in O($f(n)$) time by a deterministic Turing-machine, where $f : \mathbb{N} \rightarrow \mathbb{N}$ and $f(n + 1) \geq f(n)$ for all $n \in \mathbb{N}$ (Papadimitriou, 1993), then:

$$P = \bigcup_{k \in \mathbb{N}} DTIME(n^k)$$

Problems belonging to P are therefore by definition solvable in polynomial time because a Turing-machine is essentially a simple computer derived from a state-machine and uses a number of tapes as memory. Algorithms solving problems in P are considered fast compared to those solving problems exclusively belonging to more difficult complexity classes, while k could be arbitrary large, algorithms solving problems in P are seldom slower than O(n5) (Arora, 2009). It has also happened that newly discovered algorithms in P with an initial large time complexity were simplified enough afterwards to be no slower than O(n5) (Arora, 2009).

2.2.1.3 Nondeterministic polynomial problems

Defining the non-deterministic polynomial class NP using NTIME($f(n)$), which is the class of problems solvable by a nondeterministic Turing-machine in O($f(n)$) time (Papadimitriou, 1993), gives:

$$NP = \bigcup_{k \in \mathbb{N}} NTIME(n^k)$$

An alternative definition is that a problem belongs to NP if there exists a deterministic Turing-machine that can verify if a given answer to a problem in NP is correct in polynomial time; thus problems in NP are efficiently verifiable versus problems in P that are efficiently solvable (Arora, 2009). It should now be clear that that $P \subseteq NP$. However, it may not be clear that NP actually contains more difficult problems. A nondeterministic Turing-machine has two transition functions at any given state (Arora, 2009), and can thus search multiple computational paths with each computational step.

A simpler way to look at it is that it will always make a correct choice, or that it is extremely lucky at guessing.

NP-complete problems, which are the hardest problems in NP, are considered to be computationally hard for all practical purposes, but it has yet to be proven (Kleinberg, 2005). This leads to one of the largest questions within computational complexity and computer science in general: is $P = NP$ or is $P \subsetneq NP$, or stated more simply: does efficient algorithms exists for all problems in NP or not. This question is further promoted by the large number of problems that appear in NP, not just the well-known traveling salesman problem, but also aspects of creative effort seem to be included, and so the number of known NP problems are now thousands (Arora, 2009). An interesting feature of all NP-complete problems is that they can be reduced to each other with algorithms running in polynomial time, which means that if it can be proven that one of the NP-complete problems is in P, then all NP problems are in P (Arora, 2009). A proof for either $P = NP$ or $P \subsetneq NP$ has yet to be found, so this still remains an open question. However, most believe that $P \subsetneq NP$ because $P = NP$ is simply too good to be true, although the implications of this, were it to be true, would be of huge interest for practical, scientific and philosophical areas (Arora, 2009).

2.2.1.4 Other complexity classes

To provide a perspective of the jungle of complexity classes, it is worth mentioning that there exists close to 500 known complexity classes (Aaronson, 2012), a few of which with higher importance are listed below.

- **ALL** - The class of all languages.
- **EXP** - Exponential time.
- **PSPACE** - Polynomial space.
- **PP** - Probabilistic polynomial time.
- **coNP** - Complement of NP.
- **BPP** - Bounded-error probabilistic polynomial time.
- **NP** - Non-deterministic polynomial time.
- **P** - Polynomial time.
- **L** - Logarithmic space.

These will not be described further, as they are of little interest to the thesis, but the interested reader is referred to Aaronson (2012), Arora (2009) and Papadimitriou (1993).

2.2.2 Binary search

This is one of the most basic and common methods that quickly find a specific piece of data in a (strictly) sorted data set (Kleinberg, 2005). Binary search has a time complexity of $O(\log(n))$, compared to a naive lookup approach that uses $O(n)$ steps dataset (Kleinberg, 2005). In short, the fact that the data is sorted allows the search space to be cut in half by each operation, instead of looking at every location in sequential order using a naive lookup approach. A common data structure which utilizes this is, for instance, a tree.

2.2.3 Cache

Caching is a method used to improve performance by temporarily saving specific data, which is often normally slow to obtain, when that data is expected to be needed again soon, allowing faster retrieval for sequential request. A cache is usually a chunk of fast memory with a fixed size or limited by some other rule, such as maximum number of bytes, maximum number of items stored, or a combination of both. Examples where caching can be used to improve performance are costly computations, slow network links, slow mass storage devices and similar.

Cache efficiency is defined by how often a requested item is found in the cache or not, called hit or miss respectively, and an algorithmic optimal cache must minimize the number of misses (Kleinberg, 2005). When a miss occurs in a full cache, one or more items must be evicted to make room for the missing item. But selecting which item or items to evict is the interesting part, there exist multiple algorithmic strategies. Two such are described below.

- **Farthest-in-Future:** if future cache requests are known, the item needed as far as possible into the future will be evicted first. This algorithm has been proven optimal by Belady (1966), but in real world scenarios future requests are often not known, making this algorithm unusable. However, it is used as a quality measurement when evaluating other strategies (Kleinberg, 2005).
- **Least-Recently-Used:** an item's last-hit timestamp can be used when future requests are not known, evicting the item with the oldest timestamp first. It is known that this strategy's maximum number of misses is roughly bounded by $e * n$, where e is the optimal number of evictions and n is the maximum number of items. A randomized variation with slightly better results also exists, bounded by $e * \log(n)$ (Kleinberg, 2005).

2.2.4 Point in polygon

Just as the name suggests, this is a straightforward true or false question; is a given point inside a given polygon or not? This problem appears when, for example, a user interface needs to find out which button was clicked and in collision detection for games

(O'Rourke, 1998). There exist several different (polynomial time) strategies for solving this problem, as described below:

- **Angular sum:** uses the winding number of a polygon to determine the number of times a polygon winds around a point (Huang, 1996).
- **Odd or even:** uses a ray crossing test to determine the number of times a line, starting in a given point and ending outside a polygon, crosses each polygon edge (Huang, 1996).
- **Grid:** uses a grid instead of vertices to represent a polygon, where the cells of the grid are known to be inside or outside the polygon and the point covers a single cell (Huang, 1996).
- **Swath:** uses a ray crossing test, with several preprocessing steps to divide a polygon into swaths to allow binary search to be utilized over the y-axis (Huang, 1996).
- **Wedge:** divides a convex polygon into wedges. A point then only needs to be tested against a single wedge, which can be found quickly using binary search (Huang, 1996).

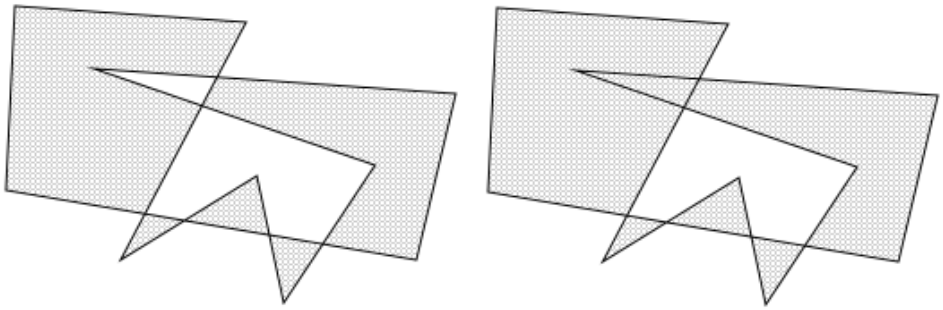
Time complexity for the angular sum, odd or even and grid strategies is $O(n)$ (Huang, 1996). Swath and wedge have a better expected time complexity of $O(\log(n))$, but require preprocessing steps (Huang, 1996). Swath's preprocessing time complexity is $O(n * \log(n))$, and additionally, in a worst case scenario swath might use $O(n^2)$ amount of memory instead of $O(n)$, while still taking $O(n)$ time (Huang, 1996). Considering this, wedge's preprocessing time complexity is only $O(n)$ (Huang, 1996), which would make wedge a better choice when only convex polygons are expected.

These tests could be extended with a bounding box check, to first testing if a point is near a polygon, thereby potentially avoiding a full point in polygon test. This could improve performance, but if a series of close points are tested in a tight-loop it will have an opposite effect on performance and improvements will be minor, if any, for polygons with few vertices.

Angular sum is the sum of all angles between a point and two endpoints in a segment of a closed curve, if and only if that sum is zero then the test point is outside the polygon (Hormann, 2001). Angular sum is often mistaken to be completely different from and inferior to odd or even, however this was disproven by Hormann (2001), by showing that a fast ray crossing algorithm equal to winding number exists. O'Rourke claimed that the reasons why winding number was not as fast was its dependence on floating point and trigonometric operations (O'Rourke, 1998), and indeed, both floating point and trigonometric operations are much slower than integer and boolean operations (Huang, 1996). But Hormann's (2001) algorithm does not need floating point numbers, nor trigonometric operations, thus avoiding those problems altogether.

Odd or even makes its decision based on if the polygon line cross count is odd or even. A line starting in a given point and ending outside a polygon is used, and the edge cross count is odd if the point is inside the polygon. The winding number strategy behaves

slightly different in comparison; if the polygon is self-intersecting, points in those intersecting areas have a non-zero winding number, but their cross count can be even, depending on the number of overlapping self-intersections (Hormann, 2001). This difference is shown in FIGURE 3.



(a) *odd or even* (b) *winding number*

Figure 3. *The shaded areas of this self-intersecting polygon represent the areas that the (a) odd or even and (b) winding number strategies consider as being inside the polygon.*

Despite a very good time complexity, wedge is at a disadvantage since it only works with convex polygons. Nevertheless, it is possible to apply this algorithm to a simple (non-self-intersecting) polygon by performing an additional preprocessing step to partition a polygon into convex polygons. Unfortunately, it is more difficult to find an optimal convex partitioning then a suboptimal one (O'Rourke, 1998); in fact it is NP-hard for polygons with holes (Lien, 2004). One way to partition a simple polygon into convex polygons is by triangulation, since all triangles are strictly convex. Triangulation can be done in $O(n)$ (Chazelle, 1991), while optimal convex partitioning can be done in $O(n + N^3)$ time where N is the number of notches (Chazelle, 1980). An interesting approximation algorithm exists, capable of exactly solving simple polygons without holes in $O(n * N^2)$ time, or finding an approximation for polygons with or without holes in $O(n * N)$ time (Lien, 2004). A short analysis of the point in polygon query for a naive approach using a convex partitioning preprocessing and wedge as core follows below.

Let $f(a, b)$ be wedge point in polygon function, where a is a point and b is a convex polygon.

Let Q denote a set of convex polygons created from a polygon in the preprocessing step.

Assume that $\forall t, t \in Q$, are already preprocessed by wedge.

Let x denote a point to test.

```
For each convex polygon p ∈ Q
    If f(x, p)
        Return true
Return false
```

Wedge query complexity is $O(\log(n))$, but with triangulation preprocessing, n equals three, giving each test a constant time complexity. The number of operations then depend on how many triangles a simple polygon can be divided into, which is exactly $n - 2$, where n in this case is the number of vertices (O'Rourke, 1998). Therefore $T(n) = (n - 2) * \log(3)$, which yield a time complexity of $O(n)$, no better than other proposed algorithms for the point in polygon problem except for the preprocessing step. If Chazelle's (1980) optimal algorithm is used for preprocessing instead, where the number of convex polygons is bounded by $N / 2 + 1$ and $N + 1$ (Chazelle, 1980), wedge will then in the worst case consider all vertices from a partitioned polygon (if the polygon was already convex). Since the number of partitions is at worst $N + 1$ (if they are all triangles), the time complexity becomes $T(N, n) = (N + 1) * \log(n)$. This results in a crude time complexity of $O(N * \log(n))$, while Chazelle (1980) expects N to often be small, making not only the preprocessing step more feasible, but also the point in polygon query as a whole. However, it is not clear if this is enough to perform better than much simpler algorithms for the point in polygon problem on any simple polygon. Intuitively, if a simple polygon with many vertices is fairly divided when partitioned, and N is reasonably small, then a faster query can be expected.

2.3 Concurrent programing and multiprocessing

Modern computers are capable of performing computations in parallel, whether it is through time multiplexing, additional computational hardware or performed by an external device. Applications that utilize multiprocessing, to fully exploit the capabilities of modern hardware, are by design split into multiple processes, where each process is a sequence of operations that can be executed in parallel. Communication between two or more processes can be viewed as a channel through which messages are passed, containing operations to execute, data to process, processed data and computational results, which are useful when dealing with external devices and distributed computing. In practice, modern computers use SMP (symmetric multiprocessing), where the same memory is shared between computational hardware and can be accessed in an equal amount of time (Andrews, 1999).

The power of multiprocessing also leads to additional problems for software developers. Consider the following scenario: process A and B work in parallel, thus the interleaved order of each operation cannot be determined. Process A fetches value x from memory; process B fetches value x from memory; process A computes $x + 1$ and stores it back into memory; process B computes $x + 1$ and stores it back into memory, this time overwriting the result from process A. This is likely unwanted behaviour since the result from process A is lost.

Two basic synchronization directives are used to solve these issues; mutual exclusion (mutex) for critical sections and conditional synchronization. These form basic synchronization methods, such as semaphores and signals used in the POSIX thread standard (IEEE, 2008). But with synchronization other risks also arise, such as processes waiting indefinitely on each other, known as a deadlock. Furthermore, the three primary overheads in parallel applications must be minimized to achieve the highest possible performance gain, namely process creation and scheduling, communication and synchronization (Andrews, 1999).

2.4 Software engineering design patterns

Software engineering design patterns are reusable programming techniques used to solve common design problems and to avoid common pitfalls.

2.4.1 Observer

Clients normally read data from relevant objects, but when objects change the clients must either read that data again, or preferably be informed of this by the objects themselves. A client may list itself as an observer for an object, making the object inform all such listed clients whenever it is has changed, effectively creating a one-to-many dependency between an object and multiple clients (Metsker, 2006). It is a common pattern found in user interfaces (UI), which can be used to update multiple views in real-time whenever a related object is changed.

2.4.2 Visitor and double dispatch

The purpose of the visitor pattern is to add new operations to an existing class hierarchy, without actually changing it. There are many advantages to this pattern, such as that a developer may include support to easily extend class hierarchies with new behaviour (Metsker, 2006). The pattern itself consists of a class and a visitor interface, which has a number of subclasses. The interface has a method, often called visit, overloaded with many different arguments to extend functionality. Each subclass that can be extended has an accept method that take this visitor as an argument and will in turn call the appropriate visit method on the provided visitor. This is called a double dispatch.

However, the most beneficial gain is that this makes it possible to eliminate runtime type information, which would otherwise cause overhead and is a common source of problems and bugs. These problems stem from that the compiler cannot guarantee that such a runtime typecast is valid, but by using the visitor pattern, the compiler can and will identify incorrect use and help to ensure that no incorrect type conversion ever take place. The visitor interface then contains visit methods that accept all possible subclasses, and all accept methods simply call visit with its own instance, thus revealing their type.

2.4.3 Iterator

The iterator pattern defines a way of sequentially accessing objects or data inside a container by traversing the underlying data structure without exposing it (Metsker, 2006).

2.4.4 Readers-writer lock

A type of concurrent design pattern used to protect a shared resource to make sure it can only be written to by one process at a time, but read by more than one process when no writers are writing (Silberschatz, 2008).

2.4.5 Message queue

A message queue can be seen as an abstract mail inbox, where messages are processed strictly in the order they arrive, i.e. first in, first out (Andrews, 1999).

2.5 Human-computer interaction

The aptly named field of human-computer interaction (HCI) focuses on the interaction between a person (the user) and a computer system, with an emphasis on understanding the user, his or her behaviour, motivations, needs and desires, as well as the context of the users and the underlying causes behind their behaviour (Hewett, 1996; Cooper, 2007). It can also be viewed as a model in which people, tasks, technology and the social, organizational, and physical environment are closely related. HCI is thus a very complex and broad concept, which covers numerous aspects, as shown in FIGURE 4.

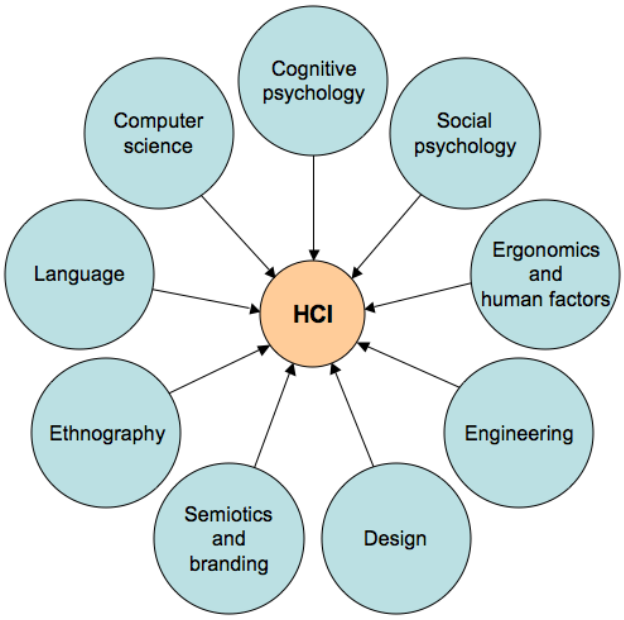


Figure 4. *An association chart showing some of the main fields contributing to human-computer interaction.*

Preece (1995) describes the main aspects of a few of the bigger fields that contribute to HCI, shown in FIGURE 4 above. For example; computer science provides knowledge and insight about technology and a wide range of software-based tools and methods for facilitating design and development of computers, systems and user interfaces; cognitive psychology provides knowledge and insight about the capabilities and limitations of the users of a system; social psychology uses ethnomethodology to help

explain the functions and structure of organizations; and ergonomics and human factors provide knowledge of how to ensure that hardware and software are designed in such a way that they do not harm the users physically (Preece, 1995).

Dix (1998) concludes that there exist no general or unified theories covering HCI, and that it may never be possible to derive one. There are however some basic underlying principles related to the three major parts of HCI; the user, the computer and the task. The understanding of these is used to improve the interaction and better meet the need and goals of the users, for example by making better and easier to use hardware, as well as better designed software and interfaces that behave more according to the mental model and expectations of the users. Cooper (2007) points out that the usage patterns of users are motivated by both the goals and mental models of the users. The mental model of a user, also called the conceptual model, is how a user thinks something works, regardless if this view is correct or not. Users thus create cognitive shorthands that are powerful enough to cover all their interactions with, for example, a system, since they do not need to know its inner workings to be able to use it (Cooper, 2007). The users' motivations and goals can further be understood by working with HCI as a user-centered and highly iterative process (Preece, 1995).

Both Cooper (2007) and Tidwell (2011) state that to help understand the need and goals of users one should also utilize common user research methods, such as interviews, direct observation, case studies, focus groups, surveys or personas. These all vary in the degree of formality and efficiency, depending on the type of user, context and tasks that are dealt with. Many of these can also be used as evaluation techniques, as described by Dix (1998). Some HCI aspects are discussed further in section 2.6.4.3 THE ACTION CYCLE AND HUMAN ASPECTS OF INTERACTION.

2.6 Information visualization

Information visualization offers great potential in understanding and gaining insight into (large) sets of data through a wide range of different methods of representation, presentation and interaction, which allow information to be derived from data (Spence, 2007). The underlying philosophy of information visualization is, according to Spence (2007), to represent (or encode) a problem in such a way that the solution becomes apparent, and he goes on pointing out that *"choosing an effective way to encode data to make the solution transparent is far from simple and continues to challenge and fascinate practitioners and researchers"* (Spence, 2007, pp. 6).

Because of the vastness of information visualization as a subject, as well as the many related and overlapping fields, this section will focus only on the most basic and some specific parts of theory and examples of information visualization, highlighting some of the most important aspects of representation, presentation and interaction. The extensive use of computers in the field also makes the broad field of human-computer interaction important, as described in the previous section.

Computer-aided information visualization has skyrocketed during the last 15-20 years due to the availability and improvement of computers and related technology, but it is important to note that, in the end, visualization is a human cognitive activity involving

perception and interpretation, which in itself has nothing to do with computers. Visualization relies heavily on the mental model of represented data, formed by human beings to help them interpret and process the data. Computers offer great support and advantages for visualizing information as they can store vast amounts of data, allow fast interactive selection of subsets with responsive interaction and have high-resolution graphics that can be used for complex presentations. To be able to encode data in a good way to support the mental model and to successfully use the advantages that computers offer one must understand the aspects of human behaviour and characteristics, as well as how human beings interact with data (Spence, 2007). Graphical representations are especially good at visualizing large amounts of data, as well as conveying complex ideas with clarity, precision, and efficiency (Brath, 1999). Brath (1999), focusing more on graphs, and also including the human aspects, basically says the same thing as Spence by stating that *"a graph has high expressive power when drawn aesthetically on a plane: the visual representation exploits human visual cognitive capabilities, activates intuitions, helps find solutions, and makes complex situations understandable"* (Brath, 1999, pp. 10).

Spence (2007) provides a compact and easy to understand visualization on the entire process of information visualization, from data to interpretation and interaction, recreated below in FIGURE 5.

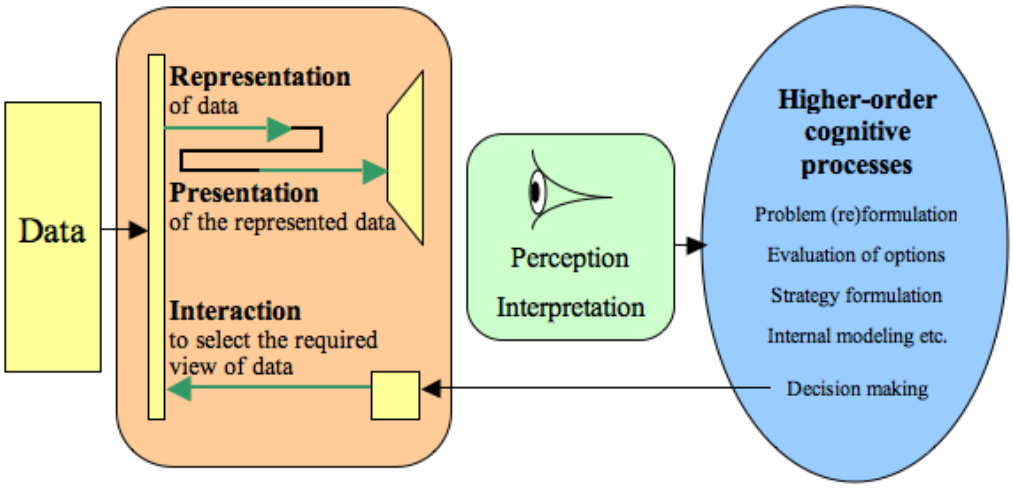


Image source: Reproduction of Spence (2007)

Figure 5. *A summary of the information visualization process.*

2.6.1 Examples of information visualization

Two excellent classical examples of data maps are presented below, each capturing the very essence of information visualization according to Spence (2007), as well as more recent examples depicting a radial table and a complex Venn diagram. The first example is Minard's map of Napoleon's march and retreat to and from Moscow in 1812-1813 (see FIGURE 6). At just a glance most people find it easy to see and understand that the thickness of the lines represent the size of the army, and that the brown line represents the advance and the black line the retreat. Thus a lot of information and understanding can be gained without reading any of the (French) explanations written on the map.

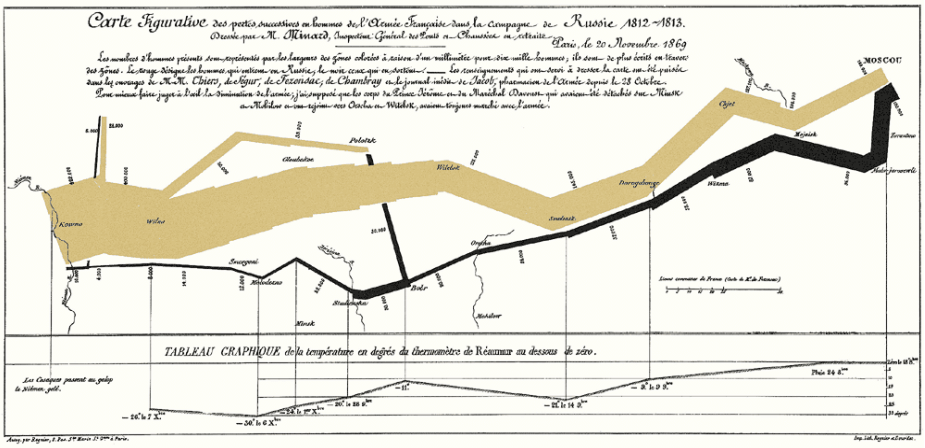


Image source: http://www.edwardtufte.com/tufte/minard

Figure 6. *Charles Joseph Minard's map of Napoleon's march and retreat to and from Moscow.*

The second example is a map of the Soho district in London during the cholera pandemic in 1845, showing the locations of cholera related deaths and the locations of the district's water pumps (see FIGURE 7). Dr John Snow, the medical officer at the time, observed that the biggest number of deaths was concentrated around the water pump at Broad Street. He concluded that this was a likely source of contaminant, and was able to decrease the number of deaths by putting a lock on the pump.

Image source: http://greatwen.com/2011/02/15/london-in-maps/

Figure 7. *A cluster map of the 1845 Soho district in London, showing cholera deaths (as dots) and water pumps (as crosses). The pump at Broad Street is located roughly at the center of the map.*

Tufte (2001) also agrees that the works of Minard and Snow are two very good examples of use of data maps to communicate complex ideas with clarity, precision and efficiency; what he refers to as graphical excellence. He even goes so far as to exclaim that Minard's map "*may well be the best statistical graphic ever drawn*" (Tufte, 2001, pp. 40).

The third final example is that of the relationships between cars and different types of car models visualized in a radial table, shown in FIGURE 8. A radial table, a design pattern defined by Tidwell (2011), is useful when you need to show the arbitrary relationships between the items in a long list. This way flows, connections, similarities, affinities and value can be represented with arc line color and thickness, and the size and length of the lines and arcs. Sometimes it is also easier to recognize patterns when data is displayed in a radial table, and well-made radial tables are often aesthetically beautiful and attractive (Tidwell, 2011).

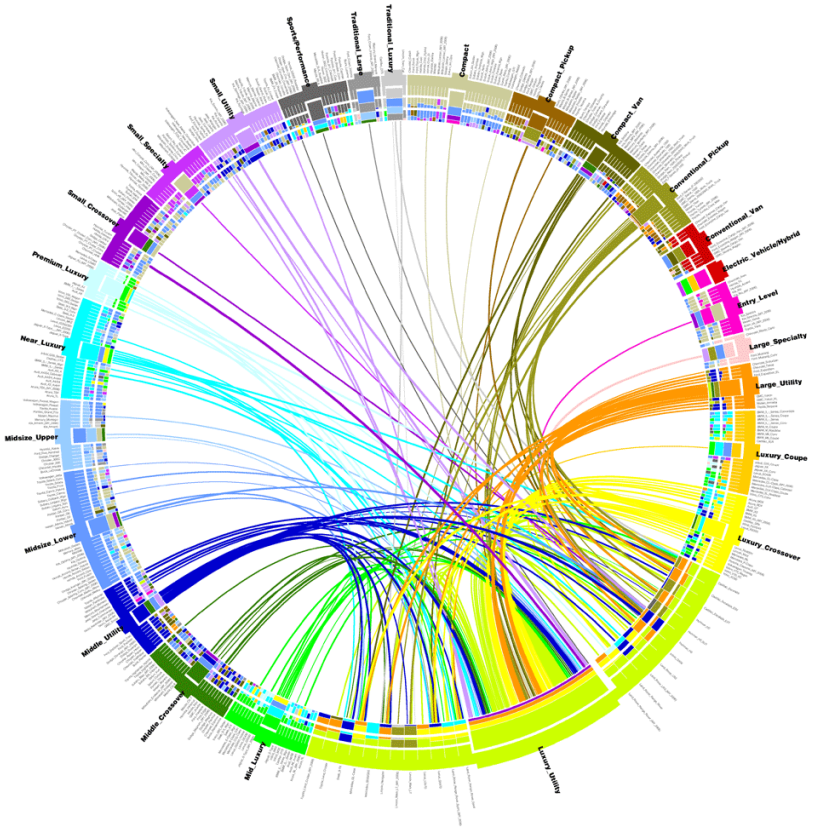


Image source: Tidwell (2011)

Figure 8. *A radial table showing the relations between different cars and car models.*

The final example is a complex and very impressive Venn diagram, shown in FIGURE 9, effectively acting as both example of information visualization and overview of the elements making up data visualization (ffunction Inc., 2010). Please also go ahead and take a look at figure image source for a larger version, as the scaled down version showed in FIGURE 9 does not do it justice; http://blog.ffctn.com/what-is-data-visualization.

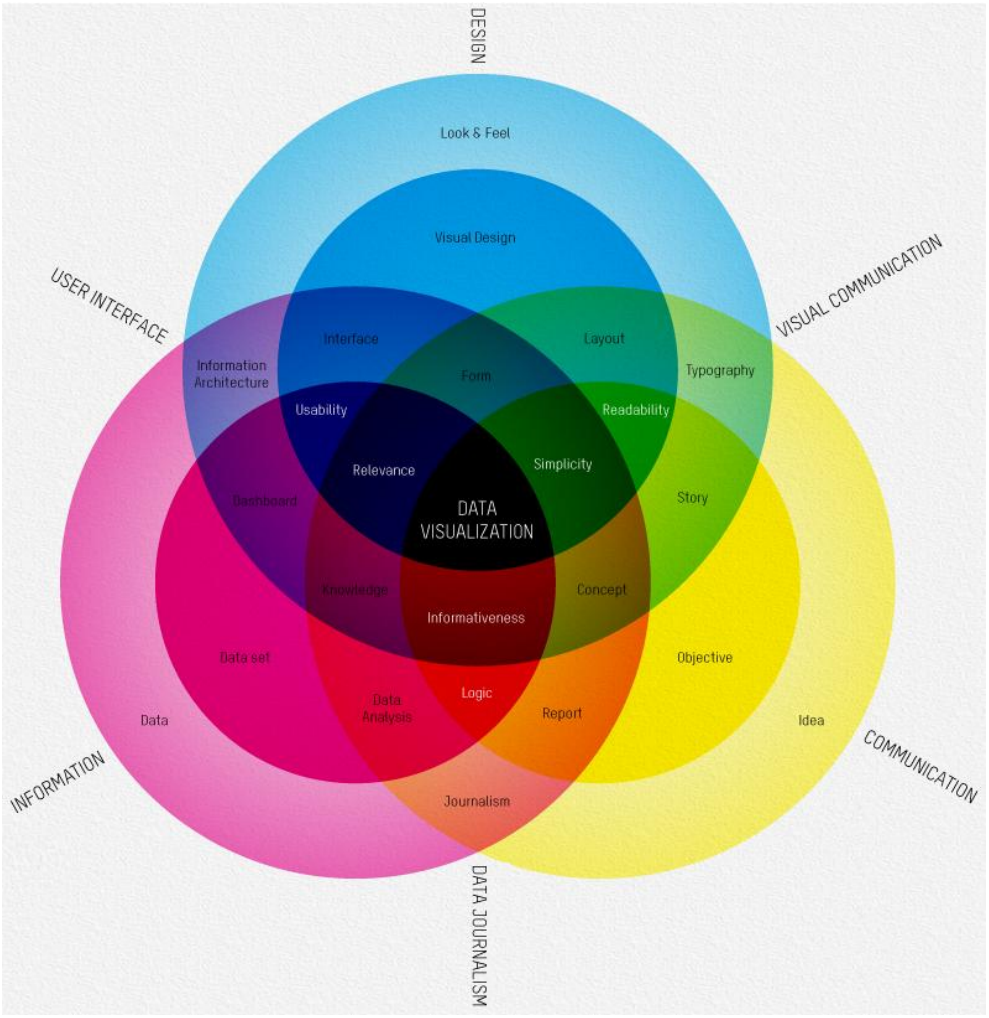


Image source: ffunction Inc. (2012)

Figure 9 *A multilayered Venn diagram showing the relations of the different elements and fields that contribute to and make up data visualization.*

2.6.2 Representation

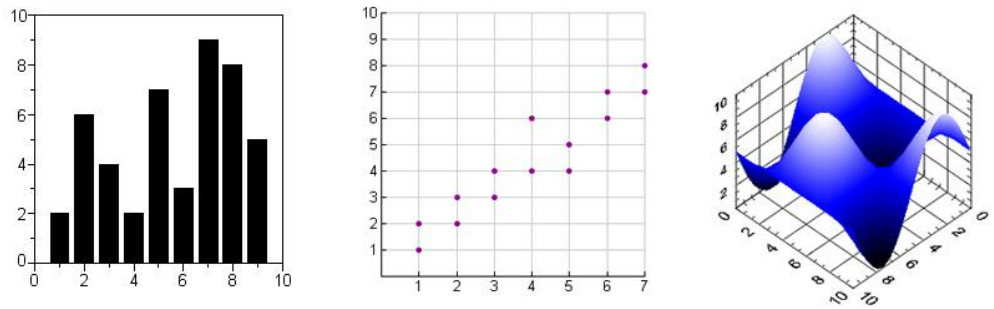
Representation is the visual encoding of data, of which Spence (2007) identifies three important aspects; the type of data to be represented, the complexity of the data and how a user interprets the encoded data. User interpretation of the encoded data has to do with aspects of human behaviour, memory and visual cognitive capabilities, and even though it is most common to represent data visually through graphics (as shown in the examples below), it is important to remember that data can also be visualized through sound and other human sensory modalities.

2.6.2.1 Data types and data complexity

When representing data types one needs to consider the relation between two or more items of the data, as well as how to represent the values and derived values, such as a range or an average (Spence, 2007). The complexity of the data is related to the number

of attributes of the data, most commonly referred to as dimensions, and as the number of dimensions increase so does the challenge of representing the data. Listed below are some representation techniques suitable for different complexities of data.

One-dimensional data can be represented as single values, such as by a number, a dial, a slider or a histogram (see FIGURE 10(a)). Two-dimensional data is usually represented by a scatter plot (see FIGURE 10(b)), or by line graphs, histograms or bar charts. It is also very common to use a time series plot, which is a special case of the scatter plot with one axis representing time and the other some function of time. Three-dimensional data can be represented by different kinds of 3D representations (see FIGURE 10(c)) or a scatter plot matrix.



(a) *histogram* (b) *scatter plot* (c) *3D graph*

Figure 10. *Some examples of representations of data with different complexities.*

Multidimensional data can also be represented by a scatter plot matrix, but as the number of attributes increase so does the possible pairs of attributes in a very rapid non-linear fashion. Multidimensional data is thus more commonly represented by star plots (see FIGURE 11), which are good for object visibility, and linked histograms, mosaic plots (see FIGURE 12) or parallel coordinate plots, which are good for showing attribute visibility. Object visibility has the property of representing each object as a single coherent visual entity, such as a point, which is desirable when a user wants to know the values of an object's attributes in many different dimensions and how objects relate to each other (Spence, 2007).

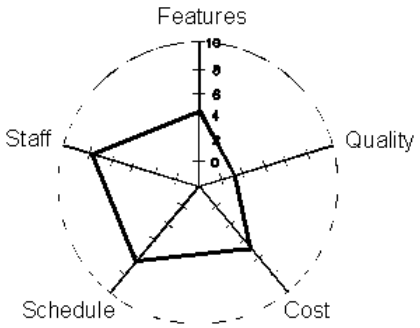


Image source: http://www.processimpact.com/articles/principle.html

Figure 11 *A star plot showing the attributes of a quality-driven application.*

Attribute visibility has the property of clearly visualizing the distribution of objects' attribute values in each dimension, thus also revealing clusters, which is desirable when a user wants to know how objects are distributed in an attribute dimension.

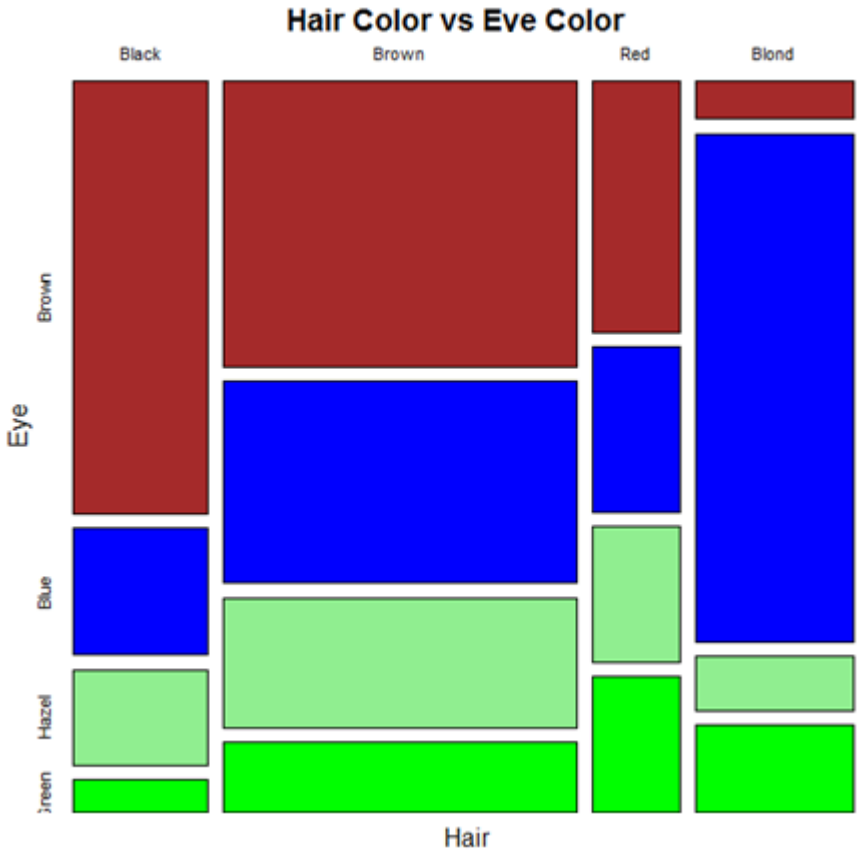


Image source: *http://www.analyticsworld.net/2010/04/22/data-visualization-example-1-mosaic-plot/*

Figure 12 *A mosaic plot showing the relation between hair color and eye color*.

The relation – the association or connection – between two or more objects can also be visualized by the use of lines. Drawing straight lines between the objects is a very simple and powerful method, and arrows, symbols, color, line thickness, line style, line annotation and relative position may also help represent and clarify the relations depending on the type of data Association charts, such as the one used in FIGURE 4 and timeline charts are two good examples using many of these techniques. Maps, diagrams and tree representations, such as cluster maps (see FIGURE 7), Venn diagrams (see FIGURE 9), cone trees and tree maps (see FIGURE 13) can also be used to represent relation (Spence, 2007). The interested reader is referred to Spence (2007) for further details.

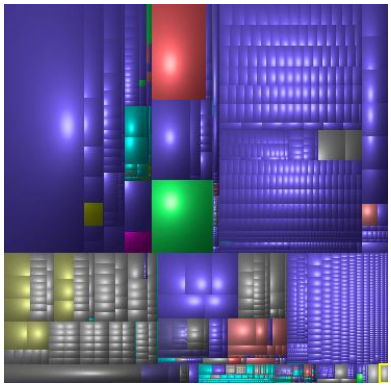


Figure 13 *A tree map showing the hierarchy, amount, size and (color-coded) types of files located in a computer directory.*

2.6.2.2 Data encoding methods and guidelines

There exist a large number of recommended encoding methods and sets of guidelines for different kinds of representations depending on the type and complexity of the data, of which most are based on both research and evaluation. Only some of the more general encoding methods and guidelines are presented below in short form.

Jacques Bertin, a French pioneer in information visualization, identified four tasks that are common to information visualization and the encoding methods he considered best suited for these tasks. He considered encoding by size, value, texture, color, orientation and shape when dealing with the common tasks of visualizing association, selection, order, and quantity (Bertin, 1983).

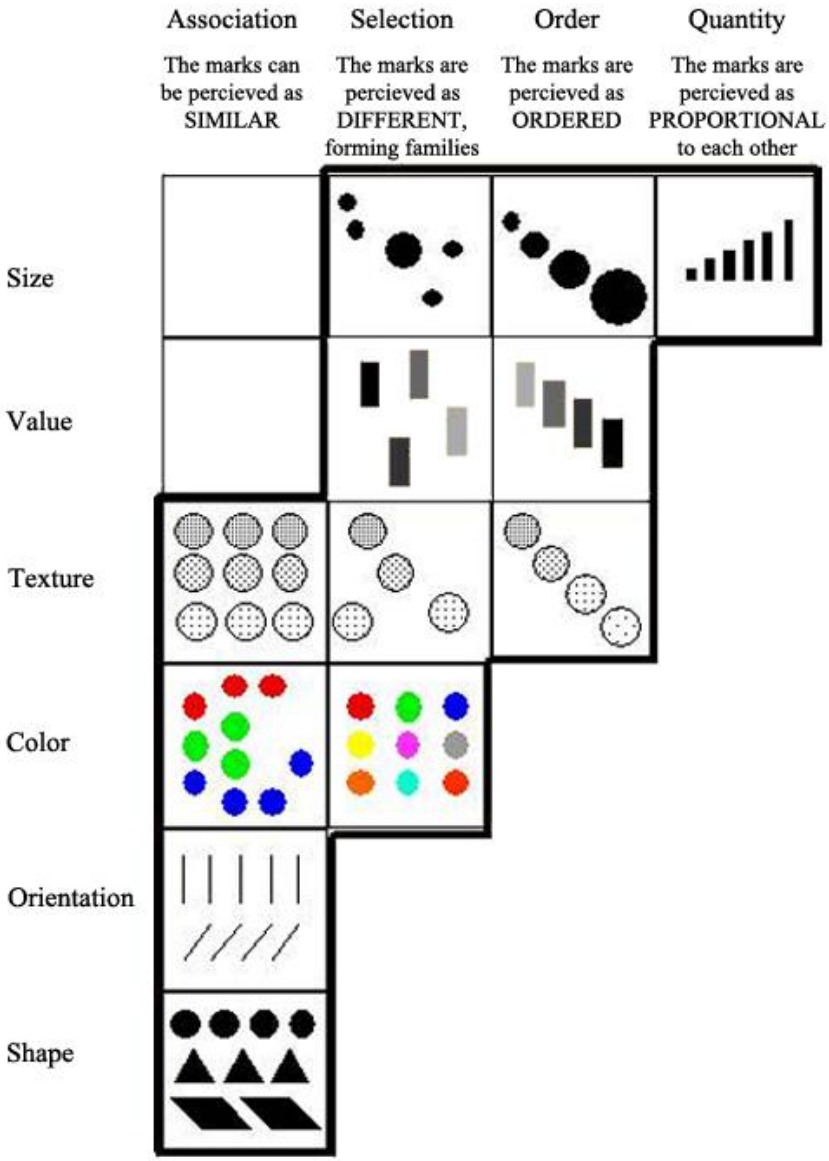


Image source: Spence (2007)

Figure 14. *Bertin's guidance regarding the suitability of various encoding methods to support common tasks in information visualization.*

FIGURE 14 shows Spence's (2007) interpretation of Bertin's conclusions. Spence also points out that the best suited encoding method for a task is very dependent on the context, and that the conclusions can be considered no more than guidance. The "marks" in the figure refer to the result of the encoding method that is used. The difficulty of accurately evaluating values depends on the encoding method used. In this aspect Spence (2007) again provides guidance for which encoding methods to use when encoding quantitative, ordinal and categorical data, summarized in TABLE 1 below.

Table 1. *Guidance for the encoding of quantitative, ordinal and categorical data. The methods are listed in descending order of accuracy for evaluating the encoded values.*
Table source: Spence (2007)

Quantitative	Ordinal	Categorical
Position	Position	Position
Length	Density	Color hue
Angle	Color saturation	Texture
Slope	Color hue	Connection
Area	Texture	Containment
Volume	Connection	Density
Color	Containment	Color saturation
Density	Length	Shape
Shape	Angle	Length
	Slope	Angle
	Area	Slope
	Volume	Area
		Volume

Images of visualized data, which is one of the most commonly used visualization mediums, are sometimes also referred to as information graphics. These are defined by Brath (1999) as highly specialized images that transform quantitative, categorical, or relationships data into understandable images. Tufte (2001) provides a set of goals that an effective information graphic should strive to:

- show the data,
- make the viewers think about the substance,
- avoid distorting the data,
- present many numbers in a small space,
- make large datasets coherent,
- encourage comparisons of different pieces of data,
- reveal data at several levels of detail,

- serve a reasonably clear purpose of exploration, description, tabulation or decoration, and
- be closely integrated with the verbal and statistical descriptions of the dataset.

To fulfil these goals he also recommends adhering to the following set of principles:

- showing the data and showing as much data as possible,
- removing non-data information and reduce redundant data information as much as possible,
- avoiding perceptual junk, such as moiré patterns, grids and outlines,
- making elements multifunctioning, such as making labels and grid lines data dependent,
- increasing data density by shrinking the area used for the graphic,
- using multiple instances of graphics to facilitate visual comparisons,
- using words, numbers and graphics together,
- providing a narrative quality,
- using words in full with standard orientation,
- avoiding legends by placing labels directly on the graphic,
- using color carefully to highlight the most important information and to separate different classes, and avoiding common color blindness combinations, and
- separating different classes of information into layers, such as separating the data foreground from the structural background.

As previously suggested, the use of color can also be very beneficial and important if used correctly. Since color and luminosity are both subjectively and contextually interpreted, they must be handled with care within a visualization. On this, Tidwell (2011) points out that you should never use red versus green as a critical color distinction due to the fact that color-blind users will not be able to see the difference, and to never use complementary colors together, since the human eye cannot separate them. Perceived contours and optical illusions of bleeding colors or incorrectly perceived shade and hue may also occur when some colors are mixed or used in close vicinity (Spence, 2007). Chromostereopsis is such an unwanted three-dimensional depth effect that may occur when mixing red and blue two-dimensional objects (Brath, 1999), as exemplified by FIGURE 15. Color should thus not carry the primary information and be restricted to discrete values, but according to Brath (1999) the effects of good use of color in graphic information displays generally improves the performances in recall task, in search and locate task, in retention task, in decision judgement, as well as improving the comprehension of instructional materials. Although, using color as a means of identification does not work for small objects, as it may not be recognizable at all.

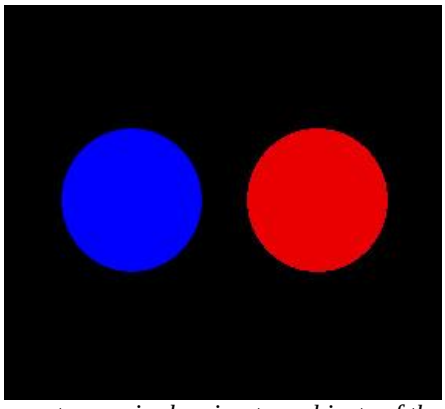


Figure 15. *An example of chromostereopsis showing two objects of the same shape and size. To most people the red circle appears closer to the front than the blue circle*.

Further, it is recommended to use certain common color combinations, such as red, yellow and green to represent danger, caution and safe respectively. Cultural differences may interfere with the meaning of these colors (Tidwell, 2011), so one could use military standards instead, where red indicates inoperation, flashing red indicates emergency conditions requiring immediate action, yellow indicates caution or delay, green indicates a fully operational system, white indicates explanation and blue indicates advice (Brath, 1999).

2.6.3 Presentation of represented data

Presentation is the selection and layout of the encoded data in the available display area and the available time. This means that one must decide which part of the representation that should be displayed, if and how it should be interactive, and even whether something should be displayed at all. The available display area and the available time restrict this as well.

2.6.3.1 Display limitation techniques

One of the most basic issues is that of displaying a document that is larger than the display area. This is usually solved with scrolling, but it is important to remember that most of the content will be hidden from view, which can make it hard for the user to keep track of where things are located and make it hard to find their way back to the current position after searching through the content. This can greatly be avoided by using a technique called overview plus detail, where two separate views are used to show an overview with a context and a more detailed view respectively (Spence, 2007; Tidwell, 2011). FIGURE 16 shows one example of this with smaller thumbnails showing the pages of a document alongside a full page view.

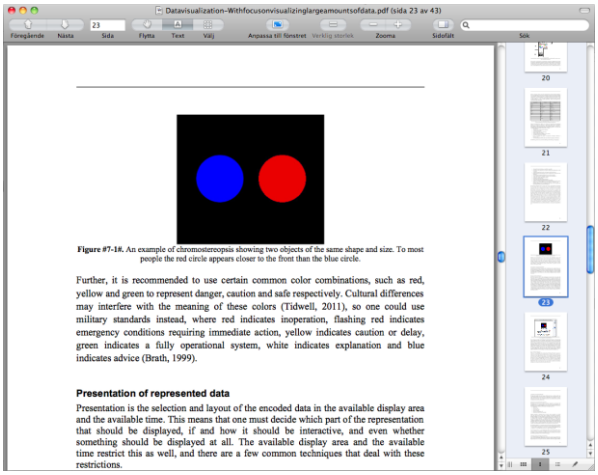


Figure 16. *An example of the overview plus detail design pattern showing a pdf-document in the Preview application running on Mac OS X*

A similar problem, referred to as the focus plus context problem, where there is a need to smoothly show the connection between the detail and overview presentations, can be solved with the use of the distortion or suppression techniques, or a combination of the two. The techniques are rather self-explanatory, but the interested reader is referred to Spence (2007).

Zoom and pan is another useful technique for utilizing the display area when showing large visualizations with a lot of details. Panning allows the user to smoothly and continuously move the view location. Zooming in hides non-essential data and may display more or magnify relevant data in a more manageable view. Zooming out may reveal hidden data and also allows the user to rediscover their location in the information space, as well as recalling context and integrating new context into their mental model. Semantic zoom is often used instead of geometric zoom to allow different levels of detailed representations to be displayed rather than just magnifying one single representation. Semantic zoom thus allows the user to focus on the more important attributes first and reveals data about other attributes (Spence, 2007). A typical example that uses this technique is basically any interactive digital map.

2.6.3.2 Time limitation techniques

There are some techniques for managing time limitations that have been proven successful. One is called rapid serial visual presentation, a technique where each item of a collection of visual representations, such as a set of images, are presented separately and at the same location for a short period of time, typically 100 milliseconds each. This gives the user a high probability of detecting a sought-after item, or detecting the absence of it (Spence, 2007). There are also some techniques involving different groupings and patterns of movement of the visual representations, mentioned by Spence (2007). These will not be discussed, since none of these available time limitations techniques are of any real benefit to the work of this thesis.

2.6.4 Interaction with presented data

A single all inclusive view of a large collection of data can seldom provide any insight without further exploration. That is why being able to change the view of a large collection of data is one of the greatest benefits of using computers to support information visualization. One can define interaction as the available means that exists to change the view of the available data.

2.6.4.1 Exploration and insight through interaction

The progression from one view of data to another is referred to as movement in the information space, or simply navigation. To be able to better support navigation one must help the user to answer the typical questions asked at each step of the interaction. According to Spence (2007), the following are such typical questions:

- where am I,
- where can I go,
- how do I get there,
- what lies beyond,
- where can I usefully go, and
- where have I been (I want to go back).

Efficient navigation is important because in general, tasks or problems to be solved most often lack precision and has vague, ambiguous and subjective requirements. Therefore problems are often formulated and specified as they are being solved. On this Spence (2007) notes that providing navigational guidance by indicating the result of a changes made to parameters – a technique called brushing – is a simple way to help the user navigate and explore the information space. This helps the specification of problems by displaying the sensitivity of the parameters and thus revealing potential possibilities that the user might not have considered otherwise. A detailed discussion and guide on how to answer the questions above and how users reason when they navigate is provided by Spence (2007) and recommended to the interested reader.

The brushing technique is an extremely useful and heavily used visualization technique that can be applied to most interactive visualizations. Spence (2007) defines brushing as *"a change in the encoding of one or more items essentially immediately following, and in response to, an interaction with another item*" (Spence, 2007, pp. 235).Some examples of this is to only highlight the objects with selected attributes from a collection of objects and attributes, or updating the result of a query when a parameter is changed, as well as the mouse-over effect, such as the tooltip text, that exists in practically every software and on all modern web pages.

2.6.4.2 Information spaces and types of interactions

Spence (2007) identifies four different types of interaction and two types of information spaces, all presented briefly with the aid of examples. Looking at Minard's map (in FIGURE 6) can be seen as passive interaction, where the mind of the viewer and its cognitive abilities does the actual interaction. The World Wide Web is an example of stepped interaction; the user navigates the views of a discrete information space, which requires both interpretation and a decision to be made by the user to continue at each step. A very responsive system is desired when navigating a continuous information space to support efficient exploration and to be able to quickly gain insight to the result of every change through continuous interaction. The last and most efficient type of interaction, called composite interaction, combines two or more of the other types.

2.6.4.3 The Action Cycle and human aspects of interaction

Norman's Action Cycle consists of the Seven Stages of Action needed for a human to achieve a goal, as shown in FIGURE 17. It helps the interface designer to see things from the user's point of view and functions as good support when designing and evaluating interfaces of all kinds (Norman, 1988).

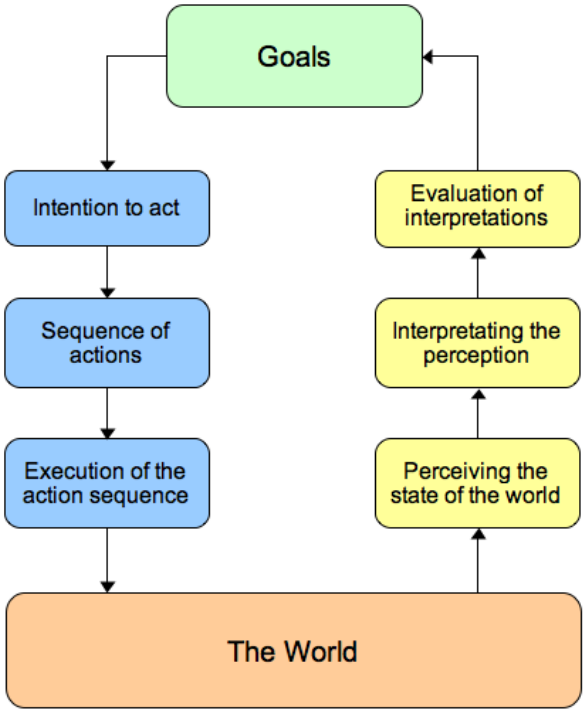


Image source: Reproduction of Norman (1988)

Figure 17. *The Action Cycle, presented by Donald Norman in 1988.*

The seven steps consist of one step for forming the goals; three steps for execution: forming the intention, specifying actions, and executing the actions; and three steps for evaluation: perceiving the state of the world, interpreting the state of the world, and evaluating the outcome. Reality is of course never that simple, and Norman points out that it is an approximate model, that the steps are not discrete entities, most behaviour

does not need to go through all the steps in sequence, and that there must be many sequences leading to sub-goals and sub-intentions, or even new goals, based on the feedback of previous steps and sequences. He also calls the three steps of execution the Gulf of Execution to illustrate the problem of knowing what to do in the specific context to change the state of the world. Similarly, he calls the three steps of evaluation the Gulf of Evaluation, illustrating the problem of detecting and evaluating changes to the state of the world in the current context (Norman, 1988).

Falling under the Gulf of Evaluation are two important human phenomena called inattentional blindness and change blindness, which have to be carefully considered when interaction is available in the visual representations and presentations. Inattentional blindness occurs when a person focuses on a specific piece of information and as a result fails to notice some other obvious piece of information. Spence (2007) gives an example of this with an experiment where people who were asked to watch a short video of a group of people tossing a ball back and forth, counting the number of times one of the girls touches the ball, completely missed a man in a gorilla suit appearing in the video and fooling around in the middle of the group. Change blindness occurs when a person is slow or completely fails to notice (sometimes large) changes between two otherwise similar visualizations. For example the change can be that of added or removed objects, or change in color or luminosity. FIGURE 18 shows an example of potential change blindness presented by both Enns (2011) and Spence (2007). The risk of change blindness also increases with the complexity of the data. These phenomena must be accounted for, and could break a design and even turn out to be fatal in critical real-time systems, since visualization strongly involves the user's attention to what is being displayed and to what changes.

Image source: Spence (2007)

(a) *original image* (b) *one object is missing*

Figure 18. *An example of change blindness. The two pictures are different (one object has been removed), but the change can be hard to notice at first glance. Can you spot the difference?*

The use of animation over instant change is recommended to minimize change blindness, give the user a better chance of perceiving the change and to better support and expand the mental model of the user. Visual cues and encoding to counteract inattentional and change blindness must be clear, pop-out and also be balanced against the increased strain and workload these additions may put on the user (Spence, 2007).

2.7 Graphical User Interface

A graphical user interface (GUI) presents the internal encoded representation of information and data to the user through windows, panels, text, images, icons, buttons and other visual elements. The user interface is limited by and dependent on the available screen real estate and space limits of the platform and presentation medium. For instance, a computer monitor has much more screen real estate to work with than a smartphone. It is thus of course very important to design the GUI for the intended platform, but what is an even more important aspect, and generally accepted as the most important aspect, is the actual user.

User interfaces should be designed to provide an engaging and enjoyable experience while not requiring too much effort from the users to operate effectively. A user interface must also be designed in such a way that they do not make users feel stupid or cause them to make big mistakes (Cooper, 2007). To be able to create a good, effective and satisfying GUI the designers must truly understand and design for the users, while also keeping in mind that they themselves rarely represent the intended users (Cooper, 2007). Identifying the intended type of users and the intended posture of the application is always a good place to start.

2.7.1 User type and application posture

Users can be divided into the three different groups of beginners, intermediates and experts, depending on their vocabulary, level of knowledge and experience. This division, like most population distributions, tend to follow the statistical bell curve, and most users thus tend to be intermediates (Cooper, 2007). All users start out as beginners, but as beginners are incompetent by definition, and no one likes being incompetent, they do not stay beginners for long, and few become experts. You should therefore, according to both Cooper (2007) and Tidwell (2011) design and optimize for intermediates. The openness of the GUI must also be balanced depending on the intended user type, and it comes down to how much effort the users are willing to spend to learn the interface. If the GUI is too open it will confuse and deter beginners and occasional users and if it is too closed it will cramp and restrict intermediate, expert and returning users (Tidwell, 2011).

Cooper (2007) defines three different postures of applications; sovereign posture, transient posture and daemonic posture. Applications that fall under the sovereign posture are usually bigger, often full screen applications, such as word processors and e-mail applications, that demands much of the user's time, the user's full attention and that the user learn the applications well. To support the goals and tasks of the user, as well as to lessen the cognitive load, they should be optimized for full screen use, have a conservative visual style and exploit rich input. Applications that have a transient posture are launched, used briefly and then dismissed, such as widgets and music players. They must be clear, simple and spot on, and preferably retaining its previous position and settings between launches. Applications that do not normally interact with the user, such as background processes (daemons), are said to have a daemonic posture (Cooper, 2007).

2.7.2 Flow and excise

People experience a state of mental ergonomics, referred to as flow, when they are able to focus and concentrate fully on an activity to such a degree that they lose awareness of peripheral problems and distractions. Flow is often associated with a mild sense of euphoria and unawareness of the passage of time, and a person in this state can be very productive, especially when performing constructive activities such as writing, designing or development (Cooper, 2007). To make the users happier and more productive the GUI should thus be designed to be responsive and promote, support and enhance flow. It is likewise important to try to avoid behaviour in the GUI that may disrupt the flow to make sure that it does not become difficult for the users to be productive. One way of doing this is by making the application more intelligent by simply having it derive as much as possible instead of asking the user, thus reducing the number of distracting dialogs, warnings and confirmations that the user have to encounter and deal with. Keeping related things together to avoid forcing the user to switch context when doing related tasks is another way of supporting flow and minimizing disruption. Minimizing excise is probably also one of the best ways to help support flow.

Excise are tasks that do not directly contribute to reaching the goals of a user, but are still necessary to accomplish the goals. It is the extra work required to meet the needs of the tools or outside agents that are used to work towards the actual goals (Cooper, 2007). Actually eliminating all excise is almost impossible, depending on what context and definitions are used, since everything short of reaching the goal without any effort or interaction can be seen as excise. One example of excise is the combined use of panning and zooming, which can make it difficult for a user to navigate without getting lost in the information space. Reducing the number of places to navigate to, minimize scrolling and providing signposts or overviews are good ways to reduce this type of excise. There also exist visual excise, which can be minimized by avoiding excessive ornamentation of the GUI, by encoding data in good ways to make it easy for the user to decode the visual information and by using metaphors carefully and sparingly.

To summarize, well-designed user interfaces are transparent, since putting focus on the interaction itself interferes with the users' goals and flow. Cooper (2007) also summarizes this with brutal honesty: *"No matter how cool your interface is, less of it would be better."* (Cooper, 2007, pp. 202). The curious reader who wants to know more is referred to Cooper (2007), who has a lot of interesting and insightful things to say about these subjects.

2.7.3 Design patterns, heuristics and guidelines

To help with designing graphical user interfaces there are a lot of design patterns, heuristics and guidelines available; Tidwell (2011) alone defines over a hundred different design patterns, and Cooper (2007) presents many interaction design patterns and processes, as well as 117 interaction design principles that cover broad ideas, rules and hints about the practice of design and use of interaction design idioms. Spence (2007) summarizes this quite nicely when he states that *"There are no step-by-step*

instructions which, when followed, will guarantee success or even an adequate result. The reason is simple: every new interface is different and usually complex; there are many requirements to satisfy (social, technological, financial, to name a few); available technology is changing with time and many different sources of expertise are drawn upon - cognitive psychology, visual design, the list is long. Design is largely a creative process and there is no 'silver bullet'." (Spence, 2007, pp. 181).

Due to general consensus on what makes a good user interface, it is unavoidable that the guidelines and design patterns of the different authors overlap with each other and with many other guidelines and recommendations regarding the subject. Some of the more general heuristics and guidelines, which every interaction designer and interface designer should know by heart, are presented below:

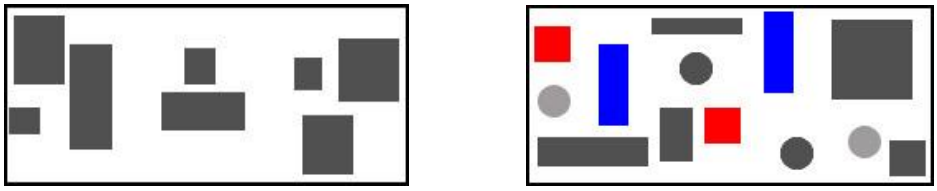
- **Design for consistency and follow standards**, by following standards and platform conventions, and by keeping the conceptual model, and meaning of semantics, syntax, situations, actions and display formats uniform and consistent (Brath, 1999; Nielsen, 1994). Use standard symbols, words, icons and metaphors whenever possible. The use of metaphors should always be handled with care, and preferably be minimized since they are hard to find, they scale poorly, and there simply do not exist enough good metaphors to go around (Cooper, 2007). They also constrict our thinking and users' ability to recognize and correctly interpret them is often questionable, especially across cultural boundaries (Spence, 2007). Avoid any exceptions or special conditions that break the consistency of the design.
- **Structure the display**, by dividing the view into different areas to present different types of information. The user's perception of structure can be enhanced by using encoding such as color, shape, line style, intensity and grouping (see section 2.7.4 GESTALT GROUPING PRINCIPLES below). Combining these also helps to further separate and directing attention to pieces of information (Brath, 1999).
- **Strive for aesthetic and minimalist design**, by keeping the information shown minimal, visible, legible and relevant, since irrelevant and extra information competes for space, visibility and relevance (Nielsen, 1994).
- **Match between system and the real world**, by letting the system use words, phrases and concepts that are familiar to the user, rather than using system-oriented terms, and by following real-world conventions, making information appear in a natural and logical order (Nielsen, 1994).
- **Feedback and visibility of system status**, by always keeping users informed through appropriate feedback within reasonable time (Nielsen, 1994). Give feedback by, for example, highlighting buttons when pressed, changing the mouse cursor to a busy icon, using prompts, or displaying error messages. Feedback and error messages should be positioned close to the task in the work area or cursor to maintain a visual continuity, rather than forcing the user to switch back and forth between the work area and a feedback area (Brath, 1999).

- **Control response time**, by relating response times to user expectations; simple immediate interactions must have a response time shorter than 100 milliseconds (Brath, 1999). Response times up to about 1 second make the user feel that the system is responsive. Up to 10 seconds make them feel that the system i slow, so make sure to provide a progress bar. After 10 seconds the users will lose interest and start doing something else outside the system. Such slow processes should thus preferably run in the background to allow the users to continue using the system for other tasks, while displaying a progress bar with estimated remaining time and allowing the users to cancel these processes if needed (Cooper, 2007).
- **Allow user control and freedom**, and accommodate errors, by having clearly marked escape hatches and supporting undo and redo. This encourages the users to explore the system (Nielsen, 1994). Tidwell (2011) also points out that one should always let the users be in control, or at least feel that they are in control.
- **Help users recognize**, diagnose and recover from errors, by expressing visible, useful and understandable error messages in plain language instead of only error codes, clearly stating the problem and constructively suggest a solution to the problem (Nielsen, 1994). The effects of ignoring a warning should also always be clear to the users (Brath, 1999).
- **Error prevention**, by eliminating error-prone conditions or by letting users confirm actions that may lead to such conditions, which is of course better than having to use error messages (Nielsen, 1994). It is also true that one should avoid breaking the flow of the user with unnecessary error, notification or confirmation dialogs.
- **Help users learn the system**, for example by providing help or tutorials for novice users, and by using prompting, such as highlighting and dimming actions and choices, to help intermediate users (Brath, 1999).
- **Provide help and documentation**, which are not too large, easy to search, focuses on the task of the user and lists clear and concrete step-by-step instructions. It is even better if the system can be used without any documentation at all (Nielsen, 1994).
- **Flexibility and efficiency of use,** by the use of accelerators and by allowing users to tailor frequent actions (Nielsen, 1994).
- **Recognition rather than recall**, by minimizing the user's memory load through good visibility of objects, actions, options and instructions (Nielsen, 1994). Avoid, as much as possible, to force the user to memorize information such as shortcuts, commands or legends (Brath, 1999). Provide memory aids and display relevant information to the user instead.

2.7.4 Gestalt grouping principles

Grouping and aligning related objects together also helps improve the interface, helps to keep the design consistent, as well as helping with keeping the display and tasks structured. Grouping normally follows the Gestalt principles, where Gestalt theory dictates that to perceive a figure it must first be separated from its background by the human perceptual organizational processes. To make a figure easier to perceive it should clearly stand out from the background, which may be aided by the following observations; a larger region, or a region that completely surrounds another smaller region tend to be seen as backgrounds, while surrounded regions and smaller regions tend to be seen as a figures, as well as tending to perceiving symmetric, vertical and concave regions as figures compared to asymmetric, diagonal, and convex regions respectively. The most important Gestalt grouping principles that humans tend to perceive, according to Brath (1999) and Tidwell (2011), are listed below. Combining these with sufficient whitespace or group boxes (a type of common region) may also help crowded GUIs, although it is better to avoid this and display less information with more clarity instead. Notice how the figures below use whitespace and group boxes to separate the different examples from each other, thus using the grouping principles themselves.

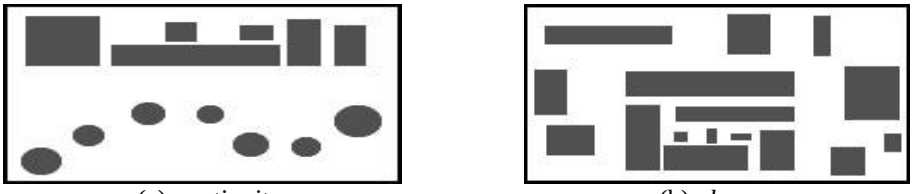
- **Proximity:** objects that are put close together will be associated with one another (see FIGURE 19(a)).
- **Similarity:** groups are formed by objects that look similar, for example by sharing shape, color, size or font (see FIGURE 19(b)).



(a) *proximity* (b) *similarity*

Figure 19. *Illustrated examples of the Gestalt grouping principles.*

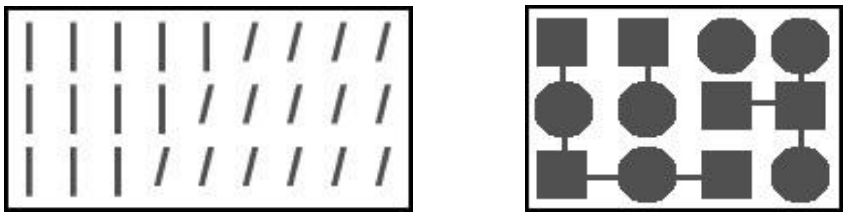
- **Continuity:** relative positioning and alignment of smaller objects form groups of continuous shapes and directions, such a line perceived from a series of dots (see FIGURE 20(a)).
- **Closure:** simple closed shapes, such as rectangles and circles, are perceived even if there are gaps in the placement of objects in the group (see FIGURE 20(b)).



(a) *continuity* (b) *closure*

Figure 20. *Illustrated examples of the Gestalt grouping principles.*

- **Common orientation:** objects with a common motion or common orientation will tend to be grouped together (see FIGURE 21(a)).
- **Connectedness:** objects connected with lines will be perceived as grouped (see FIGURE 21(b)). Compare with continuity.



(a) *common orientation* (b) *connectedness*

Figure 21. *Illustrated examples of the Gestalt grouping principles.*

- **Common region:** objects surrounded by a common background will be perceived as grouped (see FIGURE 22).

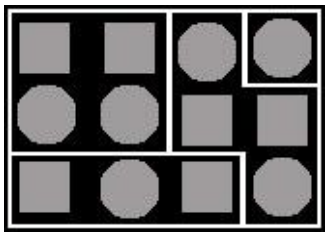


Figure 22. *Illustrated example of the common region Gestalt grouping principle.*

3 Method

This chapter describes and motivates the choice of methods used during the entire span of the thesis. See chapter 6 DISCUSSION for thoughts on how these choices and methods impacted the final result of the thesis work.

3.1 Research methods

The sources used to gather information and material were mainly those of books from Chalmers Library (2012) and Google Books (Google Inc., 2011), both physical and electronic distributions; articles, papers and publications from online databases, such as ACM Digital Library (Association for Computing Machinery, 2012) and IEEE Xplore Digital Library (IEEE, 2012); Internet in general, specifically through Google search with related keywords and phrases such as: data visualization, algorithms, algorithm complexity, memory management, graph plotting, Qt, python bindings, iterative development process, etc.

3.2 Prestudy methods

After the initial gathering of source material, a smaller prestudy of similar existing data visualization libraries was made. Comparisons of functionality and response time, for large amounts of data, was done between the libraries to get an overview of their capabilities, and to acquire a set of (very rough) performance benchmarks to compare with the final result of the thesis. The performance benchmarking was done by repeatedly timing performed actions during a stress test, and considering the response time of these actions expected by users. The results were quantified, based on the observations of Cooper (2007), as fast; 100 milliseconds or less, responsive; up to 1 second, slow; 1-10 seconds, and unusable; more than 10 seconds. Further details of the examined libraries and test results can be found in APPENDIX C - PRESTUDY RESULTS.

A large amount of data, namely 25 million points, for stress testing was created using the following Python code:

```
import numpy as np
 x = np.arange(0, 250000, 0.01)
 y = np.sin(x)
```

That amount of points in the test data was chosen so that test notebook (see section 1.4 DELIMITATIONS for specifications) would have to struggle, and even fail in some cases, as it generates the potentially massive overhead of managing and converting every point of the data to internal objects and data structures stored entirely in RAM, before displaying the data. 25 million points should thus test the limits of the studied libraries, even though the size of the generated data was only about 400 megabytes rather than in the range of a couple of gigabytes. This test data was also chosen to showcase any potential improvements where the developed library could (hopefully) manage and display the dataset more efficiently than the existing libraries.

The following questions regarding functionality, which were derived from the requirements found in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS, were answered, are answered in APPENDIX C - PRESTUDY RESULTS:

- **Which license is used?** The license recommended was BSD and preferably using a library should not impose any restrictions on a commercial product.
- **Which platforms are supported?** Ideally all platforms supported by Python, but primary Windows, Mac and Linux.
- **Are all graph types supported?** The two primary graph examined types are scatter plot and time series with error bars.
- **Which rendering methods are supported?** Many libraries support more than one rendering backend, sometimes also with different GUI toolkits. This could have an impact on performance and usability.
- **Is there any interactive functionality built into the GUI?** Capabilities to pan, zoom and select data were important requirements.
- **Are all data points stored in memory?** Attempting to keep large datasets in memory is undesired because the system can become unusable for longer periods of time as described in section 2.1 OPERATING SYSTEM MEMORY MANAGEMENT.
- **Are data points compressed?** Useful for reducing the above problem, but is by no means a silver bullet to keep large arbitrary datasets in memory. More generally it is possible to find a dataset with high information entropy, which causes any attempt to compress the data to be less efficient (Rezakhanlou, 2007).
- **How is basic library API usage?** APIs provide functions from a library by name and arguments, and it is of course easier to learn a new API if it follows some common convention or is similar to a known one
- **Are Python bindings provided?** A less critical issue is whether or not Python-bindings exist, because bindings can in most cases be written with little effort using automated tools such as SIP.
- **Which scaling methods are supported?** Linear and logarithmic scaling methods are required.
- **Are there any preprocessing capabilities?** Interpolation and similar for high quality time series and automatic error bars. Not as important since it can be precomputed and appended as part of the related dataset, before rendering.
- **Is it multithreaded?** Concurrent rendering can be used to avoid blocking the GUI from long delays while rendering. See section 2.3 CONCURRENT PROGRAMING AND MULTIPROCESSING.

3.3 Development method and software libraries

The method used during the thesis to develop the proposed library was an iterative development process, using the C++ programming language, POSIX Threads (IEEE, 2008), Qt libraries (Nokia Corporation, 2012) and the AGG library (Shemanarev, 2006). Python bindings were provided through the SIP and PyQt libraries (Riverbank Computing Limited, 2012). CMake (Kitware Inc., 2012) was used to generate platform-specific build scripts for the library source code.

Unstructured interviews with staff and developers from Combine and Volvo yielded the initial requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS. To summarize, two basic graphs were deemed necessary; a regular scatter plot and a time series graph. Functionality included point selection, grouping, configurable scales and zoom support, error bars were requested for the time series, and semi-transparent dots for the scatter plot in order to visualize the points cluster density. A graph widget example implementation prototype built upon the proposed API, acting as both a technical demonstration application and test application, was developed in parallel with the API to be able to show a more tangible result that could be evaluated and compared to the purpose, prestudy results and API requirements.

3.3.1 Iterative development process

The iterative development process, depicted with its steps in FIGURE 23, was chosen over more traditional processes, such as the waterfall model's "do it right the first time" mentality. The motivation for choosing this was because of the complexity and uncertainty of how to design and implement the proposed library efficiently, due the limited knowledge and experience with selected libraries and frameworks, as well as the added difficulty of actually designing an efficient and useful API rather than an application. Iterative development also inherently supports modular and modification-friendly design, structure and code, which goes hand in hand with a well-designed API. This is further motivated by Sotirovski (2001), who states that increased uncertainty calls for an iterative development process, of which the purpose is to discover pitfalls that may otherwise not have been foreseen. Iterative development also allows mistakes and weaknesses from previous iterations, in both design and code, to be found and corrected as soon as possible (Sotirovski, 2001).

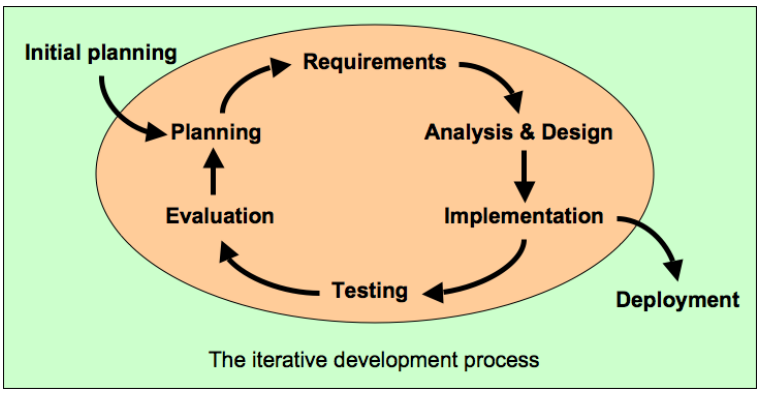


Figure 23. *An overview of the steps making up the iterative development process.*

The iterative development process consists of a step for initial overhead planning, followed by as many iterations as is deemed necessary (usually three or more), and finally deployment of the finished software. Each iteration is made up by steps for planning the tasks for the iteration, selecting requirements and tasks to implement in the current iteration, analysis and design of the selected requirements and tasks, implementation of the resulting design, testing the implementation, and finally evaluation of the iteration.

Rather obvious as it is, Sotirovski (2001) recommends developing the architecture and critical core functionality first, as this will focus on and establish the infrastructure of the necessary interactions between the system components. This foundation will act as a framework and decide the outcome of the rest of the system, since having the framework in place will focus the effort, boost confidence and provide better insight in how to implement subsequent components of the system. Focusing on implementing framework code with wide and shallow integration also reduce rework and mitigates risks in areas where requirements and design are still uncertain, as well as giving the problems that arise early attention. Implementing critical core functionality early on as well, will give it more time to mature and increase in quality as the development progresses. Once the framework is constructed, adding the other components usually involves less mutual dependencies, which is an advantage in future iterations, as it allows for more parallel development (Sotirovski, 2001). See chapter 4 DEVELOPMENT for more details on the actual use of the process in this thesis.

3.3.2 Programming language and software libraries

C++ was chosen since it is a low- to intermediate-level programming language, designed for object-oriented programming as well as providing more fine grained memory management functionality. POSIX Threads (Pthreads for short) is an open POSIX standard for threads (IEEE, 2008) with implementations available on all larger platforms and operating systems, thus making it suitable for use in the proposed library. On the request of Combine, the code style and structure defined by the Google C++ Style Guide (Weinberger, 2011) was also used.

Combine also suggested Qt (version 4), a large well-known and well-documented cross-platform application and UI framework, distributed under the LGPL license, used by many companies (Nokia Corporation, 2012). It provides the majority of the basic and most commonly used GUI components and elements across the major platforms, and is thus suitable for use as a GUI frontend component for the proposed library. It also provides rich event handling, a comprehensive signals and slots system, timers, as well as thread and synchronization support and internal memory management among many things. It should also be noted that the GUI and Qt components were all implemented by hand, even though Qt comes bundled with Qt Creator and Designer tools. The reason for this was to better get to know and learn Qt on a lower level, as well as avoiding GUI-builder tools, due to previous bad experiences with such tools used for anything other than prototyping. Although Qt is used to display all visible GUI components, the shown graph plot image is first rendered by an AGG-renderer to an image buffer in the

backend, which is then used by Qt. This allows the components to be separated and modular. AGG is a flexible open source stand-alone cross-platform (two-dimensional) graphics library, supporting transparent rendering, making it more than sufficient as a backend rendering component for the proposed library. AGG version 2.4 is used, since it is the latest version to be distributed under the BSD license, and all later versions are as of this writing released under GPL (Shemanarev, 2006).

To simplify implementing Python bindings, a tool named SIP was used to generate most of the necessary glue code required to use a C++ library in Python. Although the implemented API and library is to be distributed under a BSD license, the PyQt libraries used by the generated Python bindings are distributed under GPL. This forces the library bindings for Python to use GPL as well, creating an exception from the license requests from Combine. This exception was motivated and accepted by Combine, as they themselves use PyQt libraries in their own application development, and have had to make the same exception themselves. Finally versions 4 of both the SIP and PyQt libraries are used, since earlier versions of PyQt do not support Qt version 4.

3.4 Tools and collaboration methods

Pen, paper and whiteboards were used during the planning and design phases to sketch the structure and GUI, write pseudo code for algorithms, and to generally solve encountered problems. The NetBeans (Oracle Corporation, 2012) and Code::Blocks (The Code::Blocks Team, 2011) IDEs were used as basic tools during the development in combination with or without SVN-plugins, debugger and command line tools. The Valgrind analysis framework (Valgrind Developers, 2011) was used with the Valkyrie GUI frontend to find and identify memory leaks (see section 2.1.1 VIRTUAL MEMORY, SWAP AND MEMORY LEAKS). The Doxygen documentation system (Van Heesch, 2012) was used to generate the API documentation (see APPENDIX E - CCDVL API DOCUMENTATION), and it should be noted that although both Valgrind and Doxygen are distributed under the GPL license, their output and the generated documentation is not.

Typically, the authors would meet almost daily at the Chalmers campus, or periodically at the Combine office about once a week or every-other week, to work on the thesis. Otherwise, communication and collaboration was carried out through ordinary means such as e-mail, live chat and telephone conversations. Google Documents (Google Inc., 2012) was also used to record interviews and to share documents, material and ideas, as well as to write this thesis report and the presentation slides, and to get direct feedback from the supervisor through the built-in comments mechanism. The icons in the GUI and the figures in this report that lack stated sources, as well as the reproduced figures, were made by the authors using Adobe Photoshop (Adobe Systems Incorporated, 2012), GIMP (The GIMP Team, 2012), Microsoft Word (Microsoft Corporation, 2012b), Dia (Larsson, 2011) and yEd (yWorks, 2012). Finally, the report was transferred to a Word-document for fine tuning of the formatting and layout, before the final approved version was converted to a pdf-document and submitted for publication.

4 Development

This chapter goes into more detail about the iterative development process (described in the previous chapter) that was used to develop the resulting library. The sections below each describe one or more of the steps of the iterative development process, summarizing the tasks, design decisions, encountered problems and their solutions for the five iterations that were applied during the thesis work. The iterations varied in length and size to try to avoid splitting some of the larger tasks over several iterations. TABLE 2 below provides an overview of the iterations.

Table 2. *An overview of the iteration main tasks, along with number of subversion revisions and approximate time spent on each iteration.*

Iteration	Revisions	Length	Main tasks
1	41 (r001 – r041)	1 week	API framework, component skeleton structure and interfaces.
2	70 (r042 – r111)	5 weeks	Qt, GUI, renderer and memory manager.
3	129 (r112 – r240)	10 weeks	Multithreading, memory manager, graph tools, Doxygen and Python bindings.
4	73 (r241 – r313)	6 weeks	Refactoring, multithreading, graph clipmap cache, graph grid, graph axes value algorithm, graph neighbourhood overview and settings dialog.
5	79 (r314 – r392)	4 weeks	Refactoring, cleanup, optimization, formatting and documentation.

The authors collaborated and worked together on all parts, although the bulk of the work was split into two parts, roughly comprising the frontend and backend respectively. This allowed more parallel development, as well as allowing each author to focus on and work with the parts they were better suited for and were better aligned with the main focuses of their respective master programmes.

4.1 Planning and requirements

Initially, a rough time plan was made together with Combine, containing the bigger milestones and requirements (listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS), as well as dates for follow-up and evaluation meetings. Loose opportunistic time plans for the development and thesis in general were also made and updated at several occasions during the thesis work. Even so, they were a bit hard to keep up with due to unforeseen time sinks, some nasty bugs, too few firm milestones, and outside world obligations taking up much of the time.

Towards the end, the time series graph was dropped as a requirement due to time limitations, although the software library and API were still designed with this in mind to be able to support it in the future. Combine asked that the focus would be on finishing the basic scatter plot graph and API structure.

4.2 Analysis and design

Data and memory managers and iterators, described in sections 4.2.1.1 ITERATORS ANALYSIS AND DESIGN and 4.2.2.1 MEMORY MANAGER ANALYSIS AND DESIGN, are needed to be able to handle large amounts of data, as mentioned in section 1.2 PROBLEM, while avoiding the associated problems. This involves working with and displaying smaller portions of the data, in this case as a rendered image composed of tiles, to be able to process and interact with the data. A renderer, described in section 4.2.2.2 RENDERER ANALYSIS AND DESIGN, is thus needed to render the graph image tiles and a graph tile cache is used to speed up the rendering process. A frontend GUI, described in section 4.2.3 FRONTEND AND GUI ANALYSIS AND DESIGN, is of course also needed for displaying the graph image and to allow user interaction, as well as requiring a module to keep track of groups of data selected by the user through the GUI. All these components are also described in the following section.

4.2.1 API analysis and design

The memory manager is designed to avoid spending all system resources on system level memory management by utilizing manual memory page control to swap and free memory used for data as soon as possible, thereby avoiding system wide resource starvation as described in section 2.1 OPERATING SYSTEM MEMORY MANAGEMENT. Data access will be provided through iterators (see sections 2.4.3 ITERATOR), motivated by simplified data access and the possibility to track the number of existing iterators that point to some data, which in turn allows that allocated shared data to be freed immediately when it is no longer needed. Graphs will simply be images, which make interactive operations more difficult to implement. One of the largest problems associated with this is how to manage an interactive graph when the actual data will not be immediately available or selectable. Groups must therefore be represented as a selection of graph space, which solves data point membership by checking if a data point is inside the selection, instead of storing additional metadata for each data point.

The rendering process is expected to require a considerable amount of time to complete, and it is therefore important that it will be designed to not block continued processing of user actions and events, to fulfil the non-functional requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS. To achieve this, the rendering process will be separate from the frontend and GUI, allowing parallel rendering trough multiprocessing, as described in section 2.3 CONCURRENT PROGRAMING AND MULTIPROCESSING. Progressive rendering will also be supported by allowing graph images to be accessed even when the renderer has not finished rendering them. This requires being able to queuing graph images for rendering, requesting a graph image and an observer callback. The observer pattern, as described in section 2.4.1 OBSERVER, is used to signal the frontend that the rendering process is finished, simplifying progressive update stop

conditions. Early on it was decided that the GUI frontend, renderer and memory management were to be separate components, following good programming practice to make the API modular, portable and simple to use. An API draft based on this decision is shown in FIGURE 24.

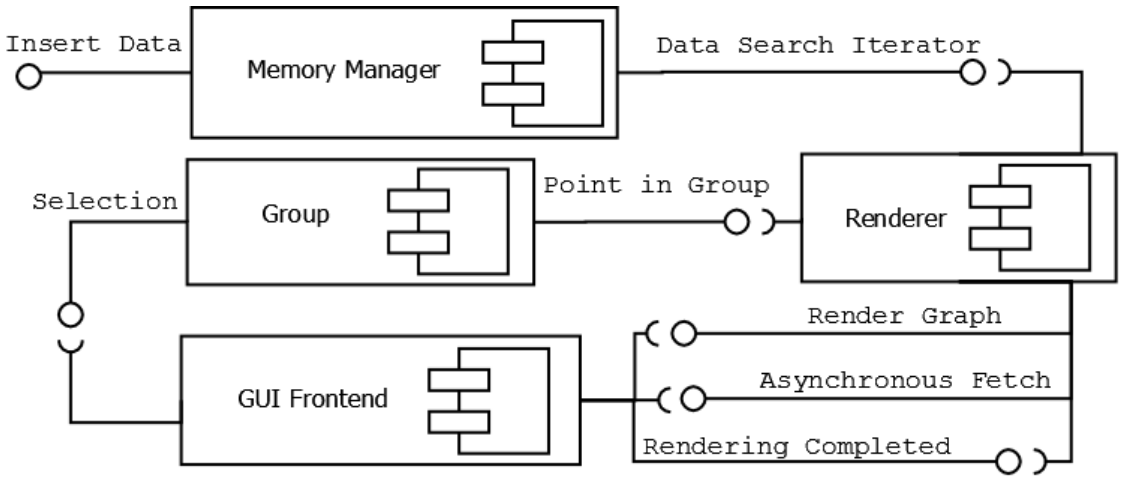


Figure 24. *Early API design overview.*

A typical usage scenario, shown in FIGURE 25, was also created based on this design.

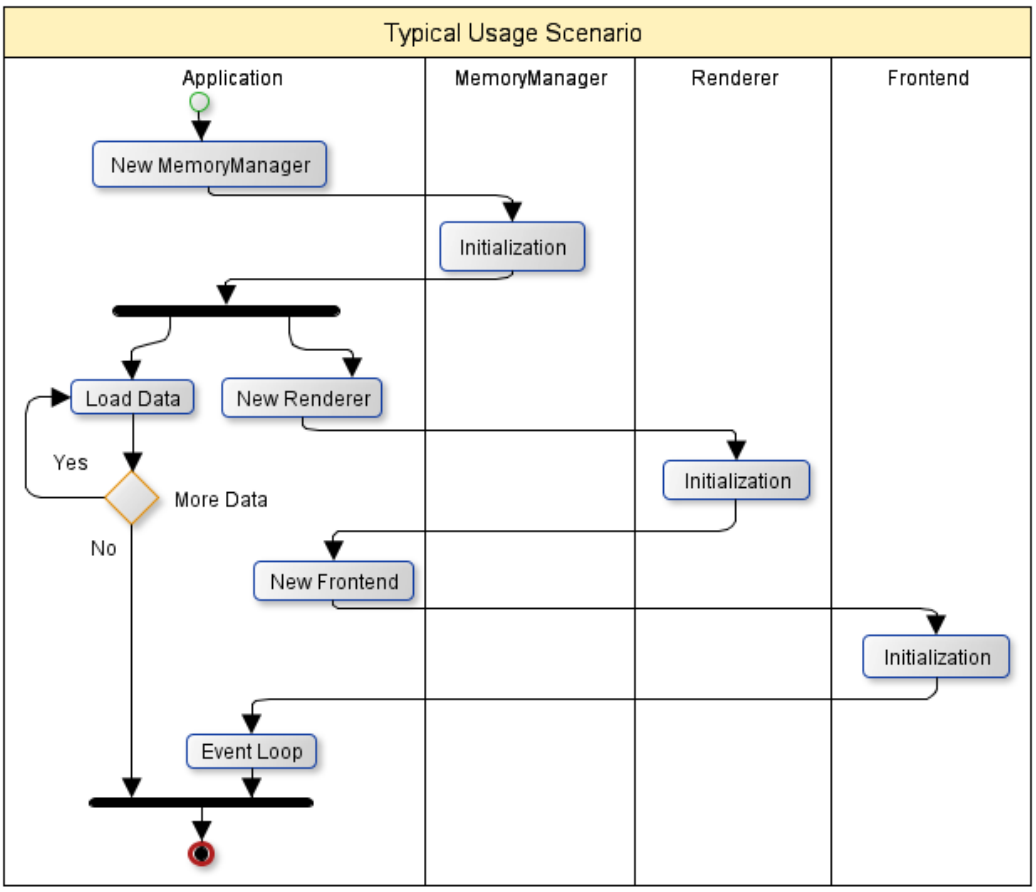


Figure 25 *A typical usage scenario as seen from the API perspective.*

A few general problems still remain to be addressed, such as the data type of data points, which is solved by letting the library work strictly with the double precision floating point data type internally. However, whenever data is returned through a user selection it would be preferable if the library returned the data in the form of the same numeric data type that was inserted into the memory manager to begin with. An abstract dataset class was thus introduced, having specific subtypes for each supported data type, which allowed conversion of data values to doubles through polymorphism, while using a visitor (see section 2.4.2 VISITOR AND DOUBLE DISPATCH) to determine the return data type of a selection.

Another problem concerns the rendering of graph tiles making up the graph image, much like a clipmap, and caching of said graph tiles. A cache module was thus added to the library and asynchronous rendering was incorporated into the cache, shifting process management from the three primary modules, particularly from the renderer, to the main library. The request for a graph tile and render graph tile was combined as it can be easily verified by a cache that a new graph tile must be created and returned. Furthermore, the renderer was tweaked to allow batch operations by accepting a list of tiles rather than a single tile at a time, primarily to improve rendering performance using the *SequentialMemoryManager* (see section 4.3.2.1 MEMORY MANAGER IMPLEMENTATION). Adding data to the memory manager while rendering should also be possible, leading to interesting synchronization requirements and communication interfaces between the different parts of the library, as seen in FIGURE 26. The memory manager was designed to be observable so that the cache could easily be invalidated whenever more data is inserted, to be able to handle the new data being inserted from another process more gracefully. In order to support graphs that can encode more than two dimensional data, the API supports components using *k*-dimensional data, where *k* is a nonzero integer

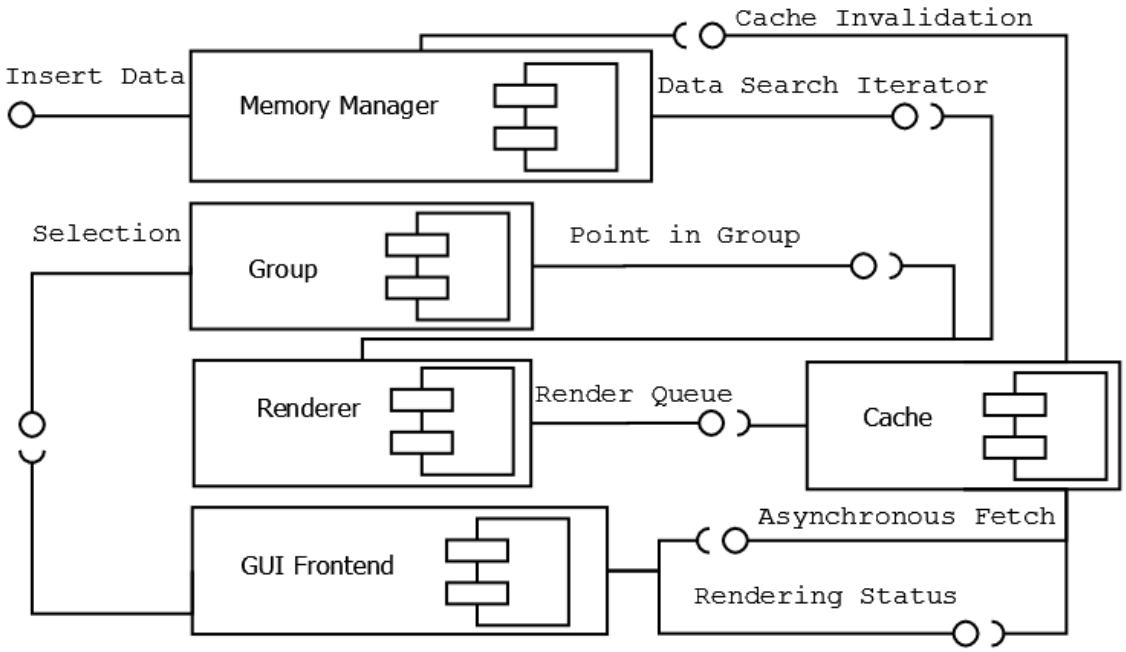


Figure 26. *Updated API design overview.*

To ensure generality, a group is thus not necessarily a selection of any kind but rather a k-dimensional bounding box and a boolean decision function that filter points from graph space covered by the bounding box, forming an abstract group. The basic idea is to consider a selection as a polygon and store it as a group, which is why the point in polygon algorithm in section 2.2.4 POINT IN POLYGON is also interesting. Finally, the frontend is required to provide methods to access user selections allowing applications relying on this library and API to query the memory manager and obtaining the user selected data points.

4.2.1.1 Iterators analysis and design

The memory manager iterator iterates over the abstract datasets mentioned above, and is stored by the memory manager, which is described further in section 4.3.2.1 MEMORY MANAGER IMPLEMENTATION. Each dataset is essentially an array representing data values for a single axis, followed by the dataset for the next axis in the iteration. Therefore, the actual data points must be obtained by pairing values from each dataset in a recurring series with a length equal to the number of dimensions used by the memory manager. This design is motivated by the need to preserve the original data type of the stored data and the order is similar to how existing libraries loads data, which is why the memory manager will work in the same way.

A group selection iterator is obtained from a user selection through the GUI frontend and iterates over the selected data points from a given memory manager. See sections 4.3.1.2 GROUPS IMPLEMENTATION and 4.3.1.3 ITERATOR IMPLEMENTIONS for further details.

4.2.2 Backend analysis and design

The backend consists of the memory manager and renderer modules, each with an important task, as described above in section 4.2.1 API ANALYSIS AND DESIGN. Further backend module details follow below.

4.2.2.1 Memory manager analysis and design

The memory manager is essentially similar to a database where the data is indexed over all its dimensions. The primary purpose is to avoid keeping data in memory whenever possible by, flushing data either to a temporary file or to the operating system swap, and partially loading data to memory as needed to and to quickly find data subsets. Additionally, safe synchronization is required, where a readers-writer lock (see section 2.4.4 READERS-WRITER LOCK) should be enough to insert data. Although, keeping active iterator instances will become more complicated if the underlying data structure accessed by the iterators is changed during iteration. Section 4.2.1.1 ITERATORS ANALYSIS AND DESIGN additionally describes that a memory manager accepts data as a series of axes each represented as an abstract dataset that manages the actual values, and that the iterator also accesses each dataset in the same way.

Creating a data structure capable of storing these abstract datasets efficiently is a complex task, perhaps worthy of its own thesis, where a multidimensional tree structure seems a good candidate. The basic idea is to exploit binary search (see section 2.2.2 BINARY SEARCH) where the search space represent the actual graph space and each element is a data point. Unfortunately this is not enough, since intersecting lines, connecting data points outside of a graph tile, will not have the intersecting part drawn on that graph tile. Those lines should obviously be rendered despite that the data points cannot be observed directly, which causes an additional requirement. A similar problem also applies to individual points close to the edges of a graph tile, which may intersect a neighbouring graph tile depending on the size of the rendered representation of the point. This could easily be solved by performing a slightly larger search request proportional to the point size. The data point order must also be preserved and the resulting iterator should follow this order, adding further requirements.

Ignoring the *k*-dimensional requirement, it turns out that there are some data structures capable of handling the problem with intersecting lines mentioned above, namely Line, MX, edge and polygon map trees (Samet, 2006). Line trees are not suitable due to approximation of arbitrary lines and because they do not store vertices. MX trees have a similar problem where the approximation decomposes lines to the last pixel. Edge trees are similar to MX-trees, but the approximation can be configured to be even less precise, leaving polygon map trees as the most promising data structure to use as base, as these handle edges by storing a reference to the endpoints of the lines that cross partitions (Samet, 2006). Another interesting data structure problem is that storage media tend to prefer reading data in sequential chunks or blocks. Tree data structures optimized for block storage are known as b-trees and are therefore commonly found in file systems (Samet, 2006). Considering all of the above, the required data structure is something similar to a polygon map based k-d-b-tree. But the truly challenging part is balancing that data structure efficiently, using only a limited amount of memory.

4.2.2.2 Renderer analysis and design

The large number of possible ways to present data to users calls for a large number of different renderers, or at the very least a vast configuration. The mindset for good data presentation is described in section 2.6 INFORMATION VISUALIZATION alongside with numerous examples of graph types and ways to encode information using a graphical representation. Obviously, attempting to support everything immediately is unrealistic and each renderer should represent a specific type of graph with a configuration as large as possible, possibly including groups as well. The way data is encoded and presented is thus entirely decided by the renderer, which may cause incompatible translations between the displayed view, the selection and the rendering of view if all components do not consider the data in the same way. For now, a standard Euclidean projection will be used, which will work well for two- and three-dimensional data, while the k-dimensional rendering model shall allow more exotic graphs and their respective view to be implementable using the API.

A basic two-dimensional plot graph could render each point as an actual (semi-transparent) point, draw lines between points and possibly aggregating close data points

to approximate their location before rendering them. The idea is to create a combined scatter and time series plot renderer with configurable points, lines and colors, as the two graph types are quite similar. Each graph tile that is used can thus be represented by a base offset, scale and zoom levels, size in pixels, saved selected groups and specific rendering settings.

4.2.3 Frontend and GUI analysis and design

As much of the provided functionality and as many as possible of the standard graphical components of Qt is to be used and built upon to both save time, to be able to focus more on fulfilling the requirements (listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS), and keep the local look and feel, which is automatically handled by Qt for each operating system. The built-in resource handling system is beneficial as it keeps that specific functionality of the library portable, and the provided localization system is also utilized to support the possible need for a translation in the future, as they are both simple and quick to use.

Communications with the backend will mainly be to fetch image buffers from the graph cache to compose the displayed graph image, and to handle saved groups of selected data points. Signals will be received and sent, both externally and internally, to stop or reload the renderer and update processes, to receive process progress updates from the renderer through the use of the observer pattern (see section 2.4.1 OBSERVER), and to notify the cache and renderer of updated parameters, such as zoom level or point color, when these change through GUI interaction. These communications and their respective triggered responses must also be synchronized with any on-going (threaded) operations, such as an update of the graph image or rendering process, as explained in section 2.4 SOFTWARE ENGINEERING DESIGN PATTERNS. Finally, the update process should be progressive to avoid causing an unresponsive GUI while waiting for the renderer and graph cache, thus allowing smoother continuous interaction (see section 2.6.4.2 INFORMATION SPACES AND TYPES OF INTERACTIONS) and supporting the design guidelines of visibility of system status and controlling response time, as described in section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES. GUI responsiveness is, as previously stated by the problem description in section 1.2 PROBLEM one of the major issues and requirements (see Appendix A – Initial requirements and requests) that need to be addressed.

4.2.3.1 GUI overview and structure

As the purpose of the proposed library is to visualize large amounts of data in graphs, it is reasonably safe to assume that applications using the library will have a sovereign posture to maximize the displayed graph area, and to allow potentially lengthy analyses and explorations of the visualized data. Sovereign posture also dictates a GUI optimized for full screen use, with a conservative look and not too many colors, as described in section 2.7.1 USER TYPE AND APPLICATION POSTURE. Use of the center stage and liquid layout design patterns, as defined by Tidwell (2011), and structuring the display while striving for a minimalist design, as described in section 2.7.3 DESIGN PATTERNS,

HEURISTICS AND GUIDELINES, also follow naturally. One interesting thing to note is that the typical user of the proposed software library and API is a software programmer, making it possible for the authors to think of themselves as typical users, which is rather unusual and something normally strongly advised against (Nielsen, 1993).

FIGURE 27 and FIGURE 28 show two of the more finalized versions of possible GUI designs and layouts of the graph widget example implementation prototype, displayed in an external window frame with a menu bar. The design and layout are somewhat influenced and inspired by Adobe Photoshop, perhaps most clearly noticeable in the graph interaction tools, tool icons, overview area and groups list, and their respective layouts in FIGURE 28. These follow standard and common look and use of icons and support the user habituation design pattern and user recognition, as recommended by Tidwell (2011) and the general guidelines found in section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES. This also supports the recommendation of designing for intermediates, as described in section 2.7.1 USER TYPE AND APPLICATION POSTURE.

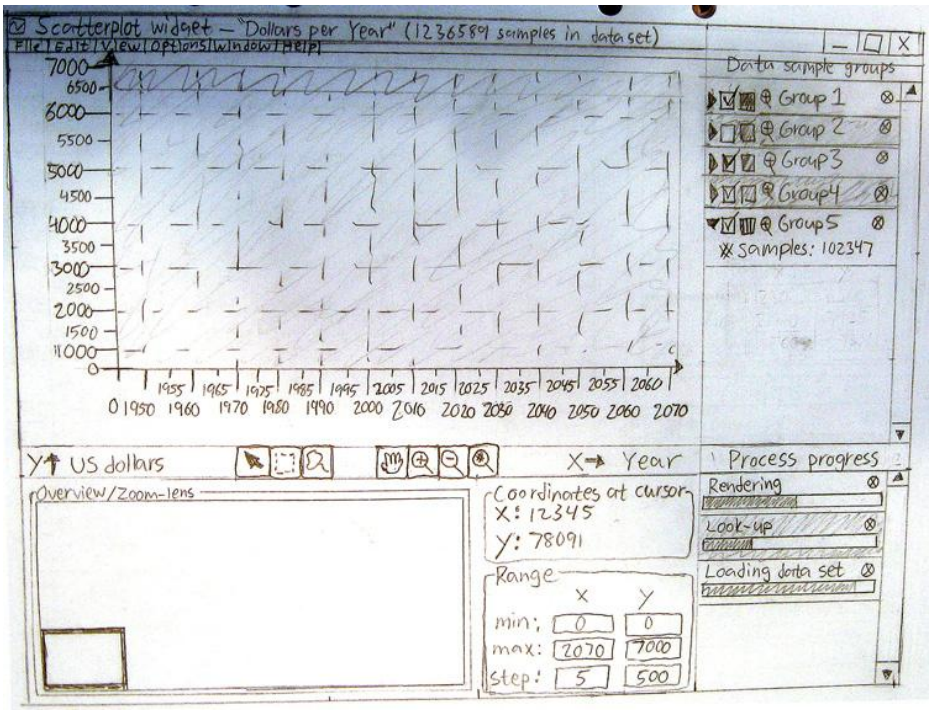


Figure 27. *A sketch of one of the proposed GUI design and layouts of the graph widget prototype, showing the main graph view area at the top left with a toolbar, overview, coordinate and graph view range boxes below. A groups panel and a process progress panel is shown to the right.*

The different main parts and areas of the GUI are designed as separate components, to allow quick and easy rearrangement of the main layout or replacement of specific components. The main components seen in FIGURE 28 are the graph view area with graph axes and toolbar (see section 4.2.3.2 GRID AND GRAPH AXES and section 4.2.3.3 GRAPH INTERACTION TOOLS), the neighbourhood overview frame (see section 4.2.3.4 NEIGHBOURHOOD OVERVIEW), the graph view range box showing the current graph view position (see section 4.2.3.5 INFORMATION LABELS), and the groups list of selected and saved groups of data points (see section 4.2.3.6 GROUPS LIST). Many of these directly or indirectly correspond to one or more of the requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS.

The process progress list, visible in the bottom right corner of the sketches in FIGURE 27 and FIGURE 28 was deemed mostly superfluous and was never implemented to avoid adding visual excise (see section 2.7.2 FLOW AND EXCISE). It was replaced by a simple status bar showing messages, mouse-over help texts through brushing (through brushing described in section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION) and a progress bar during updates, as well as containing a settings button used to open the settings dialog. The progress indicator is another Tidwell (2011) design pattern that, in this case, also encourage users to expect and look for changes during the progressive updates, which may otherwise go unnoticed due to change blindness (see section 2.6.4.3 THE ACTION CYCLE AND HUMAN ASPECTS OF INTERACTION).

The graph view area is pannable and zoomable since the amount of screen space is limited and it is not always possible or desirable to show the entire graph in one view, as described in section 2.6.3.1 DISPLAY LIMITATION TECHNIQUES. Pan and zoom also help with navigation and answer some of the questions found in section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION, as well as being two of the more important requirements (see APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS).

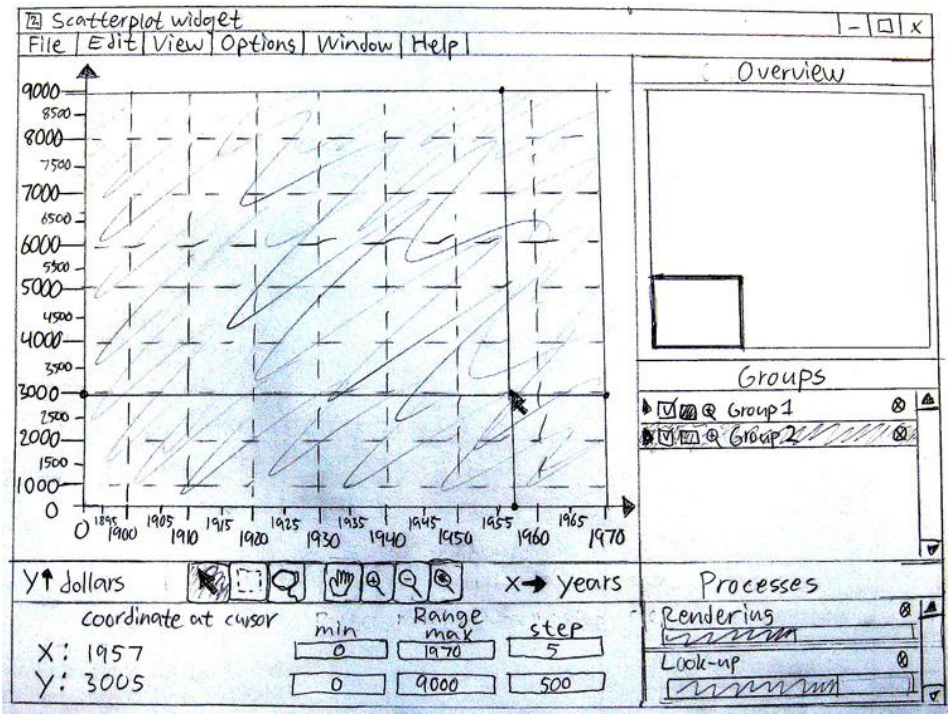


Figure 28. *Another sketch of one of the proposed GUI design and layouts of the graph widget prototype, closer to the final design with the overview box in the top right corner and the coordinate and graph view range boxes combined at the bottom*

4.2.3.2 Grid and graph axes

A basic toggleable grid, originating from the graph origin and following the step values of the graph axes, was added to make the graph easier to interpret at a glance and to support navigation (see section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION). The grid is visible in the empty graph view areas in both FIGURE 27 and

FIGURE 28, although the final grid line style was made solid and semi-transparent rather than dashed, and the grid lines intersecting the origin were made thicker to emphasize the borders of the positive and negative halves of the graph.

The alternating pattern of the specified graph axes step values and the intermediate values displayed with smaller font size, as seen along the axes in FIGURE 27 and FIGURE 28, change dynamically with the graph view when updating, panning, zooming or changing the graph axes scale factors. If the absolute value of any graph value displayed along the axes are too high or too low, it will instead be displayed with scientific notation broken into two lines to better make use of the available space. The values are also adjusted to align themselves symmetrically around the graph origin with the use of monospace fonts and padding. These design choices are based on both experience and the guidelines presented in section 2.6.2.2 DATA ENCODING METHODS AND GUIDELINES, such as presenting many numbers in a small space and showing as much data as possible, revealing data at several levels of detail and using words in full with standard orientation.

4.2.3.3 Graph interaction tools

The graph interaction tools in the toolbar are grouped into two button groups, following the button groups design pattern defined by Tidwell (2011), which is in turn based on the basic grouping principles in section 2.7.4 GESTALT GROUPING PRINCIPLES; one button for selection-tools and one button for pan- and zoom-tools. The tool buttons have simple icons and mouse-over tooltips to further clarify the respective function of each tool, which is another example of the brushing technique, described in section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION. Each tool also has a custom mouse cursor to further indicate its function, such as a cross for rectangle select and a hand for panning, corresponding to the standard cursors normally used for such tools, thus following the guidelines in section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES and the habituation design pattern (Tidwell, 2011). Some of the standard cursor icons were provided by Qt, but all of the other tool icons were made from scratch with Photoshop, since a free to use set of icons, with a satisfyingly homogeneous look and feel, that included all the needed icons could not be found. FIGURE 29 shows some suggested tools and icons, of which the majority was kept in the final version. Visual helper lines, radiating from the current mouse cursor position towards the graph axes (visible in FIGURE 28 were also added to the point select tool to make the graph easier to read and to further support navigation (see section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION).

The zoom lens tool was intended to act as a real world magnifying glass, with a dynamically changeable magnification level, to add an extra level of overview plus detail (see section 2.6.3.1 DISPLAY LIMITATION TECHNIQUES) and utilize the local zooming design pattern (Tidwell, 2011), but was excluded due to time limitations as it was not among the requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS. The undo and redo functions, even though strongly recommended in section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES, was deemed to have too little practical use in this specific context to be implemented, as the idea was more along the

lines of being able to go back and forth between previously rendered graph images, which would not have added much to the navigation, although helping to answer the navigational question "where have I been?" (see section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION).

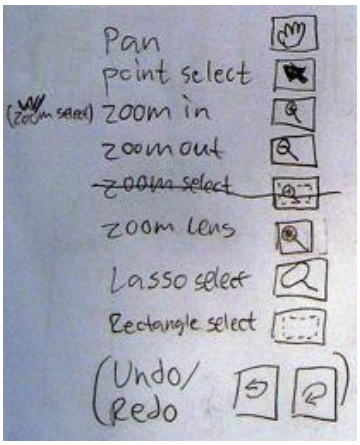


Figure 29. *A list of possible tools along with suggestions of corresponding icons. Tools from top to bottom: pan, point select, zoom in, zoom out, zoom select, lasso select, rectangle select and undo/redo. The reason why the zoom select tool has been striked out can be found in section* 4.4.2.2 GRAPH INTERACTION TOOLS EVALUATION AND ALTERATIONS.

A separate stop/reload button, not visible in any of the design sketches, was also added to allow stopping ongoing progressive updates and reloading the graph view if necessary, which was also one of the requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS. It was placed in the gap at the bottom left corner of the graph view, below the y-axis and in front of the x-axis to keep it close to the graph view as recommended by the cancelability design pattern (Tidwell, 2011).

4.2.3.4 Neighbourhood overview

The graph neighbourhood overview was initially intended to be a simple overview of the entire graph to provide an overview plus detail (see section 2.6.3.1 DISPLAY LIMITATION TECHNIQUES), which was unfortunately more or less pointless for anything but displaying all clusters large enough to be visible (in relation to the range of the data) in such a small overview. Thus, this feature was dropped in favour of a bigger graph view area, even though it was a requirement (see APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS), area, only to be brought back at a later stage in the form of a neighbourhood overview, showing a scaled down version of the immediate surrounding graph area. The overview consists of a black and white mask showing the points with clear contrast, since transparent points are hard to discern against the background when scaled down (see section 2.6.2.2 DATA ENCODING METHODS AND GUIDELINES).), or not drawn at all depending on the renderer and render settings used. A red rectangle outline directly linked and proportional to the current graph view size and shown position in the graph, moves with the graph view when it's shown position changes and may also be used to pan the graph view area. This helps navigating the graph and helps answer some of the navigational questions found in section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION.

4.2.3.5 Information labels

Displaying numbers indicating the current position of the graph view naturally helps the user navigate the graph (see section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION) and currently used scale and zoom factors were added as well to provide a better context for the shown graph view. The step values visible in FIGURE 27 and FIGURE 28 was moved to the settings dialog and grouped with the other graph axes settings. The scale and zoom factors are presented as percentages rather than floating point numbers to have a better match between the system and the real world, as advised by section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES and the habituation design pattern (Tidwell, 2011).

One initial idea was to have the graph range values as text inputs, as shown in FIGURE 27 and FIGURE 28 to allow the user to quickly enter specific ranges and step values for the graph axes values. However, this idea was abandoned and the ranges was replaced with text labels, since part of the purpose was to be able to explore the dataset through the visualized graph with the help of the tools, not by manually entering numbers. It also cluttered the GUI a bit, adding unwanted visual excise (see section 2.7.2 FLOW AND EXCISE).

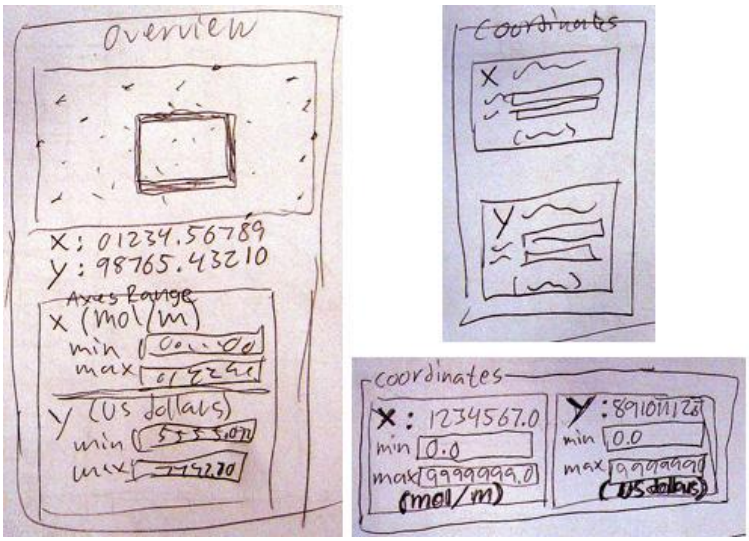


Figure 30. *Some alternative versions and layouts of the graph neighbourhood overview, graph view range and coordinates.*

The positions of the x and y unit labels, marked as “US dollars” and “years” in FIGURE 27 and FIGURE 28, changed between iterations from the toolbar below the graph view area to above the graph view area, to inside the graph view area, within and along the axes, and to different combinations of these positions, simply to see what positions were best suited. Their final positions can be seen labelled as “x test label” and “y test label” respectively in FIGURE 37 found in chapter 5 RESULT. The x- and y-coordinates showing the mouse-over values, as well as the graph view range text inputs, also changed position and orientation to a lesser extent for the same reason, but not as radically as some of the first suggestions shown in FIGURE 30 above.

4.2.3.6 Groups list

The row striped groups list to the right in FIGURE 27 and FIGURE 28 contains saved groups of points (selected by the selection tools). Row striping, i.e. alternating between lighter and darker background shades to make rows easier to separate and read, is a design pattern defined by Tidwell (2011). To fulfil the group requirements (listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS), each row in the group list contains a small open/close button with an arrow icon, a checkbox to toggle visibility in the graph, a color select button showing the current color for the points contained by the group, a "zoom to and center on group" button with a magnifying glass icon, the name of the group and a delete button with a cross icon that removes the group from the list. The buttons all have tooltips as well, and expanding the group displays additional information, such as the number of points contained by the group. This technique, which allows the user to easily show and hide additional information in a list of items to lessen visual excise (see section 2.7.2 FLOW AND EXCISE), is called list inlay and is also one of the design patterns defined by Tidwell (2011). FIGURE 31 below shows two early versions and variations of the groups list with varying layouts and information, such as combining the color with the visibility checkbox and providing values for the number of included points, as well as minimum, maximum and average values of these points for each axis. Hierarchical groups should be indented according to their respective position within the hierarchy, as well as shown and hidden when parent groups are opened and closed.

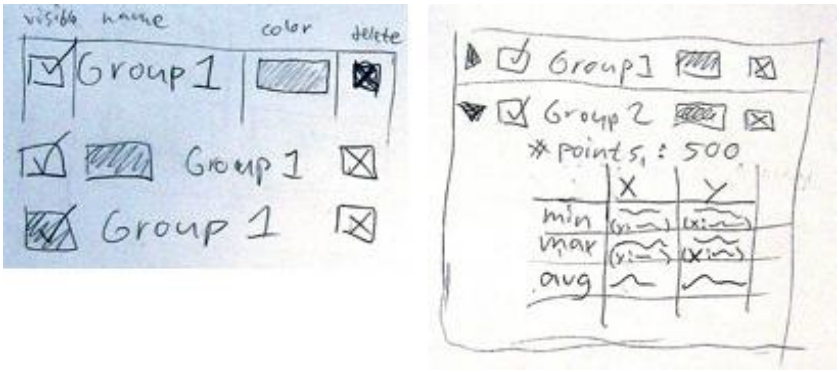


Figure 31. *Early versions of the row layout and contents of the groups list.*

4.2.3.7 Settings dialog

The modal settings dialog was initially designed to contain all settings in different group boxes (see section 2.7.4 GESTALT GROUPING PRINCIPLES) on one single page as shown in the sketch found in FIGURE 32, but this design proved to be too tight and too large to fit on a typical display. Using scrolling, as suggested in section 2.6.3.1 DISPLAY LIMITATION TECHNIQUES, seemed a bit excessive as all the settings and information did not have to be shown at the same time. The settings dialog was instead divided into several tabs in a settings editor – yet another design pattern from Tidwell (2011). Furthermore, it also incorporates the right/left alignment, fill-in-the-blanks, input hints and dropdown chooser design patterns, also defined by Tidwell (2011). This made the settings dialog more useful and reduced visual excise (described in section 2.7.2 FLOW

AND EXCISE). The layout and use of the various settings, of which some directly relate to the requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS, are shown and explained further in APPENDIX B - MANUAL AND USER GUIDE.

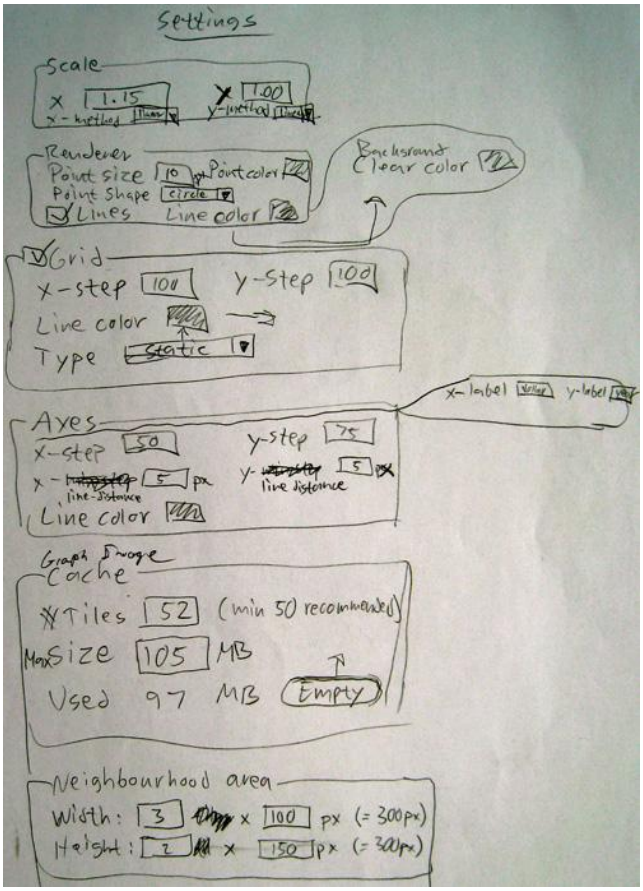


Figure 32. *A quick sketch of the contents of the settings dialog.*

4.3 Implementation

The library itself was divided into four major parts, as summarized below, which were put into the *ccdvl* namespace, named after the library. All the implementations are of course based on the result of the analysis and design described in the previous section.

- **API;** Includes class interfaces for the three different modules, groups, iterators and generic configuration options. Implements asynchronous rendering, graph tile caching and Python help-classes, as well as basic implementations for groups, iterators and mathematical transformations for scaling, zooming and computing view coordinates.
- **Memory management backend;** A backend that stores data, quickly searches for data and keeps the data accessible as long as needed. The memory manager implementations were put into yet another namespace part of *ccdvl*, simply named *memorymanager*.

- **Renderer;** A backend that renders graph image tiles. Like memory managers, renderers were also given a common namespace under *ccdvl*, called *renderer*.
- **GUI Frontend;** A frontend that include the GUI that displays, configure and navigates a graph. The frontend was also collected under *ccdvl* in a namespace called *frontend*.

4.3.1 API implementation

The primary parts of the API are just plain interfaces, some implementations and a few abstract classes. FIGURE 33 below shows the classes belonging to the library, excluding the frontend and GUI, which are shown in FIGURE 34 in section 4.3.3 FRONTEND AND GUI IMPLEMENTATION.

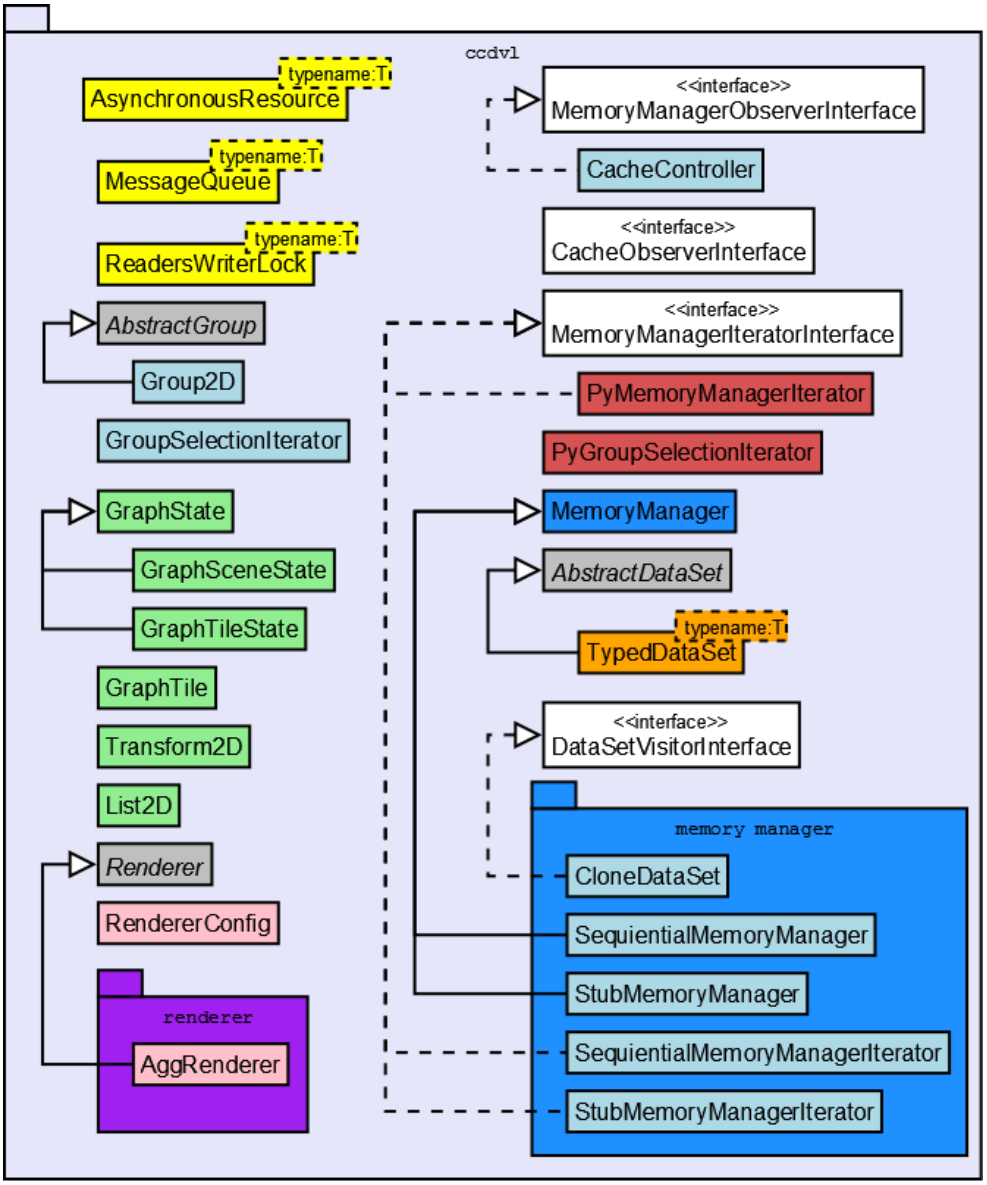


Figure 33. *A class diagram showing the library classes not belonging to the frontend or GUI.*

4.3.1.1 Cache implementation

The cache was first implemented as a graph clipmap that used a distance function to find neighbouring graph image tiles, which in theory is a large help class for the GUI, but it was later refined into a pure cache that no longer performed neighbourhood searches. The only addition to the refined cache was to allow it to handle asynchronous rendering to avoid the problems described in section 2.3 CONCURRENT PROGRAMING AND MULTIPROCESSING, by keeping a graph tile rendering queue for synchronization (see section 2.4.5 MESSAGE QUEUE). To simplify caching, the least recently used algorithm, described in section 2.2.3 CACHE was used without randomization to reduce both re-reading data from storage media and redrawing of graph tiles. The list of existing tiles and their images were stored internally in a standard map data structure where the key used for lookup was the tile itself and the stored data was simply the allocated image buffer. When a requested tile is unavailable the cache allocates a new image buffer and then proceeds to clear it to its background color using the renderer. It is then enqueued for asynchronous data rendering before finally returning the image.

4.3.1.2 Groups implementation

The *Group2D* class was implemented with the (optional) hierarchical groups requirement in mind (see APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS), based on the idea that a group is a hierarchical data structure of selections, although full hierarchical support was never tested or finished completely. The "point in group" membership test was implemented using the odd or even algorithm (see section 2.2.4 POINT IN POLYGON), as no better algorithm was found at the time.

4.3.1.3 Iterator implementions

Two kinds of special iterators and one basic iterator were implemented and specified; a wrapper memory manager iterator, a group iterator and the basic memory manager iterator. A rather nice feature is that a memory manager must provide their own iterator implementations, which are wrapped in a safe way to allow them to be used more like regular C++ iterators. In this, the iterator implementations diverted from the chosen code standard in the Google C++ Style Guide with the motivation to preserve the same behaviour as the standard C++ iterators, and the group and memory manager iterators were each given an additional wrapper iterator to provide good compatibility with Python as well.

The *SequentialMemoryManagerIterator* was implemented to also count the number of instances referencing each stored abstract dataset, allowing memory to be deallocated immediately when the counter reaches zero. The *StubMemoryManagerIterator* on the other hand, is simply a reference to the generated datasets. Further memory manager implementation details are covered in section 4.3.2.1 MEMORY MANAGER IMPLEMENTATION.

4.3.1.4 Mathematical transformations

A help class was set aside for mathematical transformations, partly to avoid code duplication between the implemented renderer and the implemented GUI module, as well as to keep related transform operations together. It was only implemented to support two dimensional data with scaling and zooming, with a flipped y-axis because of the GUI, as described in section 4.3.3.3 QT AND GRAPH COORDINATE SYSTEMS. Updating the view center coordinates was an interesting problem that was incorporated into this class as well. See section 4.3.3.4 GRAPH IMAGE CLIPMAP POSITION for a full description of this particular problem and its solution. The resulting floating point values from unsuccessful transformations, namely infinity or NaN, should be caught and passed on as an error to the calling method to ensure proper error handling. The scaling transformation formulas follow below.

Let P be a coordinate to convert.

Let $z_i \in \mathbb{R}^+$ be the ith dimension zoom ratio.

Let $s_i \in \mathbb{R}^+$ be the ith dimension scale magnitude.

Let $f : \mathbb{R}, \mathbb{R}^+ \rightarrow \mathbb{R}$ be a scaling function.

Then the converted value for P_i is equal to $f(P_i, s_i) / z_i$.

The need for linear and logarithmic scaling functions were derived from the requirements in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS. Programing language wise, the *cmath* library in C++ is only shipped with two different logarithmic functions; the natural logarithm ($\log$) and the tenth logarithm ($\log10$). Similarly, the same is also true for the corresponding exponential functions. The s_ith logarithm of x can simply be computed with $\log(x) / \log(s_i)$. The following mathematical solutions are obtained by combining the above with the required scaling methods.

Scaling method solutions
linear : $x * s_i / z_i \Leftrightarrow x / (z_i / s_i)$
logarithmic : $\log(x) / \log(s_i) / z_i \Leftrightarrow \log(x) / (z_i * \log(s_i))$

Designing an algorithm (see section 2.2.1 ALGORITHMS) using these solutions is trivial and it is also possible to use precomputed constants for the zoom and scale ratio, which reduces the number of floating point operations needed when performing multiple coordinate conversions. Pseudocode for the scaling transformation algorithms in the mathematical transformation class follow below.

Definitions
Let P be a coordinate to convert. Let $z_i \in \mathbb{R}+$ be the *i*th dimension zoom ratio. Let $s_i \in \mathbb{R}+$ be the *i*th dimension scale magnitude. Let *k* be the number of dimensions of P. Let A be an array of length *k*.

Initialization	**Transformation**	**Inverse transformation**
For each dimension *k* If logarithmic scaling $A_k = z_k * \log(s_k)$ Else $A_k = z_k / s_k$	For each dimension *k* If logarithmic scaling $P_k = \log(P_k) / A_k$ Else $P_k = P_k / A_k$	For each dimension *k* If logarithmic scaling $P_k = \exp(P_k * A_k)$ Else $P_k = P_k * A_k$

4.3.2 Backend implementation

This section contains a general overview of the implementation and workings of the memory managers in section 4.3.2.1 MEMORY MANAGER IMPLEMENTATION and the AGG renderer in section 4.3.2.2 RENDERER IMPLEMENTATION.

4.3.2.1 Memory manager implementation

Only two test memory managers were implemented due to time constraints and to keep things simple; *StubMemoryManager* and *SequentialMemoryManager*. The first was a simple generator that created a small dataset, the second had more advanced functionality capable of storing, swapping data, freeing memory and re-allocating data. It was unfortunately not fully synchronized nor indexed for fast search and therefore not optimal. The data is tracked using a simple list rather than a more complex data structure better suited for search, such as the one mentioned in section 4.2.2.1 MEMORY MANAGER ANALYSIS AND DESIGN. Implementations of the abstract dataset class, introduced in section 4.2.1 API ANALYSIS AND DESIGN, was used to represent stored data and to preserve the data type while avoiding use of runtime type information by using the visitor design pattern (see 2.4.2 VISITOR AND DOUBLE DISPATCH).

The new operator in C++ is normally used to manually allocate memory, but the *SequentialMemoryManager* asks the operating system for a memory mapped file to store the dataset in instead. The operating system proceeds to read pages of this file into frames, which are then cached, effectively creating a separate controllable swapping mechanism. The advantage of this approach is that the memory manager is directly able inform the operating system when it no longer needs the previously allocated pages, allowing it to write the pages back to the file and release the used memory more efficiently than the C++ delete operator. A data subset of reasonable size can thus be mapped to frames in memory from the swap file to sequentially traverse the entire dataset in smaller chunks. See section 2.1 OPERATING SYSTEM MEMORY MANAGEMENT for more information on operating system memory management.

A side effect from the current memory manager interface specification is that it must make a copy of the provided dataset, which is inefficient but only affects the initial loading time and the required amount of hard drive space. Ideally, an implementation of the memory manager could avoid additional copying by providing emplacement operations, which are methods to allocate objects in place. Emplacement operations are available in the new C++11 standard (International Organization for Standardization/International Electrotechnical Commission, 2011), but this was not used due to Google C++ Style Guide restrictions.

4.3.2.2 Renderer implementation

A software-renderer based on AGG was implemented, intended as a reference implementation to ease the implementation of a renderer using hardware acceleration later on. Asynchronous rendering is used to allow user actions to be processed during rendering, making the time needed to render a graph image less critical. Finally, AGG has a scan-line rasterizer limitation which can be triggered by attempting to render very long lines, possibly far outside of the canvas (Gmane, 2004). A workaround should be possible by limiting the required raster size for a line, simply by checking if the limit is about to be reached and immediately performing a partial rendering of that line if this is the case.

4.3.3 Frontend and GUI implementation

All Qt class names use the letter *Q* as prefix, and the bulk of the GUI was mainly implemented using the aptly named *QCheckBox*, *QColor*, QDialog, *QFont*, *QFrame*, *QIcon*, *QImage*, *QLabel*, *QGraphicsScene*, *QGraphicsView*, *QPainter*, *QPixmap*, *QPushButton* and *QString* classes, along with the provided layout manager classes, such as *QGridLayout*, *QGroupBox*, *QHBoxLayout* and *QVBoxLayout*. FIGURE 34 contains an overview of the CCDVL classes, using the above Qt classes to make up the frontend.

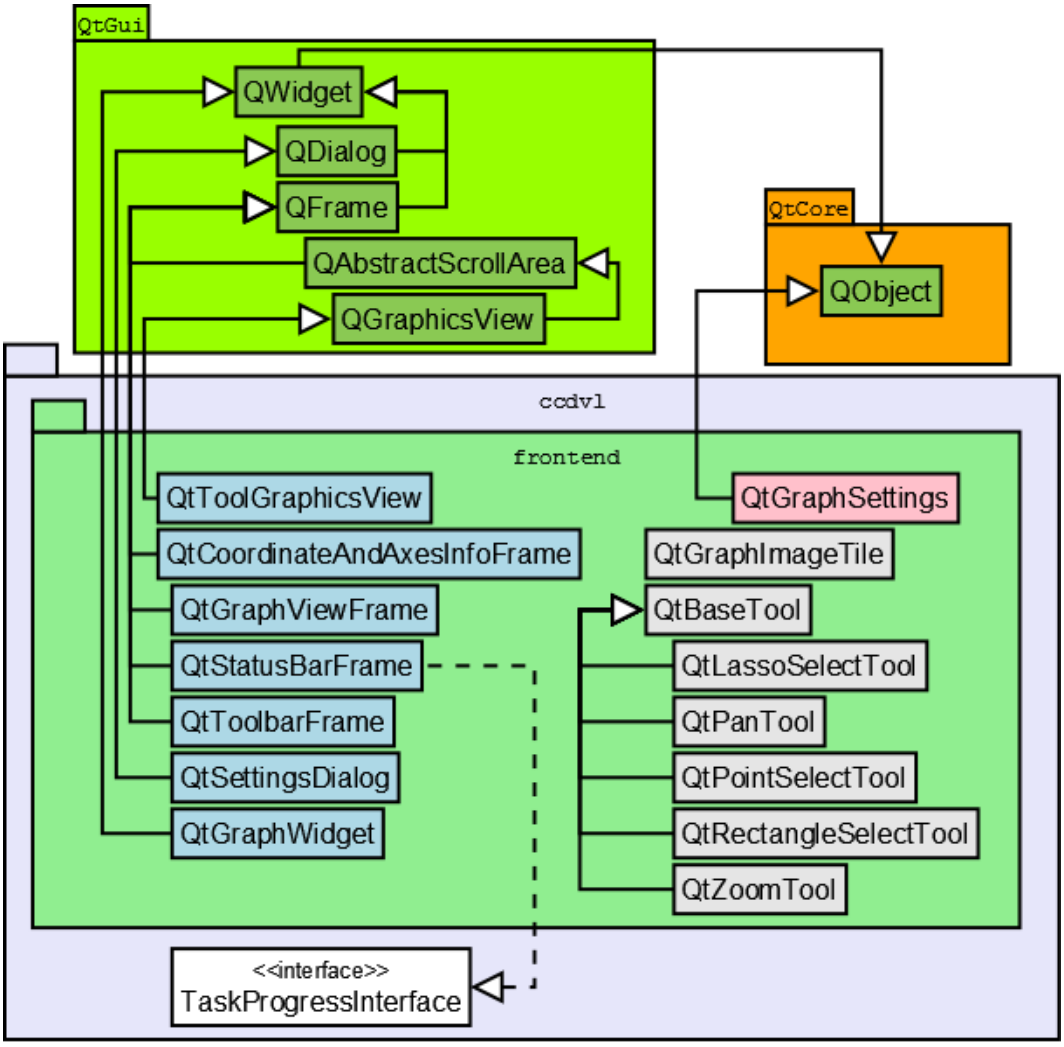


Figure 34. *A class diagram of the frontend. The QtGui and QtCore modules and their contained classes belong to the Qt framework and are not directly part of the CCDVL library. All CCDVL class names use* Qt *as prefix to signal that it is a Qt-based frontend.*

The main graph view area, *QtGraphViewFrame*, is implemented as a frame containing the toolbar of graph interaction tools (see section 4.3.3.1 GRAPH INTERACTION TOOLS AND SETTINGS below), an update stop/reload button, graph axes, graph axes labels and a graphics scene coupled with a custom graphics view (*QtToolGraphicsView*) to handle the graph tool interactions and events, as well as providing additional graph coordinate conversion functions (see section 4.3.3.3 QT AND GRAPH COORDINATE SYSTEMS). The graphics scene can be seen an off-screen canvas, managing a collection of graphics items, shapes and their respective layouts, orientations and sizes. It is viewed partially (or wholly) through the viewports of one or more graphics views connected to the graphics scene. Three overlapping layers of graphics items were added to the graphics scene; the image of the plotted graph at the bottom layer, the grid drawn on a transparent glass pane at the middle layer and graph interaction tool graphic items, such as the selected points polygon, at the top layer.

The x-axis and y-axis are also implemented as separate graphics scenes, with matching width and height of the plotted graph image respectively, along with a graphics view each. The scrollbars of all three graphics views are hidden and synchronized to make

sure that the axes values mirror the position of the graph view. The algorithm that is used to display the graph axes values proved to be a very fiddly time sink, taking *far* longer to implement than intended. This is also one of the reasons why the groups list was not implemented, simply due to time constraints.

4.3.3.1 Graph interaction tools and settings

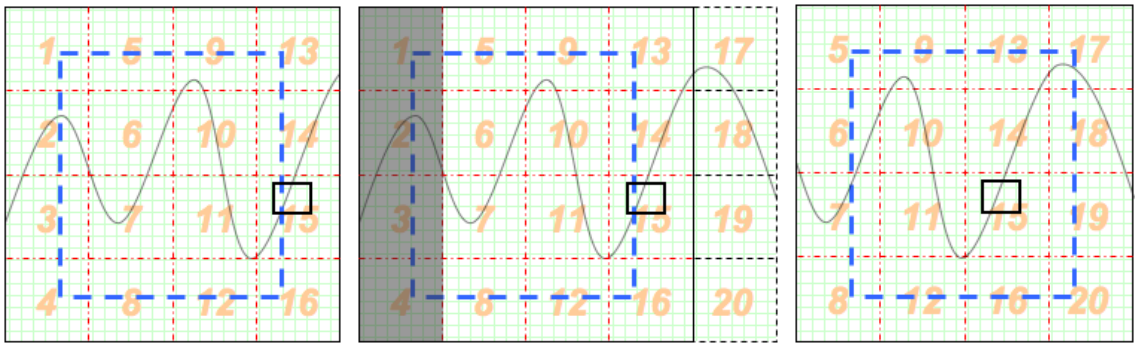
The graph interaction tools (and *QtGraphImageTile* described in section 4.3.3.2 GRAPH IMAGE CLIPMAP AND STATE OBJECTS) were the only classes of the frontend that did not extend any Qt class (see FIGURE 34), although they inevitably make use of many Qt classes, such as cursors, shapes, graphics items and events. The shared functionality and functions of the tools, such as default mouse cursor icon and functions for handling key, mouse and wheel events, are inherited from the *QtBaseTool* class, making it easy to add more shared functionality with a minimal impact on the rest of the implementation, as opposed to using a single complex event filter attached to the graphics view. As mentioned above, the tools do not listen for events, and the corresponding event handler functions must be called by the actual event listener, i.e. *QtToolGraphicsView* in this case.

Some interesting details in the implementation of the graph interaction tools include the lasso select tool using and updating an open path (a line) drawn by the cursor movement, with a closed copy of the path (a polygon) used to display the selection. Another important detail is that the point selection tool selects a small square rather than a single pixel, as a single pixel selection was too hard for the user to spot, as well as being hard for both the frontend and backend to determine exactly which data point was actually selected on low zoom levels or in tight clusters of possibly overlapping data points. This is especially true since the point selection takes place on an image representation of the graph data with possible rounding errors when converting and looking up coordinates (see section 4.3.3.3 QT AND GRAPH COORDINATE SYSTEMS). Finally, the rectangle select tool uses a rubber band selection functionality provided by Qt and the zoom selection actually makes use of a rectangle select tool internally.

A settings class, called *QtGraphSettings*, contains copies of both current frontend and current backend settings, such as axes, grid, scale, zoom, renderer and cache settings, as well as the graph scene state objects (see section 4.3.3.2 GRAPH IMAGE CLIPMAP AND STATE OBJECTS below). It is used by the frontend to show and modify settings through internal functions as well as through the settings dialog in the GUI. It using Qt signals and slots, made available by extending the *QObject* class, to notify the *QtGraphViewFrame* class when changed settings require the displayed graph view area to be updated. The settings dialog (*QtSettingsDialog*) suffered due to time running short, and was thrown together rather quickly at the end of the development, which is unfortunately reflected in its poor design and partial implementation. It extends the QDialog class and mostly makes use of the *QPushButton*, *QLineEdit*, *QComboBox*, *QSpinBox*, *QDoubleSpinBox* and *QColorDialog classes*.

4.3.3.2 Graph image clipmap and state objects

The graph image is composed by the frontend by stitching together fixed-size graph image tiles, just like a clipmap. The graph image tiles are fetched on request from the backend, where they are either rendered by the renderer or fetched from the graph cache if they have already been rendered and are still present in the graph cache. FIGURE 35 shows an example of how the graph image is updated when panning too close to an edge of the graphics scene, as shown in step (a). Step (b) shows how tiles 1 through 4 are discarded, tiles 5 through 16 reused, and tiles 17 through 20 are fetched from the backend. Step (c) shows the new graph image put together during the update, and how the viewport has been recentered on the new position in the graph image, which corresponds to the same graph coordinates as those in the previous graph image. The observant reader will notice that although the graph coordinates of the viewport are still the same in step (c) as in step (a), the coordinates within the graphics scene are not. Section 4.3.3.3 QT AND GRAPH COORDINATE SYSTEMS explains this in detail. Another thing to note is that the invisible border, shown as a dashed blue rectangle in FIGURE 35, is proportional to the size and shape of the viewport, more specifically by a factor of 1.5, to allow some further panning space while the graph image updates. The viewport, as well as the linked outline rectangle in the graph neighbourhood overview (described in section 4.2.3.4 NEIGHBOURHOOD OVERVIEW), also resize and change shape along with the graph widgets liquid layout (as mentioned previously in section 4.2.3.1 GUI OVERVIEW AND STRUCTURE).



(a) *update triggered* (b) *update in progress* (c) *update done*

Figure 35. *An example showing the three steps of a pan triggered update of the graph image. The graph image tiles have been outlined and numbered in this example for clarity. The black solid outline represents the position, size and shape of the viewport in the scene, and the blue dashed outline represents the normally invisible border which the center point of the viewport must pass to trigger an update of the graph image, as seen in step* (a). *Step* (b) *removes tiles 1-4 and appends tiles 17-20, resulting in the new graph image in step* (c).

The clipmap data structure is made up of a two-dimensional standard C list (the *List2D* class seen in FIGURE 33) containing *QtGraphImageTile* objects. Using lists rather than vectors avoids the problem of being forced to shift the entire contents every time a row or column is prepended, appended or removed. The advantage of vectors having constant access time (see section 2.2.1.1 ALGORITHM COMPLEXITY ANALYSIS) for single elements, while lists require linear time, does not make that much of a difference considering that the entire clipmap data structure usually has to be traversed for most operations, such as updates, which is done in linear time no matter which of these data structures are used.

The graph image tile objects contain a valid or invalid tile image as well as a flag used to indicate if the graph image tile has been drawn to the plotted graph image during an iteration of a progressive update, which are used to indicate if a tile needs to be fetched or redrawn during update iterations. By also looking at the renderer image tile flags, which indicate when a fetched tile is completely rendered, each completely rendered tile is guaranteed to be drawn only once per update rather than, as initially implemented, once per update iteration. An update is finished when all image tiles have been successfully rendered, fetched and drawn to the plotted graph image, which in turn also stops the timer controlling the progressive updates. The progressive update functions are also able to detect new update requests, such as repetitive zooming, before the current update has finished, allowing the current update to be aborted and replaced with the new update operation. Although this saves some time and resources in the frontend and stops further requests being sent to the renderer regarding the last update, it still allows the renderer to finish the last set of requests, since it may currently, or in the future, handle other requests simultaneously.

The updates themselves are facilitated using state objects, which keep track of the current and next state of the graph, graphics scene and the graph image tiles. The next state object replaces the current state object after the necessary updates have been made, which involve getting updated values from the next state object. When the graph widget is idle, the current and next state objects are thus identical copies of each other. See sections 4.2.1 API ANALYSIS AND DESIGN and 4.2.2.2 RENDERER ANALYSIS AND DESIGN for more information about the renderer.

4.3.3.3 Qt and graph coordinate systems

One of the major sources of bugs and headaches was caused by the many different coordinate systems used by Qt and the additional one used by the graph to address pixels and points respectively. Qt only uses the *QPoint* and *QPointF* classes to represent integral and floating point points respectively, regardless of which coordinate system the points belong to. Widgets and events may also use one or more coordinate system as well, leaving it up to the programmer to keep track of which coordinate system is used where and to which coordinate system points belong. Although, the unusually comprehensive and useful Qt documentation is a big help in this aspect.

The coordinate systems used by Qt all originate from the top left corner of the screen or widget, with positive directions to the right and down, and there is a global coordinate system based on computer screen coordinates and a local coordinate system for each widget instance. The reason it has become common practice for computers and software to use such coordinate systems probably has to do with the update cycles of the electron guns in older CRT-monitors, which start at the top left corner and work their way down to the bottom right corner of the screen. Mathematical graphs and plots on the other hand, originate from an origin with positive directions to the right and up, and negative directions to the left and down, making the positive y-axis direction opposite in the different coordinate systems, or flipped if you will. Transformation functions, which take this fact into account, as well as considering zoom and scale factors, were thus

needed to convert points between the graphics scene and graph coordinate systems (see section 4.3.1.4 MATHEMATICAL TRANSFORMATIONS).

Additionally, the local coordinate system of graphics scenes is detached and independent, since graphics scene widgets themselves are not displayed on screen. The viewports of the graphics view widgets, which as mentioned above also have local coordinate systems, must be used to display a portion of a graphics scene (see FIGURE 36). Further, the graphics scene is always resized to be the exact same size as the graph image, to avoid having to work with internal offsets inside the graphics scene as well.

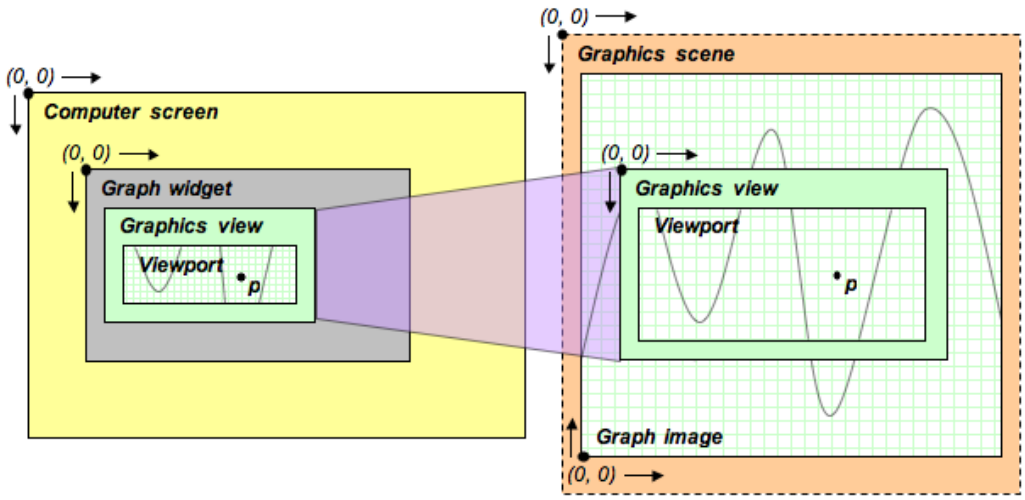


Figure 36. *An overview of the different coordinate systems in use in the frontend components, showing the respective origin and positive directions for each. Point p thus has different coordinates in each of the coordinate systems. Note that, in actuality, the viewport of the graphics view covers the entire graphics view widget, and it is made sure that the graph image covers the entire graphics scene, as mentioned in the text above.*

This means that in order to transform an onscreen point displayed in the viewport of the graphics view to the corresponding point in graph coordinates, there have to be two explicit coordinate conversions made in the correct order; first from view to scene, and then from scene to graph. Thankfully, Qt first implicitly transforms the point from screen coordinates to widget coordinates to viewport coordinates, and it also provides built-in transformation functions between viewport and scene coordinates. It is also important to note that some precision is inevitably lost in the process, due to conversions between the integer pixels and floating point graph points. Due to this fact, a new center point of the viewport, which is always given in graph coordinates, must be at a distance greater than the Manhattan length of the previous and new center points to avoid dislocating the viewport by one or more pixels during updates that should not change the center point.

As a more tangible example based of FIGURE 35 and the transformation functions described in section 4.3.1.4 MATHEMATICAL TRANSFORMATIONS, consider the following: if the computer screen is about 1000 x 700 pixels large and the graph uses a linear scale with a scale and zoom factor of 1, then point *p* in FIGURE 35 is approximately located at (440, 390) in screen coordinates, at (320, 230) in widget

coordinates, at (290, 120) in viewport coordinates, at (350, 310) in scene coordinates, and at (350, 280) in graph coordinates.

Initially, as one can imagine, all these coordinate systems were very easily mixed and mistaken for one another, even with good naming schemes for variables, which lead to potentially disastrous results and bugs that were often difficult to find. Later on, this problem was more or less neutralized by introducing one explicitly named type definition (alias) for each type of coordinate, value, size and geometrical shape used for each of the three coordinate systems (see APPENDIX E - CCDVL API DOCUMENTATION). For example, it is perfectly clear in this context what kind of points the types *ViewPoint*, *ScenePoint* and *GraphPoint* refer to, especially compared to the actual types these are translated to at compile time; *QPoint*, *QPoint* and *QPoint* respectively.

4.3.3.4 Graph image clipmap position

The graph state objects store the bottom left graph coordinate of the graph image to keep track of which graph image tiles to fetch. When triggering a pan-related update it is a simple matter of updating the position in the state object (see section 4.3.3.2 GRAPH IMAGE CLIPMAP AND STATE OBJECTS). However, this position must be recalculated, using the formula below, when the scale or zoom factors change. The idea is to compute a new graph bottom left coordinate by providing a new center coordinate together with new zoom and scale settings. The new graph center coordinate can be trivially obtained using the inverse of one of the transformation methods described in section 4.3.1.4 MATHEMATICAL TRANSFORMATIONS, and the new bottom left can be obtained by again using the inverse transform substituting the bottom left coordinate with the new center coordinate, where the scene coordinate to transform then correspond to the relative offset from the center to the relative bottom left. This is perhaps easier explained and clearer by taking it step-by-step mathematically, as shown below:

Let P_x be the new scene x-coordinate.

Let P_y be the new scene y-coordinate.

Let B be the old bottom left coordinate.

Let $R_x = -S_w / 2$, where S_w is the scene width, in pixels.

Let $R_y = S_h * 1.5$, where S_h is the scene height, in pixels (note the flipped y-axis).

Let A_o be the old zoom and scale ratio.

Let A_n be the new zoom and scale ratio.

Let $f : \mathbb{R}_x, \mathbb{R}_y, b, a \rightarrow \mathbb{R}_x, \mathbb{R}_y$ be the inverse transform for two dimensions using b as bottom left coordinate and a as zoom and scale ratio.

Then the new bottom left is equal to $f(R_x, R_y, f(P_x, P_y, B, A_o), A_n)$.

Conveniently $f(P_x, P_y, B, A_o)$ may be substituted with any graph coordinate to center on.

4.4 Testing and evaluation

A few incremental testing applications, including the graph widget example implementation prototype, which use CCDVL to render and display graphs were made in order to test and showcase the functionality of the library. These applications also serve as examples for how this library can be used by developers to display large datasets. Combine provided some actual scientific test data stored in the HDF5 database format (The HDF Group, 2011), which one of the examples uses. More information about the format can be found on the official HDF5 webpage; http://www.hdfgroup.org/HDF5/.

Besides the feedback received from meetings with Combine, there were some minor and some bigger issues to address, mostly relating to keeping the backend dynamic, to platform-specific Qt behaviour and to weaknesses in the frontend GUI. The collaboration diagrams generated by Doxygen also helped identify and eliminate some circular dependencies in the API structure (see APPENDIX E - CCDVL API DOCUMENTATION). Some of the most important and interesting of these issues are described in the sections below.

4.4.1 API and backend testing and evaluation

Initially, the modules in the API were designed with a template parameter indicating the number of dimensions of the dataset. This proved unnecessary complicated, as the amount of code imported from the header files had grown significantly after the first development iterations and affected module compatibility with the API. Therefore, all modules requiring this information instead have a public constant to keep track of the number of dimensions.

Another module refactorization was made concerning the mathematical transformations, projection, scaling, etc. mentioned previously in section 4.3.1.4 MATHEMATICAL TRANSFORMATIONS, which were initially divided into multiple implementations between the AGG render and the Qt frontend. A lot of redundant code, which was difficult to update due to the specialized use in each module, was introduced in when logarithmic scaling support was being added. A common *Transform2D* class, extracted during refactorization, was instead used for such operations to better keep things separated and avoid the redundancy issues.

The iterator design described in section 4.2.1.1 ITERATORS ANALYSIS AND DESIGN, where the accessed data is an abstract dataset, could conflict with a fast data search, making it more difficult to implement such a search method later. Perhaps a better choice is to actually iterate the data points rather than the datasets, but each iterator must then manage the number of dimensions as well, which was initially avoided by the template design described above.

Valgrind revealed an interesting memory leak (see section 2.1.1 VIRTUAL MEMORY, SWAP AND MEMORY LEAKS) caused by the glue code generated by SIP, due to confusion regarding object ownership between Python and C++. The offending code converted a

list-based Python data structure to a CCDVL abstract dataset and failed to properly free an intermediate allocation during the conversion process. While this problem was easily resolved, the impact was severe as it caused data stored by a memory manager to be duplicated, effectively creating a permanent copy of the entire dataset in memory while loading it from Python.

4.4.1.1 Cache evaluation and alterations

The current clipmap location conversion implementation, mentioned in section 4.3.3.4 GRAPH IMAGE CLIPMAP POSITION, has an accumulating rounding error, which will decrease the efficiency of the cache. It occurs when a user causes the graphics scene to update when panning too close to the edge and boils down to that the bottom left coordinate is represented using a floating point type, which is further dependant on previous values. The effects from this rounding error cause the bottom left coordinate to not retain the exact same value as before when panning back, which will have a negative impact on cache lookups from the GUI module.

The early cache implementation attempted to calculate the difference, expressed as a distance value, between graph tile request configurations, to be able to identify and provide the closest matching graph tiles to try and counter coordinate rounding errors mentioned above. Eventually this feature was dropped, primarily because of suspicious graph tile lookup results and maintenance difficulties, in combination with some other features that did not work out as intended or did not belong in the cache. It was instead refactored and replaced with a standard map data structure, which had its own set of problems. It was plagued with duplicated graph tile entries that looked like data structure corruption, but the problem was finally tracked down to an incorrectly implemented comparator.

4.4.2 Frontend testing and evaluation

It was discovered that, occasionally during updates, calculations using the size of certain components, such as the graphics scene, returned incorrect results. Debugging revealed that some Qt components would sometimes return zero instead of their actual sizes, probably due to previous, not fully propagated, resizes. The size and number of the components used to determine these size changes were always known, and were thus instead used to calculate the correct size of the Qt components in the affected calculations. One more thing that did not work out as smoothly as anticipated was the Qt resource system, which in this case was used primarily to load and use icons. It would not load correctly with the statically-linked library, and the *Q_INIT_RESOURCE* macro had to be used to forcefully initiate the resource system.

During memory leak testing, Valgrind was able to find a few memory leaks and potential memory leaks (see section 2.1.1 VIRTUAL MEMORY, SWAP AND MEMORY LEAKS) internally in several Qt classes. As Qt uses its own internal memory management of *QObject* and its sub-classes through parent-child chains, there was nothing to be done about this. These were hopefully harmless memory leaks,

considering that Qt has already been around for a couple of years and there have been plenty of time and opportunities to test, discover and remove any significant memory leaks. Valgrind was also useful for finding the memory leaks in the frontend components that were introduced by the initial confusion regarding which Qt objects and sub-classes were automatically managed and which were not.

Qt also caused various headaches in regards to multiprocessing (see section 2.3 CONCURRENT PROGRAMING AND MULTIPROCESSING), as it is possible for multiple threads to access the *QObject* class and most non-GUI Qt classes simultaneously, even though but they are designed to be created and used from within a single thread. This means that calling a function of an object created in one thread from another thread is not guaranteed to work. Specifically, a child of a *QObject* must be created in the same thread as the parent object and event driven objects, such as timers, may only be used within the thread that it belongs to. Further, all objects created by a thread must be deleted before deleting the thread itself, and the Qt GUI classes, such as *QWidget*, may only be used from the main thread running the Qt main event loop, i.e. the thread that has called the *QCoreApplication::exec()* function (Qt Project Hosting, 2011). There exist workarounds with simple worker threads, and it is possible to pass instructions directly to the Qt classes in a thread-safe manner, since the comprehensive signal and slots system that Qt provides may be used as a communication channel for signals as well as message passing and even return values. This took some time and a lot of debugging to get to run smoothly without any deadlocks, since *QMutex* objects do not behave quite like regular SMP mutexes due to how Qt handles *QObject*s and threads, as mentioned above. In the end, this meant that the majority of the used *QMutex* objects could be removed, which noticeably improved performance and response time of the frontend and GUI.

Another frontend issue that affected GUI performance was due to how different GUI window managers on different platforms obviously behave differently in some aspects. In this case it was how resizes of windows through the GUI are handled. Some window managers sends one single resize event at the end of the resize, but the native window manager on Mac OS X immediately fires a resize event for even the smallest change of the window size, which floods Qt with resize events during the entire resize operation. For instance, if the graph image and graph image tiles are very small and the graph widget window is enlarged enough so that the graphics view becomes larger than the graphics scene displaying the graph image, then the graph image must be expanded and updated. A flood of resize events will in this case cause a flood of update requests, and in turn a flood of requests to the graph cache and renderer, of which only the result of the last request will actually be used, potentially wasting a lot of resources depending on the size and speed of the cache. A resize event timer was added to avoid wasting resources on intermediate handling and integrity checking, as described in the example above. It activates on resize events of the graph widget and ignores all consecutive resize events until after an idle timeout of 500 milliseconds. The timeout is reset each time a resize event occurs, even if the timer is running, effectively delaying updates until there is a shorter pause in the resizing of the window. This is also part of the controlling response time guidelines stated in section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES, to make the GUI more responsive. It is also worth noting that the resize event is still propagated internally through Qt to the shown GUI widgets,

assuring that the liquid layout is still in effect for smoother and more natural window and widget resizes (see section 4.2.3.1 GUI OVERVIEW AND STRUCTURE).

A check to make sure that the size of the viewport of the graphics view did not exceed the size of the graphics scene (and thus the graph image) during resizes was implemented to allow dynamically increasing the size of the graph image by adding more tiles as needed. This also makes sure that the available panning space stays proportional to its original settings, thus ensuring that the pan triggered updates continue to work as intended by making sure that the invisible pan trigger border is not too tight (see section 4.3.3.2 GRAPH IMAGE CLIPMAP AND STATE OBJECTS).

4.4.2.1 GUI evaluation and alterations

It was decided to add a "make new group from selection" button, rather than automatically making new groups for each selection, to avoid unnecessary computations and bloating the groups list. Another computationally expensive process was the drawing of the grid, especially on lower zoom levels where more lines had to be drawn, due to the grid lines being drawn at intervals set in dynamically rendered graph coordinates rather than static scene coordinates, which also cluttered the graph image and added visual excise (see section 2.7.2 FLOW AND EXCISE). The default grid was changed to a static scene coordinate grid, and options for using the original grid and an additional grid proportional to the zoom level was added as well. The grid is of course only drawn once at the beginning of each update, as it does not change during progressive update iterations, and a glass pane was also added to draw the grid upon and to support future overlays, such as static functions and additional helper lines.

A loading message was added to the graph neighbourhood overview while the graph image was updating, as the two representations of the graph image otherwise could temporarily fall out of sync, which would go against the recommendations of error prevention, giving clear feedback and having good visibility of the system status (see section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES). Graph image updates triggered through panning using the neighbourhood overview were also disabled for the same reasons. Panning back and forth quickly between opposite edges would also trigger a lot of unnecessary updates, making it hard to pan the entire neighbourhood overview without accidentally triggering an update. On a side note, a purely cosmetic quirk in the presentation of the GUI, probably due to the implementation of the Qt framework on different platforms, was discovered in the black-and-white mask used by the neighbourhood overview; its color was inverted depending on platform. Another cosmetic Qt quirk, with potentially greater impact on the GUI, was found after testing the library on different platforms, and realizing that the operating system dependent default look and feel font had precedence over the internally explicitly specified fonts for most standard GUI components. It was decided to override this behaviour, to avoid potential derailment of the layout due to changing font family and font size, by using a global style sheet in Qt. Common standard monospace fonts were chosen to make sure that the fonts would most likely be present and behave similarly on all platforms.

4.4.2.2 Graph interaction tools evaluation and alterations

The zoom in, zoom out and zoom select tools were merged into one zoom tool with a single zoom button labelled with a magnifying glass icon. The zoom tool zooms in on left click by default, zooms out on left click while the option key is depressed and zooms in to the selected rectangle with click and drag, or zooms out in inverse proportion to the selected rectangle size with option-click and drag. Pressing and releasing the option key also changes the sign of the magnifying glass cursor icon from plus to minus, and back to plus again respectively, to indicate its current function.

Double clicking the zoom button would reset the zoom level to 100%, but Combine suggested adding a specific tool button with the same functionality, as it was easier for the user to find. An abort mechanism for the zoom select functionality was further requested, as it was pointed out that it was not an uncommon scenario where the user might want to change the anchor point of the selection rectangle or abort the operation altogether after initiating the zoom selection. Such user behaviour is also reflected by the changes in midstream design pattern defined by Tidwell (2011). Pressing the escape key seemed the most intuitive and natural way of doing this, based on experience and the habituation design pattern (Tidwell, 2011). Combine also suggested including an option of having the toolbar at the top above the graph image, as is common in most applications, thus also supporting the habituation design pattern in this aspect as well.

Initially, Combine wanted the stop and reload button to automatically reload the current graph view after it had been pushed. However, after some discussion they agreed that it would be better to actually be able to fully stop the update process, which might have got stuck in a very large (or possibly bug-induced infinite) loop, rather than reloading only to get stuck again. All the additions mentioned above in this section were implemented as suggested in the following iterations.

The mouse cursor icons provided by Qt that were used by the proposed library stand out well against both dark and light backgrounds due to their white outlines. The created black lasso tool cursor icon has no such outline, and tended to disappear over darker regions of the graph image. An outline was added at first, although a different solution was used in the end, as the outline looked a bit too bulky and ugly. The lasso select tool cursor icon was made to dynamically invert its color depending on the color of the background below the cursor. This was again inspired by Adobe Photoshop and provided good results with better contrast. Initially a copy of the entire rendered graph image was used for color lookup, which was composed by graph image tiles stored as instances of the *QImage* class rather than instances of the *QPixmap* class, since only the former allows pixel lookup among other image manipulation functions. The disadvantages with this first approach was that the graphics scene items only handle pixmaps, and that the conversion from an image to a pixmap is quite computationally expensive, as well as creating an additional copy of the graph image to keep in memory. This was eventually avoided by having the lasso select tool render and update a (*QImage*) snapshot of the currently shown graphics view content, which was used for color lookup when the mouse cursor moves within the graphics view. This also allowed the graph image tiles to be stored as pixmaps, reducing the amount of used memory,

speeding up the update process slightly and made the lasso select tool less dependent on other classes.

The helper lines of the point select tool were given a similar makeover as the lasso select tool icon to make them stand out with good contrast on both light and dark surfaces. They were changed from solid black dashed lines to semi-transparent white lines overlaid with semi-transparent black dotted lines, almost functioning as some kind of inverted camouflage, making the helper lines stand out while keeping the transitions over differently colored backgrounds smooth and unnoticed. Section 2.6.2.2 DATA ENCODING METHODS AND GUIDELINES discusses the use of color further.

4.4.3 Initial performance and test results

A few larger performance tests were conducted close to the end of the development phase. TABLE 3 shows the results of a late test performed with the graph widget example implementation prototype running on the test notebook specified in section 1.4 DELIMITATIONS. The six columns of TABLE 3 contain the following:

- The first two columns contain the number of graph image tiles used and their respective sizes. The check that dynamically adds more graph image tiles as needed, described in section 4.4.2 FRONTEND TESTING AND EVALUATION, was disabled during the test, partly because it was not fully implemented at the time, but primarily to make sure that the results were not skewed by adding additional tiles.
- The third column contains the time taken to generate the data, pass it from Python to C++ and finally save it to the temporary swap file. These times are roughly the same as an identical dataset, generated in the same manner as in the prestudy (see section 3.2 PRESTUDY), is used throughout the entire test.
- The fourth column contains the average time it takes to redraw the entire graph image for each iteration of a progressive update, i.e. the time during which the GUI may become noticeably slow or unresponsive.
- The fifth column contains the actual rendering time needed by the renderer to finish a batch rendering request containing all the graph image tiles. It was made sure that the graph image tile cache was empty during the test to not affect the raw results. Although, using one or more cached graph image tiles would of course have improved the results proportionately with the current renderer and memory manager implementations, as deducted by section 4.2.1.1 ITERATORS ANALYSIS AND DESIGN.
- The sixth column contains the system wide swap usage during rendering, which is of course strongly dependent on the used computer hardware, operating system and other running applications, as described in section 2.1.1 VIRTUAL MEMORY, SWAP AND MEMORY LEAKS.

Table 3. *Late test results with high total rendering times due to sub-optimal rendering. 25 million points of generated data was loaded and rendered on the test notebook with the graph widget example implementation prototype.*

Graph image tiles (width x height)	Graph image tile size (width x height in pixels)	Data load time (s)	GUI update time (s)	Total rendering time (s)	System swap use (%)
4 x 4	1000 x 1000	16.11	0.38	1031.60	0
4 x 4	500 x 500	16.05	0.08	861.50	0
4 x 4	250 x 250	16.09	0.02	833.88	0
4 x 4	125 x 125	16.06	0.00	827.08	0
5 x 5	1000 x 1000	16.21	0.59	1894.89	0
5 x 5	500 x 500	16.14	0.12	1353.30	0
5 x 5	250 x 250	16.08	0.03	1330.08	0
5 x 5	125 x 125	16.07	0.01	1912.24	0
8 x 8	1000 x 1000	16.07	2.05	6691,79	30
8 x 8	500 x 500	16.10	0.41	4369.93	1
8 x 8	250 x 250	16.89	0.08	3589.56	0
8 x 8	125 x 125	16.24	0.02	3478.48	0
10 x 10	1000 x 1000	16.08	3.11	10450.17	56
10 x 10	500 x 500	16.04	0.61	7831.47	0
10 x 10	250 x 250	16.06	0.13	5705.51	0
10 x 10	125 x 125	16.04	0.03	5511.91	0

Although the test is a bit rough as it was run on an unfinished version of the library just before the final iteration, it still serves as useful comparison with the final implementation to help highlight the results of the optimizations made in the last iteration. One such optimization was made to the rendering loop, since the renderer initially required a very long time to finish, as seen in the results of the above test.

5 Result

The purpose of this thesis, described in section 1.3 PURPOSE, was to simplify analysis of large amounts of scientific data by creating a small modular and extensible cross-platform graphics library, intended to run on ordinary workstations, capable of handling such data and present it through highly interactive plot graph widgets, while also providing Python bindings for the library API.

5.1 CCDVL library and API

The resulting cross-platform library was named CC Data Visualization Library (CCDVL), where CC is a reference to the ".cc" C++ source code file extension. It can certainly be used to facilitate and simplify analysis of large amounts of scientific data on ordinary workstations, due to the use of clipmap rendering and clever memory manager and iterator classes that dynamically read and process the data in smaller subsets, which are within the size and resource limitations of workstations. This is done in a separate thread and data is stored to a memory mapped file, ensuring good responsiveness and interactivity as well as a better balanced I/O workload at the cost of additional hard drive space and a somewhat increased CPU usage. A clipmap tile cache and progressive updates are also used to further improve performance and responsiveness. See section 4.2 ANALYSIS AND DESIGN for more details.

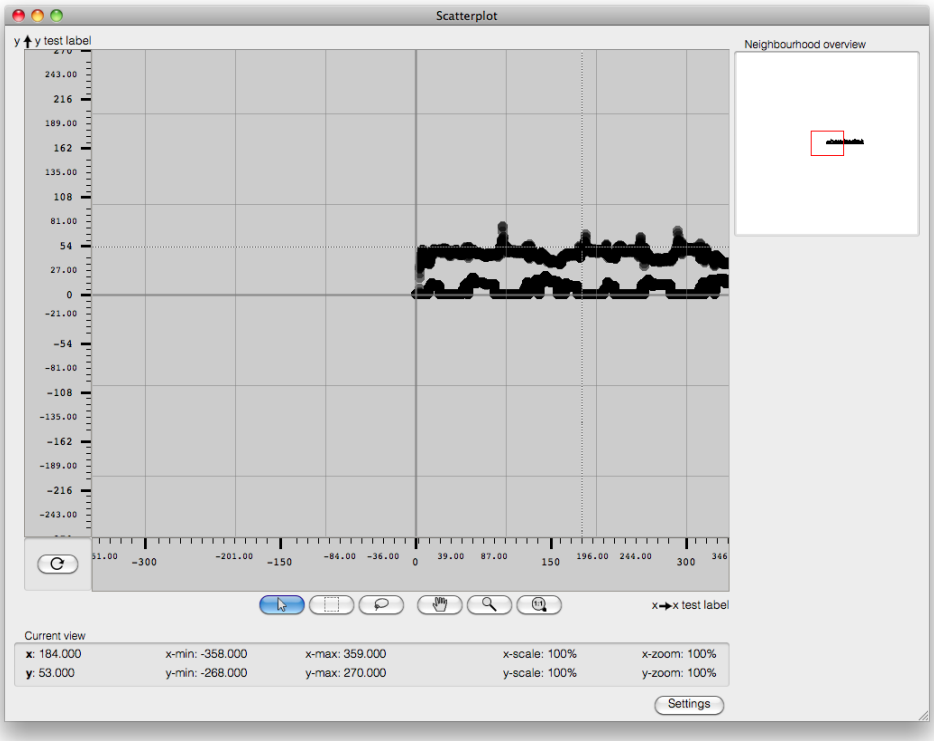


Figure 37. *A screenshot of the GUI of the final graph widget example implementation prototype showing a smaller sample of scientific data provided by Combine.*

The final graph widget example implementation prototype seen in FIGURE 37 above provides two-dimensional scatter plots with semi-transparent points for better cluster

visibility and interaction in form of different graph interaction tools with selection, zoom and pan functionalities, as well as providing various GUI and data settings, as described in detail in the previous chapter. Python bindings for basic functionality are made available through SIP and PyQt, all non-functional requirements and all other (non-optional) functional requirements listed in APPENDIX A – INITIAL REQUIREMENTS AND REQUESTS were met, with the exception of the time series graphs and its associated requirements, unfinished logarithmic scale support and selected data groups management. A few optional functional requirements were implemented as well, such as the possibility to render lines between points and configurable colors in the graph.

The CCDVL library was published under the LGPL license in accordance with the license boilerplate provided by Combine, despite the fact that they initially explicitly requested that the library should be licensed under a BSD license. The Python bindings were licensed under the GPL, as explained previously in section 3.3.2 PROGRAMMING LANGUAGE AND SOFTWARE LIBRARIES.

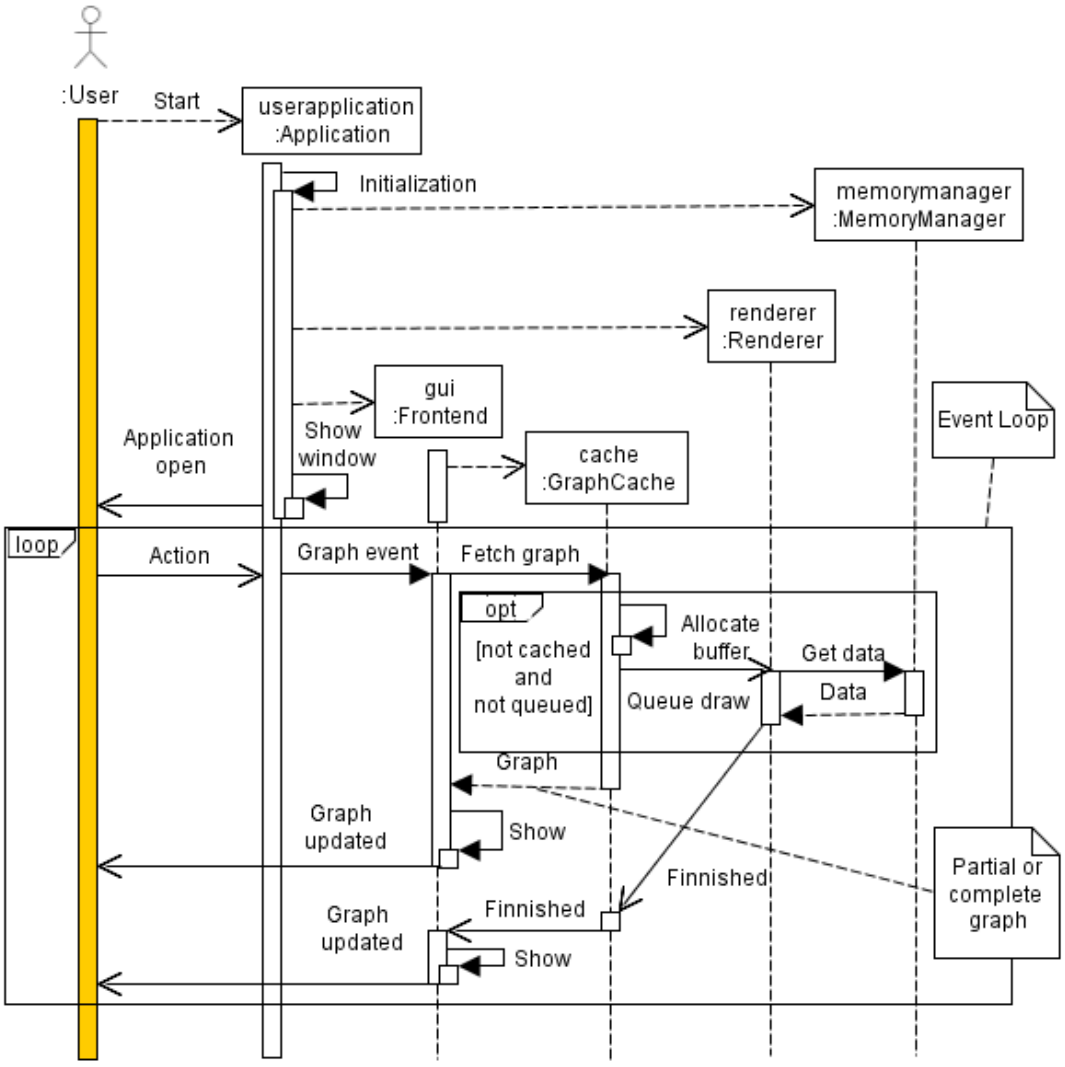


Figure 38. *A sequence diagram of typical run of an application using the CCDVL library, such as the graph widget example implementation prototype.*

APPENDIX B - MANUAL AND USER GUIDE contains more details on the GUI and how to interact with it from a user perspective, and FIGURE 38 above displays a sequence diagram showing user interaction and the internal workings and communication between the CCDVL modules of a typical run, to give the reader an overview and a better idea of the internal workings of the library. The components of the modules can be found in the class diagrams in FIGURE 33 and FIGURE 34 in the previous chapter.

As a side note for the curious reader, it is worth mentioning that the resulting version of CCDVL is rather big, as is common with APIs and software libraries. It consist of just under 7 500 lines of source code, out of which 57% are comments, 25% are code only, 15% are empty and 2% are both code and comments.

5.2 Performance and test results

The exact same performance test described in section 4.4.3 INITIAL PERFORMANCE AND TEST RESULTS was repeated with the final version of the library, yielding the results found in TABLE 4. Compared to the previous CCDVL test results, found in TABLE 3 in the aforementioned section, the most significant differences are improved data load time, somewhat slower GUI update for some test cases and that the renderer is now able to finish within a reasonable timeframe. The change in GUI update time is caused by extra computational overhead due to an added condition in the graph update loop used to lessen the load on the cache and renderer by reducing requests of completely rendered and updated graph image tiles, as described in section 4.3.3.2 GRAPH IMAGE CLIPMAP AND STATE OBJECTS. The improved total rendering time is due to both compiler optimizations and a renderer optimization, where some misplaced code was extracted from inside the rendering loop, reducing the computational overhead.

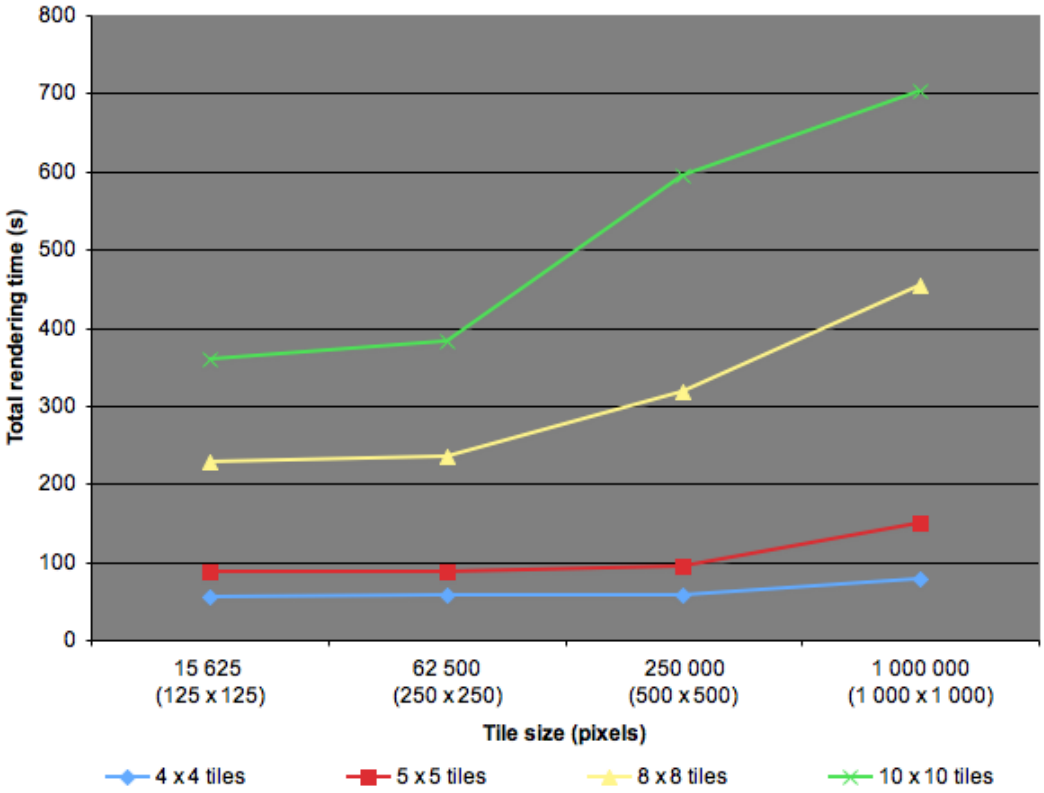
The automatic graph image tile padding for test cases where the rendered graph image was smaller than the viewport of the graphics view, described in section 4.4.2 FRONTEND TESTING AND EVALUATION, was turned off to not affect the results as well as to be able to better compare the results from the previous test (where it was not implemented). The meaning of the columns of TABLE 4 are also equivalent to those of TABLE 3, described in more detail in section 4.4.3 INITIAL PERFORMANCE AND TEST RESULTS, but can be summarized as follows:

- The first two columns contain the number of graph image tiles and their sizes.
- The third column contains the time taken to generate the data, pass it from Python to C++ and save it to the temporary swap file.
- The fourth column contains the time it takes to redraw the graph while rendering.
- The fifth column contains the actual rendering time needed by the renderer to finish a batch rendering request.
- The sixth column contains the system wide swap usage during rendering.

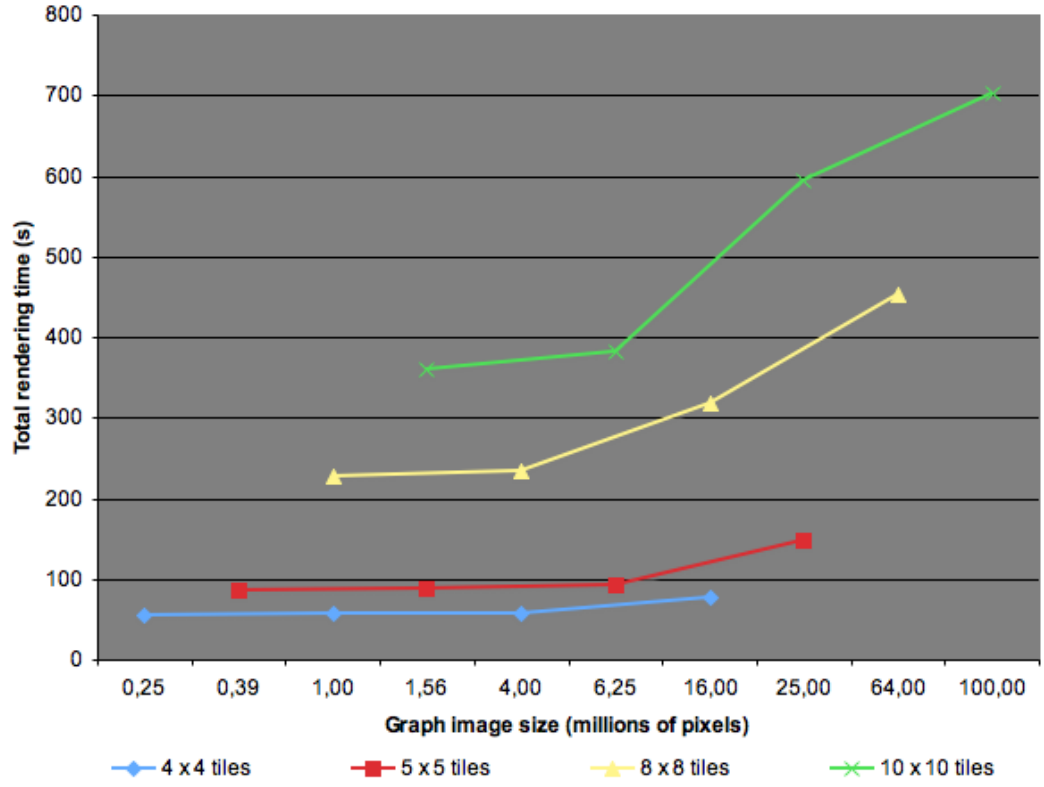
Table 4. *Final test results with improved total rendering times. 25 million points of generated data was again loaded and rendered on the test notebook with the graph widget example implementation prototype.*

Graph image tiles (width x height)	Graph image tile size (width x height in pixels)	Data load time (s)	GUI update time (s)	Total rendering time (s)	System swap use (%)
4 x 4	1000 x 1000	15.55	0.5	78.16	0
4 x 4	500 x 500	15.45	0.08	57.15	0
4 x 4	250 x 250	15.56	0.01	56.61	0
4 x 4	125 x 125	15.57	0.00	55.19	0
5 x 5	1000 x 1000	15.48	0.75	148.92	0
5 x 5	500 x 500	15.51	0.13	93.77	0
5 x 5	250 x 250	15.93	0.02	87.66	0
5 x 5	125 x 125	15.65	0.00	86.88	0
8 x 8	1000 x 1000	15.54	1.92	453.36	48
8 x 8	500 x 500	15.52	0.53	317.98	1
8 x 8	250 x 250	15.54	0.06	234.15	0
8 x 8	125 x 125	15.52	0.01	228.71	0
10 x 10	1000 x 1000	15.47	2.59	703.05	53
10 x 10	500 x 500	15.52	0.77	593.69	0
10 x 10	250 x 250	15.49	0.10	382.58	1
10 x 10	125 x 125	15.48	0.02	359.80	1

From these results it can be concluded that software rendering time increases heavily with the number of graph image tiles and also somewhat with graph image tile size, which can clearly be seen in FIGURE 39(a). This relation is also true even when the whole graph image is represented with the same number of pixels; e.g. rendering a graph image of size 1 000 x 1 000 pixels with 4 x 4 tiles of size 250 x 250 pixels is faster than rendering the same graph image with 8 x 8 tiles of size 125 x 125 pixels, as seen in FIGURE 39(b). FIGURE 39(a) also shows that the rendering times increase more rapidly for the two graph image tile configurations with the most and largest graph image tiles, which were caused by the test notebook running out of memory due to the large tile image buffers. This promptly made the operating system temporarily unusable as it was forced to start swapping (see section 2.1 OPERATING SYSTEM MEMORY MANAGEMENT), which also occurred for the corresponding test cases in both the previous and following tests as well.



(a) *graph image tile configurations and total rendering time*



(b) *graph image size and total rendering time*

Figure 39. *Relations between graph image tile configurations (a), graph image size (b) and total rendering time (with an empty cache) in the final test, based on the data in* TABLE 4.

The final test was also run on the more powerful test desktop computer, specified in section 1.4 DELIMITATIONS, to see how much this would affect the results. TABLE 5 shows the outcome, and it is clear that the loading times improved while the update and rendering time improved slightly but scaled worse. It scaled badly enough to have the test notebook surprisingly outperform the test desktop computer in the larger test cases – which could be related to some hardware issues that the test desktop computer have shown symptoms of. Nonetheless, the test showed that CCDVL runs smoother (with good configurations) on better hardware, as one would generally expect.

Table 5 *Final test results run on the more powerful test desktop computer with the graph widget example implementation prototype.*

Graph image tiles (width x height)	Graph image tile size (width x height in pixels)	Data load time (s)	GUI update time (s)	Total rendering time (s)	System swap use (%)
4 x 4	1000 x 1000	9.82	0.35	63.82	0
4 x 4	500 x 500	9.87	0.06	51.80	0
4 x 4	250 x 250	9.95	0.02	50.76	0
4 x 4	125 x 125	9.93	0.00	48.35	0
5 x 5	1000 x 1000	9.79	0.61	116.83	0
5 x 5	500 x 500	9.91	0.11	88.76	0
5 x 5	250 x 250	9.89	0.03	82.14	0
5 x 5	125 x 125	10.04	0.01	90.91	0
8 x 8	1000 x 1000	9.90	2.78	894.97	0
8 x 8	500 x 500	9.89	0.42	296.53	0
8 x 8	250 x 250	9.90	0.08	296.53	0
8 x 8	125 x 125	10.04	0.02	216.61	0
10 x 10	1000 x 1000	9.91	3.80	1564.75	40
10 x 10	500 x 500	10.08	0.71	672.59	0
10 x 10	250 x 250	10.02	0.04	396.41	0
10 x 10	125 x 125	9.88	0.08	378.50	0

Worth mentioning is that the frontend also supports rectangular graph images constructed by non-identical number of graph image tiles, such as 4 x 5 or 5 x 22, as well as any tile size larger than zero. Rectangular sizes were used in the tests to keep things simple and to make it easier to compare the results, as it is the total number of graph image tiles, the graph image size and the size of the dataset that really matters.

5.3 Comparison with prestudy results

All the performance tests were intended to estimate both the graph widget performance and the operating system performance as a whole during rendering and interaction, which is why the results are still interesting despite the fact that the measurements depend on the hardware, operating system and other applications running in the background. All of the cross-platform libraries examined in the prestudy had multiple rendering backends, supported Python and provided both scatter plots and time series graphs with both linear and logarithmic scaling (see APPENDIX C - PRESTUDY RESULTS). However, they could not handle large amounts of data as there was no memory management, highlighting the main problem described in section 1.2 PROBLEM. None of the examined libraries were multithreaded, which was clearly noticeable in the interaction performance tests, and it should be pointed out that even though a varying degree of interactivity was available between the rendering backends, these libraries does not seem to focus on interactivity, but rather on providing more static graphs and graph images.

The tests in the previous section show that CCDVL is able to handle the full test dataset rather smoothly, unlike any of the libraries examined in the prestudy. Few of the rendering backends of these libraries could manage the full test dataset without crashing. These libraries were however able to handle half of the test dataset with a severe impact on system and application usability while rendering, as described in APPENDIX C - PRESTUDY RESULTS. Furthermore, each of the examined libraries had to immediately redraw the graph for every user interaction affecting the graph view, traversing the entire dataset each time, making any such interaction more difficult and time consuming to perform on very large datasets. CCDVL mostly avoids this by rendering the graph as a set of graph image tiles and manually managing memory usage of datasets as previously described in section 5.1 CCDVL LIBRARY AND API and chapter 4 DEVELOPMENT, while simultaneously utilizing progressive updates and a graph image tile cache.

Numerous suggestions on how to further improve CCDVL are provided in section 6.6 FUTURE WORK, ADDITIONS AND EXTENSIONS, and as a closing comment, it is interesting to note that the PLplot library, which performed best out of the three libraries examined in the prestudy, is also the library that seems to be most similar to CCDVL.

6 Discussion

This was a very large thesis overlapping many huge fields, such as software engineering and information visualization described in chapter 2 THEORY, as well as comprising parts that could easily be used as subjects for independent theses, such as creating the optimal data structure for the memory manager discussed in section 4.2.2.1 MEMORY MANAGER ANALYSIS AND DESIGN. There was a lot of potential in CCDVL to explore and develop, and still is as noted by the suggestions taken off the top of our heads in section 6.6 FUTURE WORK, ADDITIONS AND EXTENSIONS below. As such, the thesis also took quite a bit longer to complete than normal. One of the major causes, if not the biggest cause, of the thesis' tendency to drag on was initial administrative setbacks. While these were being resolved we could not work full time on the thesis, and we were stuck at a work rate around 50-70% for at least half of the thesis duration.

6.1 Method discussion

If we were to take on such a huge project as this thesis again, we would generally:

- improved on planning, have more and clearer milestones, as well as a more realistic time plan with better follow ups,
- prioritized the requirements better to make sure that all important bits were in place,
- keep iterations smaller and shorter, since many of our iterations were both too large and took too long to complete,
- work closer with Combine, by doing more of the thesis work out of their office, and
- perform a wider and more accurate prestudy.

Otherwise the chosen methods worked out rather well, although it was a bit hard to do any thorough user tests of the results and iterations of an on-going development of a fresh API. As mentioned in section 3.3.2 PROGRAMMING LANGUAGE AND SOFTWARE LIBRARIES, the Qt Designer application was not used due to bad experiences with similar tools, but choosing to use it from the start would probably have saved some time initially. But the question remains if it would not have required more effort when making changes and during debugging, as our understanding of the Qt workings probably would have been less compared to working directly with the code. It would thus been interesting to make the same GUI and changes again in Qt Designer and compare the generated code, time and effort spent to see if it actually proved to be a good GUI-builder tool or not.

6.2 Prestudy

The fairness of the prestudy is somewhat questionable, as we only looked at a few free Python libraries and libraries that had Python bindings, which used a permissive license, and only performed tests on the built-in interactive GUIs. It should definitely have been

expanded to cover a few non-Python libraries and possibly also one or two proprietary solutions to be more valid. The studied libraries are also intended to be basic graph renderers, with the exception of the interactive backends, but they still worked as the very rough estimate they were intended to be. We were pretty satisfied to be able to tell if CCDVL was slower than, on par with or even faster than some existing commonly used libraries. This at least gave us a hint of that we were going in the right direction.

Finally, an odd problem occurred when verifying test results; PLplot refused to build the Python bindings while we were trying to increase the granularity of the benchmarks, which made the PLplot test results less precise as only a few initial tests could be made. It would have been interesting to test this library more as it was somewhat similar to CCDVL, as mentioned in section 5.3 COMPARISON WITH PRESTUDY RESULTS.

6.3 Development discussion

This section discusses the design and development of the CCDVL library and API, as well as the internal modules and the external python bindings. Image buffer formats, data compression and the possibilities of hardware accelerated rendering are among the discussed topics.

6.3.1 API and library specification

The library was wrapped into a common namespace and each module was also wrapped into its own namespaces, while some module classes were not. Arguably, they should all have belonged to their respective namespace, like in the case with the memory managers, which also provide an iterator in addition to an implementation of the abstract class *MemoryManager*. In hindsight, the naming conventions of the frontend classes were perhaps not the best either, as they probably are a bit too similar to the ones used by Qt to be unambiguous. The naming conventions seemed logical at first, before any namespaces were enforced, and it stuck without us giving it any more thought.

A few things were left out, either on purpose or because we did not come to think of it until much later, of which two things stand out. Firstly, the specification does not state what kind of image buffer to use, which allows any pixel format; making the API more flexible at the risk of incompatible module implementations. Initially two different pixel formats, one without an alpha channel and one with, were to be supported. Primarily to be able to handle groups as transparent overlays and thereby avoid re-rendering all tiles when a selection is hidden. This brought up an unexpected question in how to handle other pixel formats. In the end it was up to the implementation to decide freely. A good idea might be to enumerate some of the most common pixel formats, and then either let developers choose one of these, or attempt to auto-detect a compatible configuration to avoid unpleasant surprises such as incorrect colors and flipped or mirrored image tiles.

Secondly, is the ordering of the data; the order in which a memory manager iterator traverses held data is important because it currently affects how the renderer attempts to draw a line between points, as well as affecting the group selection iterator. It is preferable if the original input order could be preserved, a wish also expressed by

Combine, but using a plain tree data structure would normally sort the data causing this initial ordering to be lost. One way to counter this is to add additional metadata to keep track of the order internally, which leads to the next problem of figuring out how the memory manager iterator should mark sparse ordering (a sorted list with empty gaps) that may occur when data points in the middle of the input order do not match the search criterion, i.e. is not within the given bounding box used by the search. While this problem was never solved it matters little at the moment since the implemented memory managers do not support search.

Data compression, which could be used to improve memory usage efficiency, is not used in the current implementation to avoid making additional copies of data in memory and in order to save time. However, there is nothing preventing a memory manager implementation from compressing stored data. The image tile buffer format that the API uses to transfer tiles from a renderer to a GUI could also be compressed since no pixel format was standardised (as stated above), but this will likely prove challenging to support as well as requiring both a new renderer and GUI module capable of directly manipulating and displaying a compressed image.

An interesting but unintended side effect, which lies outside the purpose and scope of the thesis, due to passing graph image tiles through the API strictly as image buffers and implementing the asynchronous rendering as a tile message queue to handle batch request, is the possibility to divide the library between the renderer and cache with a network interface in a client-server solution. This would allow data and graph image tile rendering to be done on a large server with more capacity than a small desktop or notebook. Both the server and the client should in this scenario send compressed images and use a cache in order to reduce network usage and to maintain good performance for multiple clients viewing the same dataset.

6.3.2 Modules and components

Since Combine requested a graph rendering library that is capable of handling large datasets while keeping reasonable performance with advanced interactive functionality, there was thoughts about utilizing hardware acceleration, such as OpenGL, to ensure fast rendering separated from the graph rendering thread. In our tests we have experienced that AGG performance is slower when rendering lines or larger data point representations. The CCDVL API ended up not supporting hardware accelerated rendering both due to that AGG does not support it and that no separate OpenGL rendering was implemented. Unforeseen time sinks and a limited timeframe prevented hardware acceleration from finally being addressed. The intent was to use the software renderer as a reference implementation, simply because the API is supposed to be completely platform independent, making hardware acceleration an optional feature.

An outline for an OpenGL renderer implementation that conforms with the current API would probably use frame buffer objects, with the intention to ask the graphics card to draw the data on the image buffer located on the card and retrieving the image buffer upon completion. Although, this method has an interesting shortcoming in that to conform with the API the image buffer must first be retrieved from the card, passed to

the GUI and stitched together in the software, before finally sending it back to the graphics card – when it is possible to display it immediately by simply telling the graphics card to directly display the image at the correct location on screen. This would of course also require that the graph view was implemented using (for example) OpenGL, which will also reduce the cost of the computationally expensive image buffer to image objects conversions. A few other possible gains from hardware accelerated rendering is of course that the memory used for image buffers could be kept in video memory and that the coordinate transformations could be performed directly on the GPU, offloading the CPU further. However, it is important that the system load is balanced, and putting the entire load on a low-end graphics card to save CPU power and memory could be counterproductive.

The *QtGraphViewFrame* class ended up containing a bit too much functionality, and should have been refactored into at least one additional helper class and separate more general (possibly template) graph axes components to better follow good programming practice and the object oriented paradigm as well. Time ran short however, and there is also no explicit functionality for prerendering graph images or graph image tiles, which can be achieved quite easily by breaking out some code of one of the graph update functions and making it a bit more general. Prerendering functions should perhaps have been part of a frontend interface, and an interface would also have helped to keep the frontend and GUI components more modular and better separated. See section 6.6.2 FRONTEND AND GUI below for more details.

As a side note, the similar simpler Qwt library (Rathmann, 2011), a library with Qt widgets for technical applications, providing some graph widgets and components, was examined briefly for possible use of components. Although, it seemed too much work trying to integrate and modify these to work with our API, as there probably would have been too many components to modify, and it thus seemed better to focus on making the things we needed from scratch, rather than getting to know, modifying and being restricted by the Qwt components.

6.3.3 Python bindings

There are several issues to address with the current Python bindings; they need to be refactored into several modules and only the basic functionality is supported. But more alarming is the excessive data conversion between Python and C++, which is perhaps not surprising, but memory managers also copies and store the data. The additional copy between Python and C++ should therefore be avoidable by directly copying the data into the memory manager, which would probably also improve the data load time of the performance tests described in the previous chapters. However, Python will wait for the first data copy operation to finish before being able to continue, which is important when it comes to both concurrent data insertion and Python performance.

In general, these problems stem from the fact that we were not familiar with SIP and that it thus took quite some time to get started with them and even more time to successfully export the basic functionalities of CCDVL. It is also interesting that supporting arrays is quite difficult as SIP cannot handle variable length arrays well, and

is furthermore unable to handle multidimensional arrays without additional code. This caused a few changes to the original API design, changing a few methods' argument types from array pointers to vector objects, yielding both a more type-safe and C++ like API.

6.4 Result discussion

The bottleneck unexpectedly appears to be the CPU rather than I/O as first anticipated (see 2.1.1 VIRTUAL MEMORY, SWAP AND MEMORY LEAKS), of course the fact that the CPU is not forced to wait for I/O is a good sign that the anticipated problem has been solved, but could also be indications that the current renderer implementation is inefficient. A simple straightforward way that should boost the performance and improve the efficiency of the library is of course to run it on computers with better hardware, especially those with large amounts of RAM and disks with greater IO capacity, such as SSD disks or striped raid arrays. However, the test results shown in TABLE 4 and TABLE 5 in section 5.2 PERFORMANCE AND TEST RESULTS are inconclusive as these both show a performance boost and lower efficiency, which was unexpected, especially considering that the notebook managed to outperform the desktop computer for the larger graph sizes. The tests could have been repeated on a more stable desktop computer if there had been time, which would likely have given better and more expected results. Running the tests on a typical workstation actually used by Combine, as well as testing other datasets of gigabyte sized data would also have been interesting.

Another important aspect is of course the potential and possible applications of the CCDVL library. Its purpose is to simplify the analysis of large amounts of scientific data, which offers possibilities and advantages to both users and businesses alike. CCDVL can for instance be used as a component in a data analysis application, in a larger analysis framework or as a step in an analytical process to provide an overview of large datasets with the possibility to quickly drill down, select and extract relevant subsets of the data for further analysis. This can boost efficiency by saving much time, as the interaction in the CCDVL graph widget is responsive throughout the entire drill down process, which in turn reduces the time needed before relevant data can be passed on and processed. Thus results, such as insights into the data or the creation of mathematical models, can be achieved faster. Additionally, this can all be done on the go, as CCDVL can be run locally, without the need for any Internet connection, server or powerful hardware, which is a big advantage for the traveling analyst who wants to efficiently utilize all possible working hours.

6.5 Lessons learnt

Large projects, such as this thesis, require a lot of planning, which we already knew of course. But we realized that we did not do enough planning, especially when some iterations started to drag on, and that it would perhaps have been better if one of us took on the role of project manager and planner a bit more seriously to enforce and revise the time plans in better ways. Another management-level bump in the road that we encountered was a lot of SVN conflicts associated with sparse larger commits, caused by the unspoken consensus of not committing code that did not compile. This and the fact that we had divided the work into the frontend and backend, meant that some larger

additions and changes was not committed until completely finished rather than in smaller steps, which would have reduced the number of SVN conflicts. Perhaps using another versioning and revision control system with support for branches, such as GIT or Mercurial, would have suited this kind of approach better.

We both had a good idea how operating systems manage memory and how virtual memory and swapping worked. Using a low level memory API was very interesting, partly because it was easily available, but mostly because of its capabilities. We generally improved our C++ skills as well, and the Google C++ Style Guide, which is based on sound ideas that reinforce good programming practice, helped us along the way. We also gained some experience working with The Doxygen documentation system, which is similar to the Java Javadoc tool that we have both used before, as well as working with HDF5 database basics, such as opening database files and reading data using the C/C++ API with the testing applications.

Neither of us had worked (much) with Qt before, meaning that we were in for a few surprises, most of which are described in section 4.4.2 FRONTEND TESTING AND EVALUATION. Another one is that Qt projects requires precompiling, which is not actually a surprise but nevertheless ended with CCDVL using CMake to generate platform-specific build scripts to avoid maintaining project makefiles, as mentioned in section 3.3 DEVELOPMENT METHOD AND SOFTWARE LIBRARIES. This way we learned first-hand that CMake is a powerful tool that simplified extending CCDVL with new Qt, SIP and Python source code files and components.

We also learned that SIP is not fully automated and needs both hints and in some cases glue code that use the Python C API. To sum things up, SIP was quite difficult to get started with, and the most challenging parts were some of the more advanced features that were CCDVL was required to export (see section 6.3 DEVELOPMENT DISCUSSION for more details). Of course, SIP still fulfilled its purpose of simplifying the generation of the necessary C code that connects a C++ library with Python, which would otherwise be cumbersome and very time consuming to keep updated.

6.6 Future work, additions and extensions

There are numerous things left to address that essentially belong to few different groups; those that require the API to be revisited, those that can be addressed almost immediately, those that do not depend on the API but possibly on other parts, and finally those that require new modules. These are summarized and listed below.

6.6.1 API and backend

- **Ensure full cross-platform support;** Combine wished for platform support across Windows, Mac OS X and Linux, which was intended to be supported through the use of libraries which already support these operating systems. While this is true for Qt, neither Pthreads or POSIX memory management are natively available under Windows. A possible

workaround is to use Cygwin to compile the affected CCDVL modules for Windows (Faylor, C. et al. 2012). One could also use the included Windows implementation of Pthreads, which is currently not used by CCDVL, as a substitute for the missing APIs when compiling for Windows. However, POSIX memory management, namely the *mmap* and *msync* functions, are still missing under Windows and equivalent memory management functions from the Windows API, such as *VirtualAlloc* and *CreateFileMapping* (Microsoft Corporation, 2012a), must be used instead.

- **Hardware rendering;** Instead of only using image buffers, a window handle or canvas could be used to support OpenGL or similar, but must be optional and any component that can use it must check that it is implemented and fully supported on the current platform and computer. The current cache implementation will also need to be updated to reflect such API changes.
- **Sparse data ordering support;** Capability to indicate and decide when a line between two data point leave the area of interest, primarily an iterator enhancement that will improve and simplify the rendering process. This is also a prerequisite to be able to quickly find all data points within a bounding box.
- **More rendering progress information;** Currently the cache observer only includes information about any batch rendering start or completion. Progress on individual tiles will for example likely be interesting, however updating this too often could reduce the performance of the render.
- **Configuration observer interface;** Simplifies caching and allows settings to change on cached tiles while automatically updating the rendered graph tile.
- **Renderer configuration improvements;** Remove line color from group and changing line color to be a general configuration for the renderer, which will simplify how renderers are supposed to handle grouped data points.
- **Additional memory manager metadata;** The memory manager could hold more metadata regarding the dataset, such as labels and global minimum and maximum for each dimension of the data.
- **Dataset preprocessing;** Common preprocessing steps, such as sorting, extracting or excluding ranges of data could of course be added, although it can be argued that it is not the responsibility of CCDVL but rather the application using the library.
- ***k*-dimensional groups;** A group is not strictly a two dimensional polygon representing a selection but an abstract class representing a bounding box with an arbitrary number of dimensions and a decision function. For now, groups are strictly two dimensional (see section 4.3.1.2 GROUPS), just like the AGG renderer, but it should be possible to extend them past these two dimensions.

- **Logarithmic scaling;** Currently this scaling method is missing validation for infinity and NaN handling by classes using the scale and zoom transform methods. This causes axes to range from NaN to NaN, making it impossible to determine where to draw data points. Note that it could also be possible for this to happen with linear scaling, if one of the view coordinates reaches infinity.
- **Group hierarchies;** No behaviour for group intersections was specified by the API, and consequently no set operations were implemented for groups. One idea was to consider a group to contain a list of sub-groups were each group may not extend outside its parent nor overlap other leaf groups, making them similar to tree data structures. However groups should be free from such restrictions to simplify usage, but a group exclusively consisting of other groups while supporting set operations on the data selected by the groups could be rather challenging to design and implement, especially if we consider a few different types of groups with incompatible internal data representations. After all, the API only specifies the existence of an abstract group that can verify if a data point is part of it or not and tell the size of the covered data range, which could for example be used to create arbitrary data filters.
- **Python binding updates;** As mentioned in sections 3.3.2 PROGRAMMING LANGUAGE AND SOFTWARE LIBRARIES and 6.3.3 PYTHON BINDINGS, the Python bindings need to be refactored and cleaned up. Both to improve performance and to avoid licensing them completely under GPL by writing separate bindings for each CCDVL module and combining them under a common namespace. Furthermore, the bindings do not currently export all methods and classes, which is another reason to continue working on them.
- **Graph type superclass;** Although the scatter plot and time series graphs are very similar and can easily be combined into one class, it would be more object-oriented to have one common more abstract superclass with the shared functionality, which could also be used when adding additional graph types in the future.
- **Improved iterator design;** As stated before, the memory manager iterator needs a few improvements to handle sparse data, meaning that the memory manager iterator instead of iterating datasets should iterate data points; similar to the group iterator.

6.6.2 Frontend and GUI

- **Frontend design and interface;** Currently the internal design of the frontend is not really satisfactory, and there is no interface for the frontend, as it initially seemed hard not to tie it to a specific GUI library of framework. Bad planning and time limitations made the needed redesigns of the frontend slip by. In hindsight, there is some frontend functionality

that obviously belongs in a general frontend interface, such as getting and setting the tool selection, the saved groups of points and the current state or settings, as well as functionality for updates, stop and reloading, toggling of the grid and general communication with the backend. This would also have made the internal design of the frontend better and less coupled. Utilizing more observers (see section 2.4.1 OBSERVER) and Qt signals and slots could further reduce coupling and circular dependencies.

- **Complete settings dialog;** The settings dialog is currently not fully implemented, especially validators for the input fields are needed. An example of these is the validators needed for the neighbourhood overview box, as seen in FIGURE B.5 in APPENDIX B - MANUAL AND USER GUIDE, containing the graph image tiles size and number of graph image tiles, were the tile size times the number of tiles cannot be less than the respective size of the viewport times three. The apply button should feature the prominent "done" button design pattern (Tidwell, 2011) better by being highlighted by default, and the good defaults, input hint and same-page error messages design patterns (Tidwell, 2011) should also be utilized extensively in the settings dialog. Additionally, settings for specifying the limits controlling when scientific notation should be used for graph axes values could be added.
- **Aligned clipmap grid;** The graph image tile layout of the graph space should be quantized to eliminate the bottom left rounding error, which would improve caching and possibly improve performance of the center on graph coordinate method. Currently, when changing zoom or scale level, a new grid layout is created for the graph image tiles each time based on the current center point of the viewport, making it very unlikely to coincide with graph image tiles already stored in the graph cache. Checking the cache for graph image tiles with the correct properties and then aligning to their grid layout is a possible extension to the existing implementation. Adding additional metadata to the graph cache could also make such an alignment less costly to perform.
- **Prerendering;** It would be nice to be able to prerender nearby graph image tiles on both the current zoom level and on zoom levels one or more steps below and above, allowing the renderer to work in the background, to facilitate smoother transitions. This requires the aligning of graph image tiles mentioned in the previous bullet point, and must of course be limited by the size of the graph cache to avoid potentially filling it with tiles that will not be needed in the near future or at all. An additional invisible border or set of borders, lying inside the pan-trigger border, could be used to detect and predict the direction the user is panning to start pre-rendering of graph image tiles beyond that point in the given direction. This must of course be balanced carefully to not interfere with the user interaction and to avoid constantly running the renderer if there is little gain. An option for enabling, disabling and controlling the degree of prerendering could also be added to the settings dialog.

- **Type-safe coordinate system types;** An interesting idea is to make the type definitions for the different coordinate systems into type-safe, implicitly self-converting coordinate type classes extending the Qt classes. This could have been useful, and would probably have saved time while developing the GUI, even though some time must be spent to override, overload and wrap most of the functions and all operators implemented in the original Qt classes, to make the new type classes work seamlessly with the other Qt classes. Compile time warnings could also have been added when mixing coordinates and values from more than one type, to allow easier debugging and for possible coordinate-related mistakes to be immediately highlighted.
- **Catch coordinate conversion errors;** The Transform2D class returns an outcome of the conversion, stating if it was a success or if an error occurred, indicating if the values stored in the provided array represents valid coordinates or possibly NaN or infinity. Currently, the QtToolGraphicsView ignores these and always assumes that the conversion succeeds. If these are to be checked by the frontend, the checking step must be propagated to all functions using the transformations, and possibly to all functions receiving coordinate points as arguments or return values from other functions, which basically affects every non-trivial function in the frontend. Replacing the points that are used with pointers to points, and using NULL-pointers to represent failed conversions is one possible option. Decisions on how to handle failures must of course also be made in each case, such as deciding to retry or abort the operation, and if and how the user should be informed of this. Overloading the conversion functions in the tool graphics view to take one additional argument stating the number of retries is another possible option.
- **Grid and axes draw functions functors;** The functions responsible for drawing the grid and axes are quite complex and hard to follow with a lot of nested loops, if-statements and conditional if-statements used to keep track of all of the many parameters used. Although the code could be converted into more aesthetic recursive functions without too much work, we decided against this design early on as there are usually many hundreds of iterations with a lot of parameters to go through when drawing the axes, which does not usually sit well with operating system memory usage when doing this recursively. The code should instead be changed to use functors for each axis, which should make the whole thing a lot easier to follow and understand. Reducing the number of parameters would also be helpful, but may be hard in the graph axes case due to the amount needed to keep track of the values of the previous, current and next iteration steps, which are used for formatting, layout and drawing the graph axes. The amount could be visibly reduced and easier to handle by instead moving groups of parameters to structs.
- **Change displayed axes dimensions on-the-go;** The axes unit labels could be replaced with dropdown choosers to allow a quick and easy way of changing what dimensions are displayed along the axes when the dataset

contains multidimensional data. This approach was initially discussed and was supposed to be used, but somehow got lost on the way.

- **Movable GUI panels;** Allowing the user to move, rearrange, resize, show/expand and hide/collapse individual GUI panels to customize the layout to their liking is always a nice feature of any non-trivial GUI. Tidwell (2011) defines this as the movable panels design pattern. The only currently available option for something like this in the graph widget example implementation prototype is that the toolbar can (statically) be set to be either above or below the graph view at widget creation and initiation.
- **Display more process progress information;** A progress bar alternating between the current processes, or a progress bar displaying a mouse-over list of current processes with progress bars, much like one used in NetBeans, are two alternatives. The *TaskProgressInterface* is a remnant of the first of these ideas, which could be developed further. Another related idea is to display a small progress bar, or some other loading indication, on each graph image tile visible in the graph view during progressive updates to better show the user that all data may not be visible as of yet. Absolute positioning of such indicators would be ineffective since graph image tiles should be rather large and few, as shown by chapter 5 RESULT, which means that the user is unlikely to encounter more than one or two of these indicators when panning during a progressive update. Such a progress bar or loading indication would instead probably have to be a dynamic and floating pop-up, much like a tool tip, which moves with the graph view to make sure it stays visible to the user whenever a portion of a graph image tile is seen in the graph view.
- **Improved indication that a rendering process has been stopped;** Letting the reload button pulsate with color and setting displayed graph view values to blank may be a good idea to better indicate the possible mismatch of displayed and internal graph states. This could be further indicated by using the graph image tile progress bars or loading indicators, mentioned in the previous bullet point, to show which tiles are mismatched. Additionally, making sure that the displayed state always reflects the (potentially unfinished) internal state would of course be the best solution.
- **Support navigation with strict graph borders;** Currently there are no limitations on panning (other than reaching negative or positive infinity in C++) and no indications for where the user can (usefully) go or what lies beyond the displayed portion of the graph (see section 2.6.4.1 EXPLORATION AND INSIGHT THROUGH INTERACTION). Identifying the global minimum and maximum for each dimension of the data, preferably only once while initially reading the data, would make it possible to enforce strict graph borders and edges that could not be panned across. Data range searches could also be used in combination with graphical elements to indicate the general direction in which there is more data available outside the displayed portion of the graph to further support navigation.

- **Jump to coordinate tool button;** The functionality for this already exists in the frontend, but users currently have no way of utilizing it through the GUI. We suggest adding a button with an appropriate icon, for example an icon depicting four arrows pointing towards a dot in the center, that brings up a simple modal dialog box with text edit fields for x- and y-coordinates to center on, along with confirmation and cancel buttons. Adding such a tool would also increase the need of an undo or go back functionality, as recommended in section 2.7.3 DESIGN PATTERNS, HEURISTICS AND GUIDELINES.
- **Configurable tool switching hotkeys for expert users**. Initially there were some non-configurable hotkeys, but they were removed to not interfere with application using the library implementation.
- **Grid line style**; for example adding an option for dashed or dotted line style.
- **More advanced group selections;** Currently, the lasso select tool uses an odd-even algorithm provided by Qt to close the path of the lasso tool to create the selection polygon. This means that self-intersections will be excluded from the selection, which could easily be fixed by using the winding number algorithm, making it a true lasso selection by including everything inside the outer borders of the selection. This was however not prioritized, as Combine did not seem to mind the behaviour of the common odd-even algorithm.

6.6.3 New modules

In general, any new graph type other than scatter plot and time series will at least require a new renderer, and possibly also a new or modified GUI. 3D graphs, for example would require both.

- **OpenGL (or other hardware accelerated) renderer**; Requires API changes. See section 6.3.2 MODULES AND COMPONENTS and section 6.6.1 API AND BACKEND for more details.
- **Memory manager with fast search**; Either by finishing and using the originally planned model partially analysed in section 4.2.2.1 MEMORY MANAGER ANALYSIS AND DESIGN, or by using a regular database. Experimenting with the possibilities and limitations of the HDF5 file format would also be interesting, especially since Combine already use it. According to The HDF Group (2011) their file format provides a versatile data model capable of representing very complex data objects and a wide variety of metadata, with no limit on the number or size of data objects in the dataset. They also claim that it has many integrated performance features that allow for access time and storage space optimizations among other benefits, making it a possible candidate to be used in a database based memory manager. Fast search could also improve performance

when using many graph image tiles, as the current renderer would be able to tell which graph image tiles to render (the sorted) data to rather than having to go through all graph image tiles over and over again.

7 Conclusion

The development of the proposed library and API is a very large project, but feasible to complete within a reasonable time frame for two hard-working persons with good planning and good delimitations. Many interesting problems and solutions have been considered and weighed against each other, even if not all of them were dealt with, and many enhancements have thus been suggested in section 6.6 FUTURE WORK, ADDITIONS AND EXTENSIONS.

The CCDVL library itself was semi-successful as it works quite well even if it is not completely finished, and considering the results it is more usable for large datasets than the libraries studied during the prestudy (see section 5.3 COMPARISON WITH PRESTUDY RESULTS). The resulting library may not be faster at rendering than existing libraries, but it allows interaction during loading of data and updates and avoids consuming all system resources while doing so, although it may be a bit jerky depending on the configuration, as seen in TABLE 4 in chapter 5 RESULT.

It is thus very possible to create an interactive graph widget that is capable of handling very large datasets without crashing or making the computer unusable during rendering, even when running on a low-end computer such as the test notebook specified in section 1.4 DELIMITATIONS, as shown with CCDVL. The potential and possible applications of CCDVL are also promising within many areas, as discussed previously in section 6.4 RESULT DISCUSSION.

References

Aaronson, S. et al. (2012) Complexity Zoo. *Qwiki*. http://qwiki.stanford.edu/index.php?title=Complexity_Zoo&oldid=24714 (29 Mar. 2012).

Adobe Systems Incorporated (2012) *Adobe*. http://www.adobe.com/ (4 May 2012).

Andrews, G. R. (1999) *Foundations of Multithreaded, Parallel, and Distributed Programing*. Reading: Addison-Wesley Longman Inc..

Association for Computing Machinery (2012) *ACM Digital Library*. http://dl.acm.org/ (23 Apr. 2012).

Arora, S. and Barak, B. (2009) *Computational Complexity: A modern approach. Cambridge:* Cambridge University Press.

Bertin, J. (1983) *Semiology of graphics*. Madison: University of Wisconsin Press.

Belady, L. (1966) A study of replacement algorithms for virtual storage computers. IBM *Systems Journal*, vol. 5, nr 2, pp. 78-101.

Brath, R. K. (1999) *Effective information visualization: guidelines and metrics for 3D interactive representations of business data. Ottawa:* National Library of Canada.

Chalmers Library (2012) *Chalmers library – Chalmers University of Technology library, Gothenburg, Sweden*. http://www.lib.chalmers.se/ (23 Apr. 2012).

Chazelle, B. (1980) *Computational Geometry and Convexity*. New Haven, Connecticut: Yale University.

Chazelle, B. (1991) Triangulating a Simple Polygon in Linear Time. *Discrete Computational Geometry*, vol 6, nr 1, pp. 485-524.

Collins, W. J. (2005) *Data structures and the Java collection framework.* Second Edition. New York: McGraw-Hill Companies, Inc..

Combine (2012) *Enter the next level*. http://www.combine.se/ (13 Mar. 2012).

Cooper, A., Reimann, R. and Cronin, D. (2007) About Face 3*: The Essentials of Interaction Design.* Indianapolis: Wiley Publishing, Inc..

Dix, A. J. et al. (1998) *Human-computer interaction.* London: Prentice Hall Europé.

Enns, J. T. and Healey, C. G. (2011) Attention and Visual Memory in Visualization and Computer Graphics. IEEE Transactions on Visualization and Computer Graphics. In print.

Faylor, C. et al. (2012) *Cygwin*. http://www.cygwin.com/ (20 July 2012).

ffunction Inc. (2010) *What is Data Visualization*? http://blog.ffctn.com/ (24 May 2012).

Gmane (2004) *vector-agg-general@lists.sourceforge.net Anti-Grain Geometry*. http://blog.gmane.org/gmane.comp.graphics.agg/month=20040801 (8 May. 2012).

Gnuplot homepage (2012) http://www.gnuplot.info/ (11 May. 2012).

Google Inc. (2011) *Google Books*. http://books.google.com/ (23 Apr. 2012).

Google Inc. (2012) *Google Documents*. https://docs.google.com/ (25 Apr. 2012).

Hewett, T. T. et al. (1996) *ACM SIGCHI Curricula for Human-Computer Interaction*. [Electronic] New York: Association for Computing Machinery, Inc..

Hormann, K. and Agathos, A. (2001) *The point in polygon problem for arbitrary polygons. Computational geometry*, vol. 20, nr 3, pp. 131-144.

Huang, C-W. and Sihi, T-Y. (1996) On the complexity of point-in-polygon algorithms. *Computers & Geosciences*, vol. 23, nr 1, pp. 109-118.

Hunter, J. et al. (2011*) matplotlib: python plotting - Matplotlib V1.1.0 documentation.* http://matplotlib.sourceforge.net/ (11 May. 2012).

IEEE (2008) IEEE *Standard for Information Technology - Portable Operating System Interface (POSIX) Base Specifications, Issue 7*. IEEE Std 1003.1-2008 (Revision of IEEE Std 1003.1-2004).

IEEE (2012) *IEEE Xplore Digital Library*. http://ieeexplore.ieee.org/ (23 Apr. 2012).

International Organization for Standardization/International Electrotechnical Commission (2011) *Information technology - Programming languages* - C++. ISO/IEC 14882:2011. International Organization for Standardization, Geneva.

Kleinberg, J. and Tardos, É. (2005) *Algorithm* Design. Boston, Mass: Pearson Education, Inc..

Larsson, A. et al. (2011) *Dia - GNOME Live!*. http://live.gnome.org/Dia (11 May 2012).

Lien, J-M. and Amato, N. M. (2004) *Approximate convex decomposition of polygons. In Proceedings of the twentieth annual symposium on Computational geometry (SCG '04)*; June 8-11 2004, New York. pp. 17-26.

Microsoft Corporation (2012a) *Memory Management Functions*. http://msdn.microsoft.com/ (20 July 2012).

Microsoft Corporation (2012b) *Microsoft Office*. http://office.microsoft.com/ (11 May 2012).

Metsker, S-J. and Wake, W-C. (2006) *Design Patterns in Java*. Second Edition. Wokingham: Addison-Wesley Professional.

Musumeci, G-P. D. and Loukides, M. (2002) *System Performance Tuning*. Second edition. [Electronic] Sebastopol: O'Reilly Media.

National Endowment for the Humanities, Office of Digital Humanities (2012) *Humanities High Performance Computing*. http://www.neh.gov/ODH/ (15 Mar. 2012).

Nielsen, J. (1993) *Usability Engineering.* San Francisco: Morgan Kaufmann.

Nielsen, J. and Mack, R. L. (1994) *Usability Inspection Methods.* New York: John Wiley & Sons, Inc..

Nokia Corporation (2012) Qt - *A cross-platform application and UI framework.* http://qt.nokia.com/ (16 Mar. 2012).

Norman, D. A. (1988) *The Design of Everyday Things*. New York: Doubleday.

O'Rourke, J. (1998) *Computational geometry in C*. Second edition. Cambridge: Cambridge University Press.

Oracle Corporation (2012) *Welcome to NetBeans*. http://netbeans.org/ (25 Apr. 2012).

Papadimitriou, C. H. (1994) *Computational Complexity.* Reading: Addison-Wesley.

PLplot developer community (2008) *PLplot Home Page - Main*. http://plplot.sourceforge.net/ (13 May. 2012).

Preece, J. et al. (1994) *Human-computer interaction*. Wokingham: Addison-Wesley.

Pudov, A. (2011) *Hard drive and RAM disk Comparison Report*. *Audry Pudov.* http://www.andreypudov.com/ (13 Mar. 2012).

Qt Project Hosting (2011) *Threads and QObjects*. http://qt-project.org/ (11 May 2012).

Rathmann, U. and Wilgen, J. (2011) *Qwt User's Guide: Qwt - Qt Widgets for Technical Applications*. http://qwt.sourceforge.net/ (13 May 2012).

Rezakhanlou, F. et al. (2007*) Lecture Notes in Mathematics - Entropy Methods for the Boltzmann Equation: Lectures from a Special Semester at the Centre mile Borel, Institut H. Poincar, Paris, 2001*. [Electronic] Berlin: Springer.

Riverbank Computing Limited (2012) *Riverbank Computing Limited*. http://www.riverbankcomputing.com/ (3 Apr. 2012).

Samet, H. (2006) *Foundations of Multidimensional and Metric Data Structures*. Boston: Morgan Kaufmann.

Shemanarev, M. (2006) *Anti-Grain Geometry*. http://www.antigrain.com/ (8 Apr. 2012).

Silberschatz, A., Galivn, P. B. and Gagne, G. (2008) *Operating System Concepts*. Eighth Edition. Hoboken: John Wiley & Sons, Inc..

Sotirovski, D. (2001) *Heuristics for iterative software development*. IEEE Software, vol. 18, nr 3, pp. 66-73.

Spence, R. (2007) *Information Visualization: Design for Interaction*. Second Edition. Harlow: Pearson Education Limited.

The Code::Blocks Team (2011) *Code::Blocks*. http://www.codeblocks.org/ (25 Apr. 2012).

The GIMP Team (2012) *GIMP - The GNU Image Manipulation Program*. http://www.gimp.org/ (4 May 2012).

The HDF Group (2011) *HDF5*. http://www.hdfgroup.org/ (24 Apr. 2012).

Tidwell, J. (2011) *Designing Interfaces*. Second Edition. Sebastopol: O'Reilly Media.

Tufte, R. E. (2001) *The Visual Display of Quantitative Information*. Second Edition. Cheshire: Graphics Press LLC..

Valgrind Developers (2011) *Valgrind Home*. http://valgrind.org/ (7 May 2012).

Van Heesch, D. (2012) *Doxygen*. http://www.stack.nl/~dimitri/doxygen/ (24 Apr. 2012).

Weinberger, B. et al. (2011) Google C++ Style Guide (Revision 3.199). *google-styleguide – Style guides for Google-originated open-source projects*. http://code.google.com/p/google-styleguide/ (3 Apr. 2012).

Wiseman, Y. and Jiang, S. (2009) *Advanced operating systems and kernel applications; techniques and technologies*. Portland: Book News, Inc.

yWorks (2012) yEd - *Graph Editor*. http://www.yworks.com/ (22 Jul. 2012).

Zverina, J. (2010) What's Next for High-Performance Computing? *University of California, San Diego*. http://www.ucsd.edu/ (15 Mar. 2012).

Appendices

Appendix A – Initial requirements and requests

This appendix contains the initial requirements and requests that emerged during the first couple of meetings with Combine when the vision and goals of the proposed library were also discussed. These are listed roughly in order of importance within each block below, although some requirements were optional. Requirements and requests that fall outside the delimitations, such as for 3D graphs, have been omitted.

A.1 Non-functional requirements

- The library should handle and visualize large amounts of data efficiently.
- The library should minimize memory and I/O usage to maximize performance and responsiveness of the operating system, widget and GUI.
- The graph widget should have a responsive user interface with fast and informative feedback, by having one GUI thread and one or more background threads.
- The graph widget should be highly interactive.
- The library should be usable on most platforms.
- The API should be modular and extendable.

A.2 Functional requirements

- The widget should make single points discernible in the graph plot.
- The widget should use transparent points or color to indicate point density in packed clusters in graph plots.
- The widget should provide a pannable view of the graph plot.
- The library should provide zoom functionality in the graph plot.
- The library should provide a lasso tool for selecting groups of points in the graph plot.
- The widget should be resizable and maintain the GUI layout and functionality on resize.
- The library should provide and be able to toggle a grid and other helper lines in the graph plot.

- The widget should auto adjust graph axes units and text labels to fit the current position, scale and zoom level of the graph plot.
- The widget should display large values in scientific notation on the graph axes.
- The library should support for both linear and logarithmic scales in the graph plot.
- The library should provide Python bindings.
- The widget should have an overview window with zoom box to show an overview plus detail of the graph plot.
- The library should be able to save and manage groups of selected data.
- The widget should be able to toggle the visibility of saved groups in the graph plot.
- The widget should support color coding of groups, points, signals and plotted functions in the graph plot.
- The widget should be able to show both overlapping and separate signals for easy comparison in time series graphs.
- The widget should have one shared (x-)axis for easy comparison in time series graphs.
- The widget should have normalized y-axis in time series graphs.
- The widget should support selectable signals, showing the corresponding unit of the y-axis when selected in time series graphs.
- The widget should show min and max values (“error bars”) based on the given confidence interval around the signal in time series graphs.
- The library should have an abort mechanism to be able to stop rendering if needed, allowing the operating system time to recover if pushed beyond its limits or if a deadlock or infinite loop somehow occurs.
- The library should support hierarchical grouping (optional).
- The library should support configurable colors and toggle color coding in graph plot (optional).
- The widget should be able to toggle lines connecting points to represent the internal order of the data in graph plots on and off (optional).
- The widget should allow the user to add and manage custom helper lines and plotted functions on the fly in the graph plot (optional).

Appendix B - Manual and user guide

The manual and user guide contains images and explanations of how the different parts of the graphical user interface (GUI) and settings dialog work.

B.1 Main widget GUI

The main widget interface consists of the areas outlined and numbered in FIGURE B.1, which are each described in more detail below.

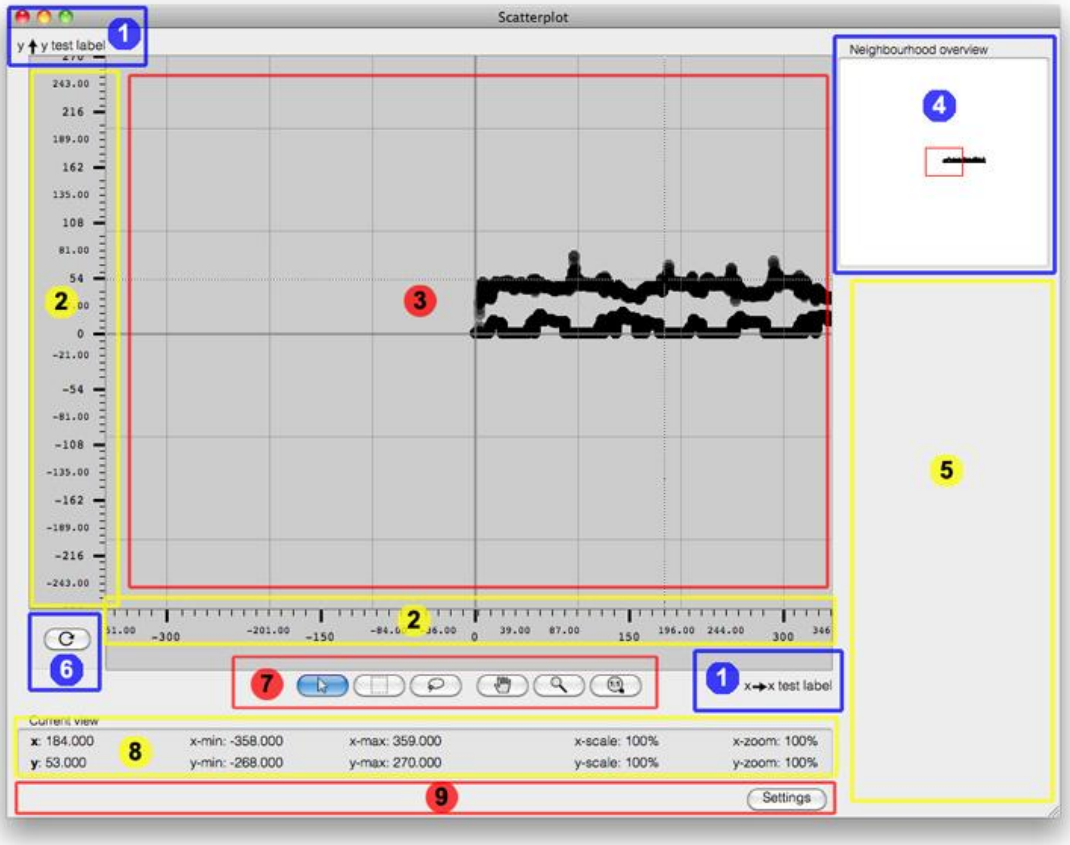


Figure B.1. *The different parts and areas of the GUI have been numbered and outlined as follows: (1) axes variable and unit labels, (2) axes values, (3) plotted graph area with grid, (4) neighbourhood overview, (5) empty group area, (6) stop/reload button, (7) toolbar buttons, (8) current graph view values, and (9) status bar with settings button.*

1. Axes labels containing the variable, a small arrow to indicate the direction of the axes and the unit labels (see FIGURE B.2), specified by the settings.

(a) y-axis labels *(b) x-axis labels*

Figure B.2. *Axes variable and unit labels.*

2. Axes values corresponding to, and changing with, the current view and its changes (see FIGURE B.3). The intervals between the steps and values can

be changed in the settings. If any of the values are too close to zero, too large or too small they will be written in scientific notation divided over two lines instead.

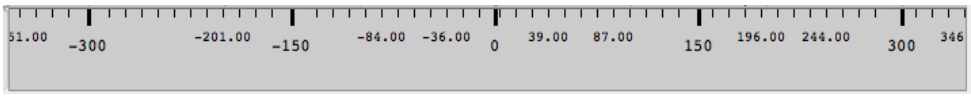


Figure B.3. *Axis values for the x-axis with a step dash interval of 150.*

3. The main graph view area, displaying a portion of the plotted graph (see FIGURE B.4). The tools interact with this area, and there is also a grid that can be toggled and modified in the settings. The origin lines are drawn a bit thicker to make them easier to identify.

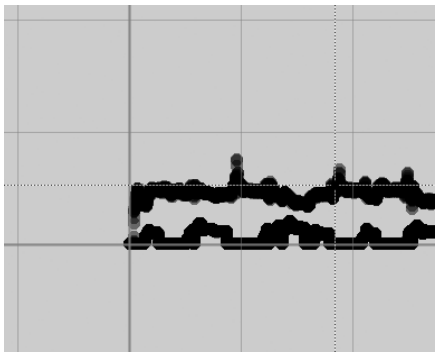


Figure B.4. *Plotted graph area with grid.*

4. The neighbourhood overview shows a scaled overview of the currently loaded pannable portion of the plotted graph, or a loading message if the graph portion is being rendered (see FIGURE B.5). The red outlined box shows the current position of the main graph view area, and the outline box can be used to move or pan the main graph view area by clicking, or clicking and dragging, inside the neighbourhood overview area. To load a new portion of the plotted graph, release the outline box close to an edge, or use the panning tool in the main graph view area. The neighbourhood overview uses a black-and-white mask, ignoring transparency and color, which gives better contrast and a more useful overview.

(a) *loading overview* (b) *showing overview*

Figure B.5. *Neighbourhood overview.*

5. An area reserved for showing and managing saved groups of selected data.

6. A stop/reload button, which can be used to reload the current loaded portion of the plotted graph, or to stop an on-going rendering process that might have stalled (see FIGURE B.6). Stopping a rendering process will most likely leave the main graph view and the loaded portion of the plotted graph incomplete and out of sync. A reload is strongly recommended after each use of the stop button.

(a) *stop* (b) *reload*

Figure B.6. *The two states of the stop/reload button.*

7. The toolbar containing the tools used to interact with the main graph view area.

(a) (b) (c) (d) (e) (f)

Figure B.7. *The toolbar buttons;* (a) *point select,* (b) *rectangle rubber band select,* (c) *lasso select,* (d) *pan,* (e) *zoom and* (f) *reset zoom.*

They are, from left to right as seen in FIGURE B.7 above:

(a) **Point select;** left-click selects a point, right-click deselects. Double-clicking the tool button toggles the mouse cursor between a crosshair and a normal cursor.

(b) **Rectangle rubber band select;** left-click and drag brings up the rubber band selection, right-click deselects. FIGURE B.8 shows an example of a rectangle rubber band selection.

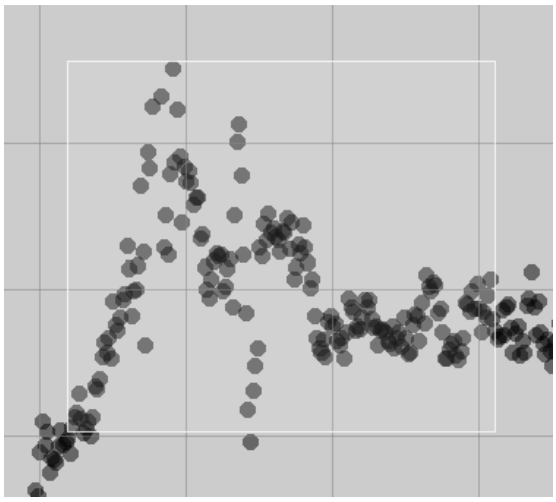


Figure B.8. *An example of a rectangle rubber band selection.*

(c) **Lasso select;** left-click and drag brings up the lasso selection, right-click deselects. Currently, the lasso tool uses and odd-even fill, meaning that crossing the selection with itself will deselect the crossed areas FIGURE B.9 shows an example of a lasso selection.

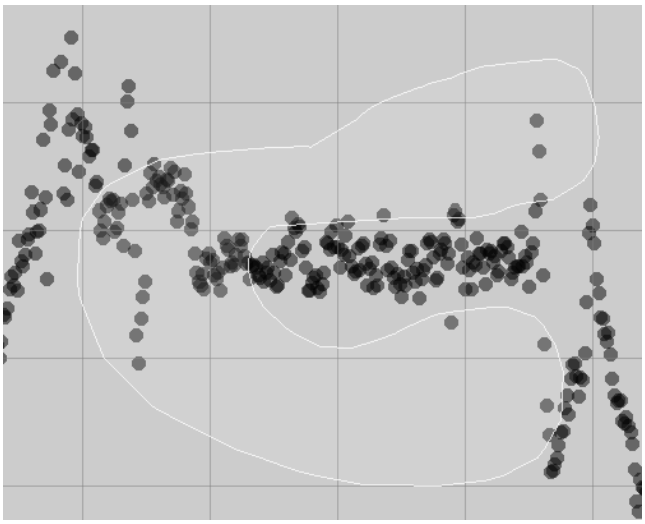


Figure B.9. *An example of a lasso selection.*

(d) **Pan;** left-click and drag to pan the graph view area. Panning close to an edge will cause the displayed portion of the plotted graph to be updated.

(e) **Zoom;** left-click to zoom in, option-left-click to zoom out, centering on the clicked position. Left-click and drag or option-left-click and drag will bring up a selection rubber band and zoom in or out to the selected area. The escape button may be pressed before releasing the mouse button to cancel the selection rubber band zoom action. Using the mouse scroll wheel will zoom in or out depending on the direction of the wheel, keeping the graph view centered on its current center position. Double clicking the zoom tool button resets the zoom level to 100%.

(f) **Reset zoom;** resets the zoom level to 100%, keeping the graph view centered on its current center position.

8. The values and parameters associated with the current graph view area (see FIGURE B.10). These will change and update with graph view area changes. The x- and y-coordinates change with the movements of the cursor when it is positioned over the graph view area or the neighbourhood overview area. If any of the values are too close to zero, too large or too small they will be written in scientific notation divided over two lines instead.

Figure B.10. *Current graph view values from left to right: mouse-over coordinates, minimum coordinate values, maximum coordinate values, scale-factors and zoom-factors.*

9. A status bar displaying status messages and mouse over information about the tools and stop/reload button, as well as a progress bar visible during view updates (see FIGURE B.11). The settings button to the right is always displayed, and opens the settings dialog.

Figure B.11. *Status bar with settings button currently displaying the mouse-over message for the reset zoom tool button, a message and progress bar informing the user that the view is currently updating, and the settings dialog button.*

B.2 Settings dialog

The setting dialog contains three tabs for configuring the graph view, window view and advanced settings for the graph view cache. The graph view tab contains settings for the point and line size, shape and color, as well as background clear color and scale settings (see FIGURE B.12).

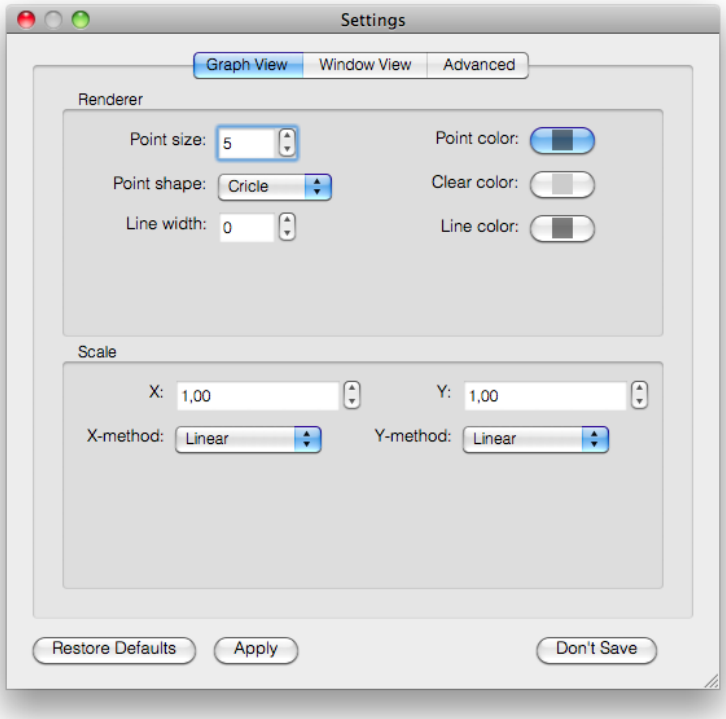


Figure B.12. *The Graph View tab in the Settings dialog, allowing renderer and scale adjustments.*

The window view tab, which is currently disabled, contains settings for the axes unit labels, axes value step size, axes value dash distance and line/text color (see FIGURE B.13). It also contains grid settings for determining the behaviour and color of the grid, as well as settings for controlling the number of graph image tiles and tile size, determining the size and shape of the pannable shown portion of the plotted graph that is shown to scale in the neighbourhood overview.

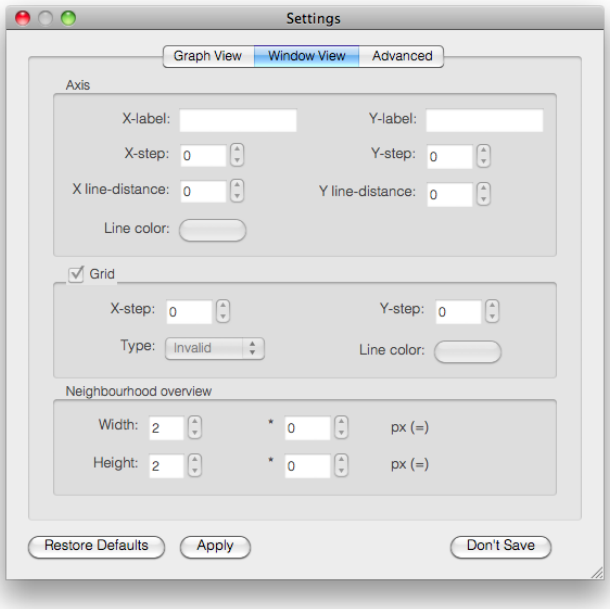


Figure B.13. *The Window View tab in the Settings dialog, allowing axes, grid and neighbourhood overview adjustments.*

Finally, the advanced tab contains settings for deciding how many tiles the graph view cache should contain (see FIGURE B.14). These tiles are used to render the shown portion of the plotted graph, and speeds up rendering when returning to previously visited portions, if the cache is large enough. A larger cache of course uses more memory, and the currently used amount is shown. There is also an (irreversible) option for emptying the cache.

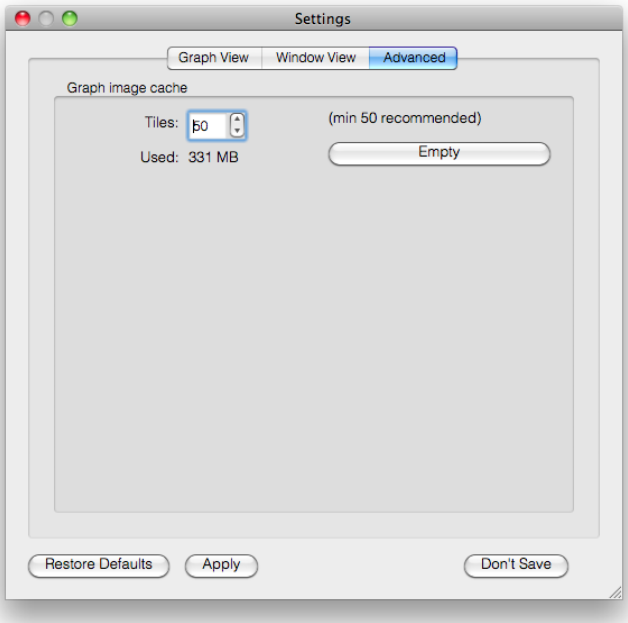


Figure B.14. *The Advanced tab in the Settings dialog, allowing graph image cache adjustments.*

Appendix C - Prestudy results

Three larger Python graph plotting libraries were studied; Matplotlib, Gnuplot and PLplot. TABLE C.1 contains a compressed summary of the results, followed by more detailed analyses and performance tests of each library in the sections below. The GUI interactions were quantified, based on the observations of Cooper (2007), as fast; 100 milliseconds or less, responsive; up to 1 second, slow; 1-10 seconds, and unusable; more than 10 seconds.

Table C.1. *Prestudy summary.*

	Matplotlib	**Gnuplot**	**PLplot**
License	BSD compatible	LGPL	LGPL
Platform support (Windows, Mac OS X, Linux)	All	All	All
Scatter plot and time series graphs	Both	Both	Both
Renderer	Multiple	Multiple	Multiple
Interactive	Yes	Yes	Yes
Memory management	No	No	No
Compresses data	No	No	No
API behaviour	List of axis values	List of axis values	List of axis values
Python	Pure	Bindings	Bindings
Linear scaling and logarithmic scaling	Both	Both	Both
Preprocessing	No	Yes	Yes
True multithreading	No	No	N/A
Can handle large datasets	No	No	No

C.1 Matplotlib

Since Combine currently uses Matplotlib for graph rendering, it is one of the more interesting libraries to examine. Hunter (2011) provides further information on Matplotlib for the interested reader.

- **Which license is used?** Matplotlib is distributed under a BSD compatible license.
- **Which platforms are supported?** Windows, Mac OS X and Linux are supported.
- **Are all graph types supported?** Both scatter plots and time series graphs (with error bars) are supported.
- **Which rendering methods are supported?** The following interactive backends are supported: GTK, GTK-AGG, GTK-Cairo, Qt-AGG, Qt4-AGG, WX, WX-AGG, TK-AGG, FLTK-AGG, Mac-OSX and Cocoa-AGG.
- **Is there any interactive functionality built into the GUI?** At least one of the GUIs from above were capable of zooming, changing scale and saving the graph as an image.
- **Are all data points stored in memory?** Yes.
- **Are data points compressed?** No.
- **How is basic library API usage?** Window creation is straightforward and data is passed to the library as a list of axis values.
- **Are Python bindings provided?** Yes, since it is a native Python library.
- **Which scaling methods are supported?** Both linear and logarithmic scaling methods are supported.
- **Are there any preprocessing capabilities?** None, users are recommended to use third party numeric libraries.
- **Is it Multithreaded?** The library itself is not thread-safe nor is such functionality provided. Although it is possible to render graphs asynchronous with one thread for each figure and canvas using the *figure* API. However, the *cpython* implementation can only use one CPU core, which is suboptimal.

C.1.1 Matplotlib benchmarks

- **GTK;** The rendering results themselves contained graphical artefacts and rendering the scatter plot was notably slower and required much more memory than the time series. Therefore, only half of the generated

test data was used to avoid crashes. The interactive operations were unusably slow and depended on the amount of data visible.

- **GTK-AGG;** The AGG-render hit its complexity limit for line rendering (line smoothing), which again meant that only half of the generated test dataset could be used for the time series and the scatter plot due to the high memory usage. While the rendering results have better quality than plain GTK, the rendering time is slower and again depends on the amount of data visible. As before, the GUI was unresponsive and slow in addition to being unable to render the full test set.
- **GTK-Cairo;** Just like above, in order to obtain any useful test results only half of the generated test data was used. Despite this, the GUI was unusable and the time series graph still failed due to an even lower rendering complexity limit than AGG and the scatter plot was much slower than both plain GTK and GTK-AGG, which is interesting considering that Cairo should use hardware acceleration whenever available.
- **Qt-AGG;** Not tested since it is a legacy Qt support backend using AGG.
- **Qt4-AGG**; Same limitations and similar performance as GTK-AGG and is therefore also unusable
- **TK-AGG;** At this point it seems safe to conclude that Matplotlib uses a different memory model or has a memory problem related to rendering the scatter plot graph in comparison with the time series graph. Again only half of the test data was used for the scatter plot and time series graphs due to AGG limitations. Results are still unusable due to bad responsiveness.
- **WX;** The WX backend is better known as wxWidgets and is also unusably slow, even with only half of the test data, and as with previous Matplotlib results the time series was significantly faster than the scatter plot.
- **WX-AGG;** A rendering backend based on wxWidgets and AGG. Unusable, with performance similar to previous AGG based combinations.
- **FLTK-AGG;** A rendering backend based on Fast Light ToolKit and AGG. Unusable, with performance similar to previous AGG based combinations.
- **Mac-OSX and Cocoa-AGG;** Not tested as neither of the test computers is a Mac.

C.2 Gnuplot

Gnuplot is a powerful free graph rendering library, with more information and documentation found on the official Gnuplot homepage (2012).

- **Which license is used?** Gnuplot is distributed as-is, but the Python bindings are distributed under LGPL.
- **Which platforms are supported?** Windows, Mac OS X and Linux are supported, as well as even very old legacy systems are still supported.
- **Are all graph types supported?** Both scatter plots and time series graphs (with error bars) are supported.
- **Which rendering methods are supported?** The following interactive backends are supported: WXT-Cairo, Qt and X11.
- **Is there any interactive functionality built into the GUI?** Only zooming and a few other interactive operations are supported.
- **Are all data points stored in memory?** Yes.
- **Are data points compressed?** No.
- **How is basic library API usage?** Window creation is straightforward and data is passed to the library as a list of axis values. The API itself is actually a script interpreter using a programing language that describes a graph.
- **Are Python bindings provided?** Provided by a third party.
- **Which scaling methods are supported?** Both linear and logarithmic scaling methods are supported.
- **Are there any preprocessing capabilities?** Various built-in splines, curves and sums to process data
- **Is it Multithreaded?** No, since the WXT-Cairo backend is unresponsive while rendering, however the plot method returns to Python before closing the GUI, suggesting that it spawns a new process separate from the Python interpreter to handle both rendering and user input.

C.2.1 Gnuplot benchmark

- **WXT-Cairo;** A wxWidgets based GUI using the Cairo and Pango libraries to render fonts. The tests were only performed with half of the generated test dataset due to its high memory usage and slowness. It is notable that this backend is pretty fast compared to Matplotlib's backend that used Cairo.

- **X11;** The raw X11 backend had high memory usage, a low quality rendering and almost no interactive functionality. Only half of generated dataset was again used.
- **Qt;** The Qt backend is considered experimental and may not support all features according to the documentation, which is why it was not tested.

C.3 PLplot

PLplot is a cross-platform graph rendering software for drawing scientific graphs that has bindings for a wide variety of programing languages. Visit the PLplot website (PLplot developer community, 2008) for more information

- **Which license is used?** PLplot is distributed under LGPL.
- **Which platforms are supported?** Windows, Mac OS X and Linux are supported.
- **Are all graph types supported?** Both scatter plots and time series graphs (with error bars) are supported.
- **Which rendering methods are supported?** The following interactive backends are supported: Cairo, X11, Qt, wxWidgets and Aqua.
- **Is there any interactive functionality built into the GUI?** The wxWidgets backend has slightly more functionality than the other backends, allowing graphs to be saved and locating values.
- **Are all data points stored in memory?** Yes.
- **Are data points compressed?** No.
- **How is basic library API usage?** Window creation is straightforward and data is passed to the library as a list of axis values.
- **Are Python bindings provided?** Yes, but there could be problems and limited support for all of PLplot's functionality according to the documentation. Successfully installing this library with Python support seems to be difficult. This could depend on the Linux distribution on the test netbook, but the automated build script fails (with multiple problems) to configure the Python bindings.
- **Which scaling methods are supported?** Both linear and logarithmic scaling methods are supported.
- **Are there any preprocessing capabilities?** Interpolation.
- **Is it Multithreaded?** The library i not thread-safe and there is otherwise virtually no information available

C.3.1 Tests

Performance is actually good and responsive after rendering for all backends, and it is likely that PLplot only renders and displays an image representation of a graph.

- **Cairo;** Behaves similar to the Cairo backend of Gnuplot above, but it has a responsive GUI interaction after rendering has finished.

- **X11;** It is interesting that the X11 backend appears to render graphs using progressive updates, although it seems unlikely that this is intentional since the other interactive backends do not support this functionality.

- **Qt;** Slower than the other backends of PLplot.

- **wxWidgets;** Similar behaviour to previously tested WX backends.

- **Aqua;** Not tested as neither of the test computers is a Mac.

Appendix D - Planning report

Data Visualization

With a focus on visualizing large amounts of data
(preliminary title)

Background

To be able to analyse any amount of data it needs to be visualized in some way, such as presenting it in a diagram or a graph. Currently there is no easily available way to visualize large amounts of scientific data on ordinary workstation personal computers, while satisfying response time requirements and graphics quality requirements. In this case large amounts of scientific data is considered to be millions of measurements or reference points stored in files that are in the size range of a couple of gigabytes each. Such files are often too large to be handled by existing plotting software running on workstations, often causing these to crash due to assumptions such that all data will fit in main memory, or seemingly freeze due to lack of responsiveness in the software user interface.

Aim

The aim of this thesis is simplify analysis of large amounts of scientific data by building a small graphics library of highly interactive plot graph widgets using the graphics library OpenGL, the Qt framework or a combination of these.

Delimitations

At the request of Combine, this thesis will only look at 2D-graphs, focusing on scatter plot graphs and time series graphs.

Initial requests and requirements

Shared

- Minimize memory and I/O usage
- Responsive user interface with fast and informative feedback (one GUI thread and one or more background threads)
- Lasso tool for selecting group
- Renaming groups

- Hierarchical grouping
- Toggle groups on or off
- Color coding of groups, points, signals and functions
- Configurable colors
- Toggle color coding in graph
- Zoom functionality
- Toggle grid and help lines
- Add and toggle straight lines and functions manually, such as y=x
- Overview window with zoom box
- Auto adjusting axis units and text labels
- Resizable window
- Support for both linear and logarithmic scales

Scatter plot

- Make single points discernible
- Transparent points or color to indicate point density
- Toggle lines connecting points to represent the internal order of the points

Time series graph

- Overlapping and separable ”uniformly distributed” signals
- One shared (x-)axis for easy comparison
- Normalized y-axis
- Selectable signals, showing the corresponding unit of the y-axis when selected
- Show min and max values based of the confidence interval (Error bars)

Method

A prestudy and comparison of response time and functionality with similar data visualization libraries, mainly matplotlib, for large amounts of data will also be made to be able to compare the efficiency of the result of this thesis. The prestudy is followed by incrementally building a library better suited for such use than currently existing software. In order to obtain good performance memory usage must be limited, data points will be rendered on image textures, efficient interpolation of data must be used, I/O bottlenecks must be identified and either avoided or handled carefully. Separate threads for plotting and handling the user interface will be used to increase responsiveness and minimize unnecessary delays. The library will be coded in C/C++ and called from a Python environment, which means that Python bindings are also required.

Time plan

05/2011 – meetings with Combine

06/2011 – preparations, contact Chalmers and check prerequisites

07/2011 – begin planning report and prestudy, check more details with Combine, get to know Qt

08/2011 – fulfil requirements, register master's thesis at Chalmers and begin working on the thesis, retouch and submit planning report

09/2011 – work on thesis

10/2011 – work on thesis

11/2011 – start writing final report

12/2011 – finish work and final report, present result

Continuous work on the thesis, time plan and final report will be done throughout the thesis.

Appendix E - CCDVL API documentation

This appendix contains the full CCDVL API documentation generated by Doxygen (Van Heesch, 2012), which is over 250 pages long, includes a table of contents, seven chapters with detailed type definition, class and function descriptions, as well as a complete index of these.

CC Data Visualization Library

0

Generated by Doxygen 1.8.1.1

Wed Jul 4 2012 16:36:17

Contents

Chapter 1

CC Data Visualization Library (CCDVL)

1.1 Introduction

The CC Data Visualization Library is a flexible library that aims for efficient graph redering. It is divided into three primary modules each which can be found under its own namespace.

The three modules are memorymanagers, renderers and frontends. Memorymanagers handle data passed to this library for rendering. Renderes render selected data onto image buffers for viewing. Frontends binds everything thogether by accepting a renderer, a memorymanager and finally accepting user interaction for easy usage.

1.2 Compiling

Provided that all dependencies are met; run the following commands to compile:

cmake .

make

1.2.1 Dependencies

Required:

ccdvl::CacheController uses POSIX pthreads.

CMake or equivalent building tools.

Additionally at least one memory manager; renderer and frontend is needed.

Optional:

ccdvl::frontend::QtGraphWidget require Qt4.7 or newer.

ccdvl::renderer::AggRenderer uses the included AGG 2.4 (Anti-grain geometry) source code; which is lincensed under a BSD lincese.

ccdvl::memorymanager::SequentialMemoryManager uses POSIX pthreads; POSIX mmap and friends for manual page management.

Python bindings require Python, obviously.

1.3 Documentation

Regenerating documentation with Doxygen; provided that Doxygen is installed run the following commands:

cmake .

make doc

1.4 Library Usage

To use this library one instance of each module must be created, this is achived by first instantiate an appropriate memory manager. Then running Init(), if present. Continue with a renderer and then a frontend.

Adding data for rendering can be done immediately after the memory manager has been instantiated and it should be possible to do from a different thread while the user interface is running as the primary one. This ensures that the GUI created by the frontend module is responsive even while loading massive amounts of data.

Usage examples can be found under src/qt_test∗ and python/examples/test∗.py for now.

Chapter 2

Todo List

Class ccdvl::AbstractDataSet

Possibly; move inlined code to a CC file.

Member ccdvl::CacheController::Remove (const GraphTileState ∗state)

Perhaps this method should return a boolean indicating if the state was destoyed or not.

Class ccdvl::frontend::QtGraphViewFrame

TODO(Max): this should be split into several classes and helper classes.

Class ccdvl::GraphTileState

3D require rotation of view. (quaternions)

Class ccdvl::Group2D

Implement proper set operations for groups.

Member ccdvl::MemoryManager::AddData (const AbstractDataSet ∗∗data)=0

Change parameter to std::vector<const AbstractDataSet∗>.

Member ccdvl::memorymanager::SequentialMemoryManager::mapped_space_

It is tempting to define SequentialMemoryManagerIterator a friend class, Alternatively provide the needed functionality as methods which is already true for page control methods.

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 4

Class Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

Namespace Documentation

6.1 ccdvl Namespace Reference

Namespaces

- namespace frontend
- namespace memorymanager
- namespace renderer

Classes

- class AbstractGroup

  Generalization of hierarchical group selections.
- class CacheController

  A cache for graph image tiles.
- class CacheObserverInterface

  Observer for CacheController changes of active graph.
- class AbstractDataSet

  Abstract dataset which can hold data of almost any numeric type.
- class DataSetVisitorInterface

  Visitor interface for visitors accessing subtypes of AbstractDataSet.
- class TypedDataSet

  AbstractDataSet container.
- class GraphSceneState

  A class that keeps track of a two dimensional graph scene state, i.e. the values associated with a whole graph image composed of graph image tiles.
- class GraphState

  A class that keeps track of a graph scene state, i.e. the values associated with a whole graph image composed of graph image tiles.
- class GraphTile

  Structure which composes a rendered graph tile.
- class GraphTileState

  Holds configuration for rendering a graph tile.
- class Group2D

  A 2D group selection of graph space; or simply a polygon.
- class GroupSelectionIterator

  Generic group selection iterator which fetches points selected by a group.
- class List2D

A two dimensional standard C++ list template.

- class MemoryManager

  Abstract class for arbitrary memory managers.
- class MemoryManagerIteratorInterface

  A simple iterator interface for iterating various memory managers.
- class MemoryManagerIterator

  Wrapper class used to create std::iterator like objects.
- class MemoryManagerObserverInterface

  Observer for MemoryManager dataset update events.
- class Renderer

  Abstract class for renderers used to render a graph to image slices.
- class RendererConfig

  A class that holds group specific rendering configuration.
- class AsynchronousResource

  Synchronization wrapper class which allows multiple readers.
- class MessageQueue

  Simple message passing queue.
- class ReadersWriterLock

  Synchronization wrapper class which allows multiple readers.
- class TaskProgressInterface

  A simple interface that provides a function for sending information to a progress bar or similar type of progress display.
- class Transform2D

  Simple point transformation class for two dimensional data.
- class PyGroupSelectionIterator

  Iterator used to wrap GroupSelectionIterator and provide Python like iterator methods.
- class PyMemoryManagerIterator

  Iterator used to wrap MemoryManagerIterator and provide Python like iterator methods.

Typedefs

- typedef std::map< const GraphTileState *, GraphTile *, GraphTileState::functor_compare > ::iterator CacheIterator

  An iterator type to iterate cached tiles.
- typedef double GraphDouble

  Floating point type used internally for graph coordinates.
- typedef std::pair< GraphDouble, GraphDouble > GraphPair

  A pair of graph coordinates representing a double precision point.
- typedef std::vector< GraphPair > GraphPairVector

  A series of graph points represented as a vector of pairs.

6.1.1 Detailed Description

The CCDVL namespace, everything related to this library can be found underneath this name.

6.2 ccdvl::frontend Namespace Reference

Classes

- class QtBaseTool

  A base class for tools providing basic functionality and event handling functions to be overridden by subclasses.
- class QtCoordinateAndAxesInfoFrame

  The panel containing information and controls for displaying the x/y-coordinate for mouse over and ranges in the graph.
- class QtGraphImageTile

  A class representing a graph image tile; a `QPixmap` *and a flag that keeps track of if the graph image tile has been drawn onto the graph image.*
- class QtGraphNeighbourhoodFrame

  Shows a small overview image of the neighbourhood area of the current graph view with a rectangle outlining the current view.
- class QtGraphSettings

  Holds the configurable settings related to the Qt frontend, with public access to the setting member variables, rather than using mutators, for convenience.
- class QtGraphViewFrame

  A frame with the rendered plot graphics and axes.
- class QtGraphWidget

  The main widget for the graph GUI, responsible for creating, displaying and updating the GUI components.
- class QtLassoSelectTool

  Provides a reusable lasso select rubber band for the graphics scene of the given QtToolGraphicsView.
- class QtPanTool

  A small reusable pan tool class that pans in a QtToolGraphicsView.
- class QtPointSelectTool

  A reusable point selection tool class that selects "single points" in a QtToolGraphicsView.
- class QtRectangleSelectTool

  A rectangle selection rubberband that keeps track of the start position to make it simpler to set new geometry.
- class QtSettingsDialog

  Graph view settings configuration dialog.
- class QtStatusBarFrame

  A status bar displaying mouse-over information, the progress bar and the settings button.
- class QtToolGraphicsView

  A graphics view that handles mouse button, wheel and key events according to the currently selected tool in the toolbar frame.
- class QtToolbarFrame

  The toolbar, containing the tools and tool buttons.
- class QtZoomTool

  A small reusable zoom tool class that zooms in or out in a QtToolGraphicsView.

Typedefs

- typedef int SceneInt

  Scene coordinate integer.
- typedef int ViewInt

  View coordinate integer.
- typedef int GraphInt

  Graph coordinate integer.
- typedef double SceneDouble

Scene coordinate floating point value.

- typedef double ViewDouble

  View coordinate floating point value.
- typedef QPoint ScenePoint

  Scene coordinate (integer) point.
- typedef QPoint ViewPoint

  View coordinate (integer) point.
- typedef QPoint GraphPoint

  Graph coordinate (integer) point.
- typedef QPointF ScenePointF

  Scene coordinate (floating point) point.
- typedef QPointF ViewPointF

  View coordinate (floating point) point.
- typedef QPointF GraphPointF

  Graph coordinate (integer) point.
- typedef QPolygon ScenePolygon

  Scene coordinate (integer) polygon.
- typedef QPolygon ViewPolygon

  View coordinate (integer) polygon.
- typedef QPolygon GraphPolygon

  Graph coordinate (integer) polygon.
- typedef QPolygonF ScenePolygonF

  Scene coordinate (floating point) polygon.
- typedef QPolygonF ViewPolygonF

  View coordinate (floating point) polygon.
- typedef QPolygonF GraphPolygonF

  Graph coordinate (floating point) polygon.
- typedef QRect SceneRect

  Scene coordinate (integer) rectangle.
- typedef QRect ViewRect

  View coordinate (integer) rectangle.
- typedef QRect GraphRect

  Graph coordinate (integer) rectangle.
- typedef QRectF SceneRectF

  Scene coordinate (floating point) rectangle.
- typedef QRectF ViewRectF

  View coordinate (floating point) rectangle.
- typedef QRectF GraphRectF

  Graph coordinate (floating point) rectangle.
- typedef QSize SceneSize

  Scene coordinate (integer) size object.
- typedef QSize ViewSize

  View coordinate (integer) size object.
- typedef QSize GraphSize

  Graph coordinate (integer) size object.
- typedef QSizeF SceneSizeF

  Scene coordinate (floating point) size object.
- typedef QSizeF ViewSizeF

  View coordinate (floating point) size object.
- typedef QSizeF GraphSizeF

  Graph coordinate (floating point) size object.

- typedef std::list
  < QtGraphImageTile > GraphImageTileList

  A standard list of graph image tiles.
- typedef List2D< QtGraphImageTile > GraphImageTile2DList

  A two-dimensional list of graph image tiles.

6.2.1 Detailed Description

Namespace collection of GUI frontends and related components.

6.2.2 Typedef Documentation

6.2.2.1 typedef int ccdvl::frontend::GraphInt

Graph coordinate integer.

See also

ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygon, GraphPolygonF, GraphRect, GraphRectF, GraphSize and GraphSizeF.

6.2.2.2 typedef QPoint ccdvl::frontend::GraphPoint

Graph coordinate (integer) point.

See also

GraphInt, ccdvl::GraphDouble, GraphPointF, GraphPolygon, GraphPolygonF, GraphRect, GraphRectF, GraphSize and GraphSizeF.

6.2.2.3 typedef QPointF ccdvl::frontend::GraphPointF

Graph coordinate (integer) point.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPolygon, GraphPolygonF, GraphRect, GraphRectF, GraphSize and GraphSizeF.

6.2.2.4 typedef QPolygon ccdvl::frontend::GraphPolygon

Graph coordinate (integer) polygon.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygonF, GraphRect, GraphRectF, GraphSize and GraphSizeF.

6.2.2.5 typedef QPolygonF ccdvl::frontend::GraphPolygonF

Graph coordinate (floating point) polygon.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygon, GraphRect, GraphRectF, GraphSize and GraphSizeF.

6.2.2.6 typedef QRect ccdvl::frontend::GraphRect

Graph coordinate (integer) rectangle.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygon, GraphPolygonF, GraphRectF, GraphSize and GraphSizeF.

6.2.2.7 typedef QRectF ccdvl::frontend::GraphRectF

Graph coordinate (floating point) rectangle.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygon, GraphPolygonF, GraphRect, GraphSize and GraphSizeF.

6.2.2.8 typedef QSize ccdvl::frontend::GraphSize

Graph coordinate (integer) size object.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygon, GraphPolygonF, GraphRect, GraphRectF and GraphSizeF.

6.2.2.9 typedef QSizeF ccdvl::frontend::GraphSizeF

Graph coordinate (floating point) size object.

See also

GraphInt, ccdvl::GraphDouble, GraphPoint, GraphPointF, GraphPolygon, GraphPolygonF, GraphRect, GraphRectF and GraphSize.

6.2.2.10 typedef double ccdvl::frontend::SceneDouble

Scene coordinate floating point value.

See also

SceneInt, ScenePoint, ScenePointF, ScenePolygon, ScenePolygonF, SceneRect, SceneRectF, SceneSize and SceneSizeF.

6.2.2.11 typedef int ccdvl::frontend::SceneInt

Scene coordinate integer.

See also

SceneDouble, ScenePoint, ScenePointF, ScenePolygon, ScenePolygonF, SceneRect, SceneRectF, SceneSize and SceneSizeF.

6.2.2.12 typedef QPoint ccdvl::frontend::ScenePoint

Scene coordinate (integer) point.

See also

SceneInt, SceneDouble, ScenePointF, ScenePolygon, ScenePolygonF, SceneRect, SceneRectF, SceneSize and SceneSizeF.

6.2.2.13 typedef QPointF ccdvl::frontend::ScenePointF

Scene coordinate (floating point) point.

See also

SceneInt, SceneDouble, ScenePoint, ScenePolygon, ScenePolygonF, SceneRect, SceneRectF, SceneSize and SceneSizeF.

6.2.2.14 typedef QPolygon ccdvl::frontend::ScenePolygon

Scene coordinate (integer) polygon.

See also

SceneInt, SceneDouble, ScenePoint, ScenePointF, ScenePolygonF, SceneRect, SceneRectF, SceneSize and SceneSizeF.

6.2.2.15 typedef QPolygonF ccdvl::frontend::ScenePolygonF

Scene coordinate (floating point) polygon.

See also

SceneInt, SceneDouble, ScenePoint, ScenePointF, ScenePolygon, SceneRect, SceneRectF, SceneSize and SceneSizeF.

6.2.2.16 typedef QRect ccdvl::frontend::SceneRect

Scene coordinate (integer) rectangle.

See also

SceneInt, SceneDouble, ScenePoint, ScenePointF, ScenePolygon, ScenePolygonF, SceneRectF, SceneSize and SceneSizeF.

6.2.2.17 typedef QRectF ccdvl::frontend::SceneRectF

Scene coordinate (floating point) rectangle.

See also

SceneInt, SceneDouble, ScenePoint, ScenePointF, ScenePolygon, ScenePolygonF, SceneRect, SceneSize and SceneSizeF.

6.2.2.18 typedef QSize ccdvl::frontend::SceneSize

Scene coordinate (integer) size object.

See also

SceneInt, SceneDouble, ScenePoint, ScenePointF, ScenePolygon, ScenePolygonF, SceneRect, SceneRectF and SceneSizeF.

6.2.2.19 typedef QSizeF ccdvl::frontend::SceneSizeF

Scene coordinate (floating point) size object.

See also

SceneInt, SceneDouble, ScenePoint, ScenePointF, ScenePolygon, ScenePolygonF, SceneRect, SceneRectF and SceneSize.

6.2.2.20 typedef double ccdvl::frontend::ViewDouble

View coordinate floating point value.

See also

ViewInt, ViewPoint, ViewPointF, ViewPolygon, ViewPolygonF, ViewRect, ViewRectF, ViewSize and ViewSizeF.

6.2.2.21 typedef int ccdvl::frontend::ViewInt

View coordinate integer.

See also

ViewDouble, ViewPoint, ViewPointF, ViewPolygon, ViewPolygonF, ViewRect, ViewRectF, ViewSize and ViewSizeF.

6.2.2.22 typedef QPoint ccdvl::frontend::ViewPoint

View coordinate (integer) point.

See also

ViewInt, ViewDouble, ViewPointF, ViewPolygon, ViewPolygonF, ViewRect, ViewRectF, ViewSize and ViewSizeF.

6.2.2.23 typedef QPointF ccdvl::frontend::ViewPointF

View coordinate (floating point) point.

See also

ViewInt, ViewDouble, ViewPoint, ViewPolygon, ViewPolygonF, ViewRect, ViewRectF, ViewSize and ViewSizeF.

6.2.2.24 typedef QPolygon ccdvl::frontend::ViewPolygon

View coordinate (integer) polygon.

See also

ViewInt, ViewDouble, ViewPoint, ViewPointF, ViewPolygonF, ViewRect, ViewRectF, ViewSize and ViewSizeF.

6.2.2.25 typedef QPolygonF ccdvl::frontend::ViewPolygonF

View coordinate (floating point) polygon.

See also

ViewInt, ViewDouble, ViewPoint, ViewPointF, ViewPolygon, ViewRect, ViewRectF, ViewSize and ViewSizeF.

6.2.2.26 typedef QRect ccdvl::frontend::ViewRect

View coordinate (integer) rectangle.

See also

ViewInt, ViewDouble, ViewPoint, ViewPointF, ViewPolygon, ViewPolygonF, ViewRectF, ViewSize and ViewSizeF.

6.2.2.27 typedef QRectF ccdvl::frontend::ViewRectF

View coordinate (floating point) rectangle.

See also

ViewInt, ViewDouble, ViewPoint, ViewPointF, ViewPolygon, ViewPolygonF, ViewRect, ViewSize and ViewSizeF.

6.2.2.28 typedef QSize ccdvl::frontend::ViewSize

View coordinate (integer) size object.

See also

ViewInt, ViewDouble, ViewPoint, ViewPointF, ViewPolygon, ViewPolygonF, ViewRect, ViewRectF and ViewSizeF.

6.2.2.29 typedef QSizeF ccdvl::frontend::ViewSizeF

View coordinate (floating point) size object.

See also

ViewInt, ViewDouble, ViewPoint, ViewPointF, ViewPolygon, ViewPolygonF, ViewRect, ViewRectF and ViewSize.

6.3 ccdvl::memorymanager Namespace Reference

Classes

- class CloneDataSet

  AbstractDataSet clone allocator for SequentialMemoryManager.
- class SequentialMemoryManager

  A regular memory manager which stores data to be accessed in sequence.
- class SequentialMemoryManagerIterator

  An iterator used to iterate a SequentialMemoryManager.
- class StubMemoryManager

  A MemoryManager which do not support add or clear.
- class StubMemoryManagerIterator

  An iterator used it iterate data in a StubMemoryManager.

6.3.1 Detailed Description

Namespace collection of memory manager backends.

6.4 ccdvl::renderer Namespace Reference

Classes

- class AggRenderer

  A simple 2D software renderer using Anti-grain Geometry (AGG).

Typedefs

- typedef std::map< const AbstractGroup *, uint8_t * > ::iterator GroupImageMapIterator

  Group image iterator type.

6.4.1 Detailed Description

Namespace collection of renderer backends.

Chapter 7

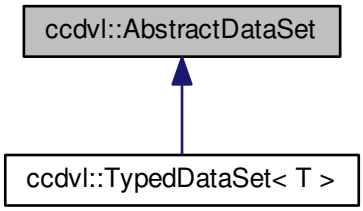
Class Documentation

7.1 ccdvl::AbstractDataSet Class Reference

Abstract dataset which can hold data of almost any numeric type.

`#include <abstract_data_set.h>`

Inheritance diagram for ccdvl::AbstractDataSet:



Public Member Functions

- AbstractDataSet (size_t size)
  Initialize constants for subclasses.
- virtual ~AbstractDataSet ()
  Destroys the object and frees any allocated resources.
- virtual void Accept (DataSetVisitorInterface *v)=0
  Visitor accept method.
- virtual double GetValue (size_t index) const =0
  Fetches a value as a double floating point.
- virtual size_t GetTypeSize () const =0
  Get the type size in bytes.
- size_t GetCount () const
  Get the number of elements in this set.
- size_t GetDataSize () const
  Get the size of all elements.

Protected Attributes

- const size_t size_

  Number of elements in set.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (AbstractDataSet)

7.1.1 Detailed Description

Abstract dataset which can hold data of almost any numeric type.

Todo Possibly; move inlined code to a CC file.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 ccdvl::AbstractDataSet::AbstractDataSet (size_t *size*) [inline]

Initialize constants for subclasses.

Parameters

size	The number of elements in a this dataset.

7.1.3 Member Function Documentation

7.1.3.1 virtual void ccdvl::AbstractDataSet::Accept (DataSetVisitorInterface * *v*) [pure virtual]

Visitor accept method.

Parameters

in	*v*	Visitor to use.

Implemented in ccdvl::TypedDataSet< T >.

7.1.3.2 size_t ccdvl::AbstractDataSet::GetCount ( ) const [inline]

Get the number of elements in this set.

Returns

Number of elements.

7.1.3.3 size_t ccdvl::AbstractDataSet::GetDataSize ( ) const [inline]

Get the size of all elements.

Returns

Size of all elements in set.

Note

This doesn't tell how much space is allocated for this dataset, only how much it uses. However the memory manager used should know how much space was allocated for this dataset.

7.1.3.4 virtual size_t ccdvl::AbstractDataSet::GetTypeSize ( ) const [pure virtual]

Get the type size in bytes.

Returns

Number of bytes.

Implemented in ccdvl::TypedDataSet< T >.

7.1.3.5 virtual double ccdvl::AbstractDataSet::GetValue (size_t *index*) const [pure virtual]

Fetches a value as a double floating point.

Parameters

index	Index of the value to get.

Returns

The value at the provided index as a double floating point.

Implemented in ccdvl::TypedDataSet< T >.

The documentation for this class was generated from the following file:

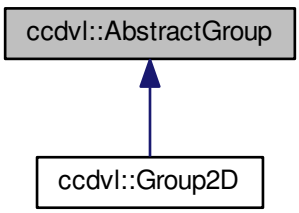
- include/data_set/abstract_data_set.h

7.2 ccdvl::AbstractGroup Class Reference

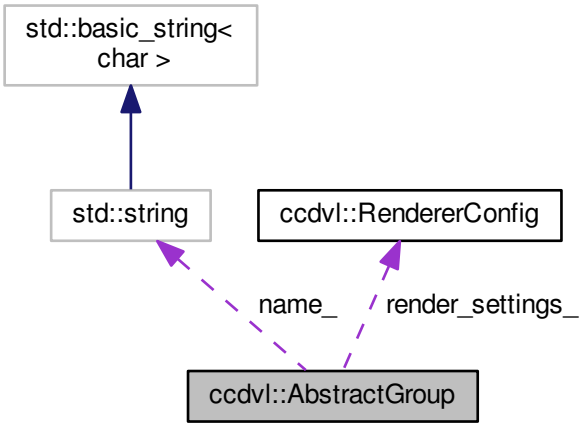
Generalization of hierarchical group selections.

`#include <abstract_group.h>`

Inheritance diagram for ccdvl::AbstractGroup:



Collaboration diagram for ccdvl::AbstractGroup:



Public Types

- typedef std::vector< GraphDouble > GraphPoint

  Datatype representing a single data point.

Public Member Functions

- AbstractGroup (int8_t dimensions)

  Initialize constants for subclasses.
- virtual ~AbstractGroup ()

  Destroys the object and frees any allocated resources.
- virtual bool PointInGroup (const GraphPoint &point) const =0

  Test if a point is within/selected by this group.
- virtual void GetBoundingBox (double *box) const =0

  Compute selection polygon bounding box.
- virtual const std::list< const AbstractGroup * > & GetLeafs () const =0

  Get the list of leaf group nodes.
- GroupSelectionIterator begin (MemoryManager *memorymanager)

  Get an iterator returning points selected by this group from a specific memory manager.
- GroupSelectionIterator end (MemoryManager *memorymanager)

  Get an iterator pointing past the end of points selected by this group from a specific memory manager.

Public Attributes

- RendererConfig render_settings_

  Specific render settings for points in this group.
- bool show_

  Display this group.

- std::string name_

  Group name.
- const int8_t dimensions_

  The number of dimensions.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (AbstractGroup)

7.2.1 Detailed Description

Generalization of hierarchical group selections.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 ccdvl::AbstractGroup::AbstractGroup (int8_t *dimensions*) [explicit]

Initialize constants for subclasses.

Parameters

dimensions	The number of dimensions. Must be positive and larger than zero.

7.2.3 Member Function Documentation

7.2.3.1 GroupSelectionIterator ccdvl::AbstractGroup::begin (MemoryManager * *memorymanager*)

Get an iterator returning points selected by this group from a specific memory manager.

Parameters

in	*memorymanager*	Memory manager containing interesting data.

Returns

Memory manager iterator for this selection.

7.2.3.2 GroupSelectionIterator ccdvl::AbstractGroup::end (MemoryManager * *memorymanager*)

Get an iterator pointing past the end of points selected by this group from a specific memory manager.

Parameters

in	*memorymanager*	Memory manager containing interesting data.

Returns

Memory manager iterator past the end of points.

7.2.3.3 virtual void ccdvl::AbstractGroup::GetBoundingBox (double * *box*) const [pure virtual]

Compute selection polygon bounding box.

Parameters

out	*box*	The polygon bounding box. Array pointer given as x_start, y_start, ..., k_start, x_end, y_end, ..., k_end So its length is equal to dimensions_ times two.

Implemented in ccdvl::Group2D.

7.2.3.4 virtual const std::list<const AbstractGroup*>& ccdvl::AbstractGroup::GetLeafs ( ) const [pure virtual]

Get the list of leaf group nodes.

Returns

The std::list of leafs.

Implemented in ccdvl::Group2D.

7.2.3.5 virtual bool ccdvl::AbstractGroup::PointInGroup (const GraphPoint & *point*) const [pure virtual]

Test if a point is within/selected by this group.

Parameters

in	*point*	A point of dimensions_ dimensions to test.

Returns

True iff the point is within/selected by this group.

Implemented in ccdvl::Group2D.

7.2.4 Member Data Documentation

7.2.4.1 bool ccdvl::AbstractGroup::show_

Display this group.

This must be respected by the frontend when displaying the graph.

Note

Any renderer will still render the image buffer for this group. regardless of this setting.

The documentation for this class was generated from the following files:

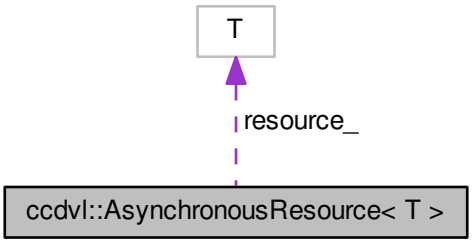
- include/abstract_group.h
- src/abstract_group.cc

7.3 ccdvl::AsynchronousResource< T > Class Template Reference

Synchronization wrapper class which allows multiple readers.

`#include <asynchronous_resource.h>`

Collaboration diagram for ccdvl::AsynchronousResource< T >:



Public Member Functions

- AsynchronousResource (T resource)

  Initializes a new asynchronous resource.
- virtual ~AsynchronousResource ()

  Destroys the object and frees any allocated resources.
- const T & GetResource ()

  Get wrapped resource for read access.
- T & AtomicWriteLock ()

  Obtain exclusive write for wrapped resource.
- void AtomicWriteUnlock ()

  Releases exclusive write access to wrapped resource previously obtained with AtomicWriteLock().

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (AsynchronousResource)

Static Private Member Functions

- static void CancellationHandler (void *mutex)

  Cancellation handler.

Private Attributes

- T resource_

  Protected resource.
- int32_t writer_count_

  Number of writers waiting for, or accessing resource.
- pthread_mutex_t resource_mutex_

  Resource mutex.
- pthread_cond_t writers_waiting_cond_

  Writers wait on this condition.

7.3.1 Detailed Description

template<typename T>class ccdvl::AsynchronousResource< T >

Synchronization wrapper class which allows multiple readers.

This class wraps a resource and allows free read access while only one writer access the resource.

Template Parameters

T	The type to wrap.

Note

It is strongly advised to avoid any kind of complex datatype which may change structure on write as this will be very confusing to maintain. Rule of thumb, POD types only. Arrays are ok while lists are not.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 template<typename T > ccdvl::AsynchronousResource< T >::AsynchronousResource (T *resource*) [inline],[explicit]

Initializes a new asynchronous resource.

Parameters

resource	The resource to wrap.

7.3.3 Member Function Documentation

7.3.3.1 template<typename T > T& ccdvl::AsynchronousResource< T >::AtomicWriteLock ( ) [inline]

Obtain exclusive write for wrapped resource.

This guarantee that no other writer uses the wrapped resource. Blocks until exclusive write access is obtained. This method is also a cancellation point.

Returns

The wrapped resource.

7.3.3.2 template<typename T > static void ccdvl::AsynchronousResource< T >::CancellationHandler (void * *mutex*) [inline],[static],[private]

Cancellation handler.

Unlocks mutex.

Parameters

in	*mutex*	Mutex to realese on cancel.

7.3.3.3 template<typename T > const T& ccdvl::AsynchronousResource< T >::GetResource ( ) [inline]

Get wrapped resource for read access.

Returns

Wrapped resource.

The documentation for this class was generated from the following file:

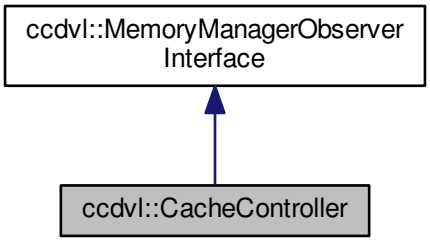
- include/synchronization/asynchronous_resource.h

7.4 ccdvl::CacheController Class Reference

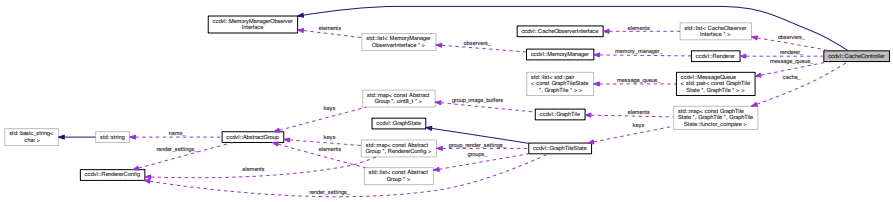
A cache for graph image tiles.

`#include <cache_controller.h>`

Inheritance diagram for ccdvl::CacheController:



Collaboration diagram for ccdvl::CacheController:



Public Types

- enum CacheState { kInvalid = -1, kNotCached, kInProgress, kCached }

  Enumeration of tile cache statuses.

Public Member Functions

- CacheController (Renderer *renderer, int32_t cache_size)

  Cache constructor.
- ~CacheController ()

  Destroys the object and frees any allocated resources.
- virtual void MemoryManagerUpdate (MemoryManager &callee, const AbstractDataSet **new_data)

Update callback when more data is added.

- virtual void MemoryManagerCleared (MemoryManager &callee)

  Update callback when a MemoryManager is cleared.

- void AddObserver (CacheObserverInterface *observer)

  Adds an observer for changes to the active graph.

- void SetMaxCacheSize (int32_t cache_size)

  Sets the number of tiles to cache.

- int32_t GetMaxCacheSize ()

  Get the number of tiles to cache.

- size_t GetMemoryUsage ()

  Get memory used by cached objects.

- bool StopRenderer ()

  Stop renderer.

- void Clear ()

  Attepts to remove all graph tiles.

- CacheState GraphTileStatus (const GraphTileState *state)

  Check status of graph tile state.

- GraphTile * GetGraphTile (const GraphTileState *state)

  Request a tile from graph tile state.

Public Attributes

- const int8_t dimensions_

  Number of dimensions supported.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (CacheController)
- void Enter ()

  Internal entry method for new threads.

- void NotifyObservers (CacheObserverInterface::CacheEvent reason)

  Notifies all observers.

- void FlushCache (size_t count)

  Removes a number of cached states, starting with the oldest.

- void Remove (const GraphTileState *state)

  Attempt to remove and free a single graph tile state.

- void RemoveAll (bool unsafe)

  Removes all cached states.

Static Private Member Functions

- static void * CacheControllerEnter (void *cachecontroller)

  Internal start method for new threads.

Private Attributes

- Renderer * renderer_

  Renderer used to render graph image slices.
- int32_t cache_size_

  Maximum number of cached graph states.
- bool renderer_canceled_

  True iff rendering is cancelled.
- pthread_t rendering_thread_

  Rendering thread.
- pthread_mutex_t cache_mutex_

  Cache mutex.
- std::map< const GraphTileState *, GraphTile *, GraphTileState::functor_compare > cache_

  Tile cache.
- MessageQueue< std::pair< const GraphTileState *, GraphTile * > > message_queue_

  Message queue of tiles to render.
- std::list < CacheObserverInterface * > observers_

  List of registered observers.

7.4.1 Detailed Description

A cache for graph image tiles.

This class also automatically manages asychronous rendering.

7.4.2 Member Enumeration Documentation

7.4.2.1 enum ccdvl::CacheController::CacheState

Enumeration of tile cache statuses.

Enumerator:

kInvalid Invalid cache state.

kNotCached Graph tile not cached.

kInProgress Graph tile is being rendered.

kCached Graph tile is cached and ready.

7.4.3 Constructor & Destructor Documentation

7.4.3.1 ccdvl::CacheController::CacheController (Renderer * *renderer*, int32_t *cache_size*)

Cache constructor.

Initializes the cache and starts rendering threads. The renderer is kept internally and must not be destroyed before this new instance.

Parameters

in	*renderer*	The renderer to use for rendering graph tiles.
	cache_size	Cache size.

7.4.3.2 ccdvl::CacheController::~CacheController ( )

Destroys the object and frees any allocated resources.

Rendering threads are stopped, all tiles created are destroyed and becomes invalid.

7.4.4 Member Function Documentation

7.4.4.1 void ccdvl::CacheController::AddObserver (CacheObserverInterface * *observer*)

Adds an observer for changes to the active graph.

Parameters

in	*observer*	A new observer to inform.

7.4.4.2 static void* ccdvl::CacheController::CacheControllerEnter (void * *cachecontroller*) [inline], [static], [private]

Internal start method for new threads.

New threads enter a provided CacheController through this method.

Parameters

in	*cachecontroller*	The CacheController to enter.

See also

Enter().

7.4.4.3 void ccdvl::CacheController::Clear ( )

Attepts to remove all graph tiles.

Only graph tiles that have the CacheState equal to kCached are removed.

7.4.4.4 void ccdvl::CacheController::Enter ( ) [private]

Internal entry method for new threads.

Threads enter this instance through this method.

See also

CacheControllerEnter().

7.4.4.5 void ccdvl::CacheController::FlushCache (size_t *count*) [private]

Removes a number of cached states, starting with the oldest.

Parameters

count	The number of cached states to remove.

7.4.4.6 GraphTile * ccdvl::CacheController::GetGraphTile (const GraphTileState * *state*)

Request a tile from graph tile state.

The method always returns a valid tile, as long as it is not called more then once in a row (tiles from previous calls could then be deallocated) or StopRenderer() is invoked; which will invalidate any unfinnished tile.

Parameters

in	*state*	State used to generate new tile. It is used both for lookup and allocation, for lookup state will be dereferenced and for allocation it will be copied.

Returns

The associated graph tile.

See also

GraphTileStatus().

7.4.4.7 int32_t ccdvl::CacheController::GetMaxCacheSize ( )

Get the number of tiles to cache.

Returns

The maximum number of graph tiles to cache.

See also

SetMaxCacheSize().

7.4.4.8 size_t ccdvl::CacheController::GetMemoryUsage ( )

Get memory used by cached objects.

Returns

Amount of bytes allocated.

7.4.4.9 CacheController::CacheState ccdvl::CacheController::GraphTileStatus (const GraphTileState * *state*)

Check status of graph tile state.

Parameters

in	*state*	State used to generate a tile.

Returns

Cache status for the tile related to the provided state.

See also

GetGraphTile().

7.4.4.10 void ccdvl::CacheController::MemoryManagerCleared (MemoryManager & *callee*) [virtual]

Update callback when a MemoryManager is cleared.

Parameters

in	*callee*	MemoryManager that caused this update.

Implements ccdvl::MemoryManagerObserverInterface.

7.4.4.11 void ccdvl::CacheController::MemoryManagerUpdate (MemoryManager & *callee*, const AbstractDataSet ** *new_data*) [virtual]

Update callback when more data is added.

Parameters

in	*callee*	MemoryManager that caused this update.
in	*new_data*	The newly added dataset.

Implements ccdvl::MemoryManagerObserverInterface.

7.4.4.12 void ccdvl::CacheController::NotifyObservers (CacheObserverInterface::CacheEvent *reason*) [private]

Notifies all observers.

Parameters

	reason	A cause for the notification.

7.4.4.13 void ccdvl::CacheController::Remove (const GraphTileState * *state*) [private]

Attempt to remove and free a single graph tile state.

Parameters

in	*state*	The state to remove.

See also

RemoveAll() and FlushCache().

Todo Perhaps this method should return a boolean indicating if the state was destoyed or not.

7.4.4.14 void ccdvl::CacheController::RemoveAll (bool *unsafe*) [private]

Removes all cached states.

Parameters

unsafe	When set to true skip all thread and safety checks.

Note

Ensure that the cache mutex is locked before calling or that no other thread uses the cache.

7.4.4.15 void ccdvl::CacheController::SetMaxCacheSize (int32_t *cache_size*)

Sets the number of tiles to cache.

Parameters

cache_size	The maximum number of graph tiles to cache.

See also

GetMaxCacheSize().

7.4.4.16 bool ccdvl::CacheController::StopRenderer ( )

Stop renderer.

Abort current rendering operation and wipe the state render queue. Beware that doing this will invalidate any non-finnished image buffer held from earlier calls to GetGraphTile().

Returns

True if the renderer was stopped and trying to stop an idle renderer will return false.

7.4.5 Member Data Documentation

7.4.5.1 const int8_t ccdvl::CacheController::dimensions_

Number of dimensions supported.

Note

This may become irrelevant later when 3D-graphs are finnished; for now it is set to two.

7.4.5.2 bool ccdvl::CacheController::renderer_canceled_ [private]

True iff rendering is cancelled.

Used to send the correct notification to registered observers.

The documentation for this class was generated from the following files:

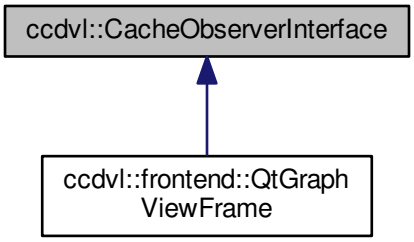
- include/cache_controller.h
- src/cache_controller.cc

7.5 ccdvl::CacheObserverInterface Class Reference

Observer for CacheController changes of active graph.

`#include <cache_observer_interface.h>`

Inheritance diagram for ccdvl::CacheObserverInterface:



Public Types

- enum CacheEvent { kRendererBegin, kRendererFinished, kRendererCanceled }

  Enumeration of different cache events.

Public Member Functions

- virtual ~CacheObserverInterface ()

  Destroys the object and frees any allocated resources.
- virtual void CacheObserverUpdate (CacheController *callee, CacheEvent reason)=0

  Cache observer update callback method.

7.5.1 Detailed Description

Observer for CacheController changes of active graph.

Template Parameters

kd	Positive number of dimensions to support, should be two or larger.

7.5.2 Member Enumeration Documentation

7.5.2.1 enum ccdvl::CacheObserverInterface::CacheEvent

Enumeration of different cache events.

Enumerator:

kRendererBegin Rendering of tiles started.

kRendererFinished Rendering of tiles finished.

kRendererCanceled Rendering of tiles was cancelled.

7.5.3 Member Function Documentation

7.5.3.1 virtual void ccdvl::CacheObserverInterface::CacheObserverUpdate (CacheController * *callee*, CacheEvent *reason*) [pure virtual]

Cache observer update callback method.

This callback method may be invoked by threads running inside the calling CacheController and therefore the implementation of this method must be threadsafe.

Parameters

in	*callee*	CacheController that caused this update.
	reason	The CacheEvent which caused this update.

Implemented in ccdvl::frontend::QtGraphViewFrame.

The documentation for this class was generated from the following file:

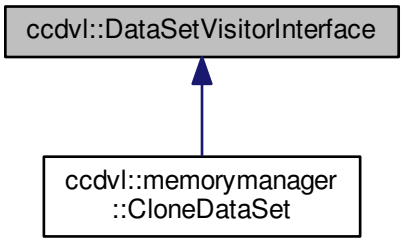
- include/cache_observer_interface.h

7.6 ccdvl::DataSetVisitorInterface Class Reference

Visitor interface for visitors accessing subtypes of AbstractDataSet.

`#include <data_set_visitor_interface.h>`

Inheritance diagram for ccdvl::DataSetVisitorInterface:



Public Member Functions

- virtual void Visit (TypedDataSet< uint8_t > *uint8)=0

  Callback for TypedDataSet holding uint8_t.
- virtual void Visit (TypedDataSet< int8_t > *int8)=0

  Callback for TypedDataSet holding int8_t.
- virtual void Visit (TypedDataSet< uint16_t > *uint16)=0

  Callback for TypedDataSet holding uint16_t.
- virtual void Visit (TypedDataSet< int16_t > *int16)=0

  Callback for TypedDataSet holding int16_t.
- virtual void Visit (TypedDataSet< uint32_t > *uint32)=0

Callback for TypedDataSet holding uint32_t.

- virtual void Visit (TypedDataSet< int32_t > *int32)=0

  Callback for TypedDataSet holding int32_t.

- virtual void Visit (TypedDataSet< uint64_t > *uint64)=0

  Callback for TypedDataSet holding uint64_t.

- virtual void Visit (TypedDataSet< int64_t > *int64)=0

  Callback for TypedDataSet holding int64_t.

- virtual void Visit (TypedDataSet< float > *tfloat)=0

  Callback for TypedDataSet holding regular floats.

- virtual void Visit (TypedDataSet< double > *tdouble)=0

  Callback for TypedDataSet holding regular doubles.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (DataSetVisitorInterface)

7.6.1 Detailed Description

Visitor interface for visitors accessing subtypes of AbstractDataSet.

7.6.2 Member Function Documentation

7.6.2.1 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< uint8_t > * *uint8*) [pure virtual]

Callback for TypedDataSet holding uint8_t.

Parameters

in,out	*uint8*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.2 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< int8_t > * *int8*) [pure virtual]

Callback for TypedDataSet holding int8_t.

Parameters

in,out	*int8*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.3 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< uint16_t > * *uint16*) [pure virtual]

Callback for TypedDataSet holding uint16_t.

Parameters

in,out	*uint16*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.4 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< int16_t > * *int16*) [pure virtual]

Callback for TypedDataSet holding int16_t.

Parameters

in,out	*int16*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.5 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< uint32_t > * *uint32*) [pure virtual]

Callback for TypedDataSet holding uint32_t.

Parameters

in,out	*uint32*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.6 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< int32_t > * *int32*) [pure virtual]

Callback for TypedDataSet holding int32_t.

Parameters

in,out	*int32*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.7 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< uint64_t > * *uint64*) [pure virtual]

Callback for TypedDataSet holding uint64_t.

Parameters

in,out	*uint64*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.8 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< int64_t > * *int64*) [pure virtual]

Callback for TypedDataSet holding int64_t.

Parameters

in,out	*int64*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.9 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< float > * *tfloat*) [pure virtual]

Callback for TypedDataSet holding regular floats.

Parameters

in,out	*tfloat*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

7.6.2.10 virtual void ccdvl::DataSetVisitorInterface::Visit (TypedDataSet< double > * *tdouble*) [pure virtual]

Callback for TypedDataSet holding regular doubles.

Parameters

in,out	*tdouble*	Calling class instance.

Implemented in ccdvl::memorymanager::CloneDataSet.

The documentation for this class was generated from the following file:

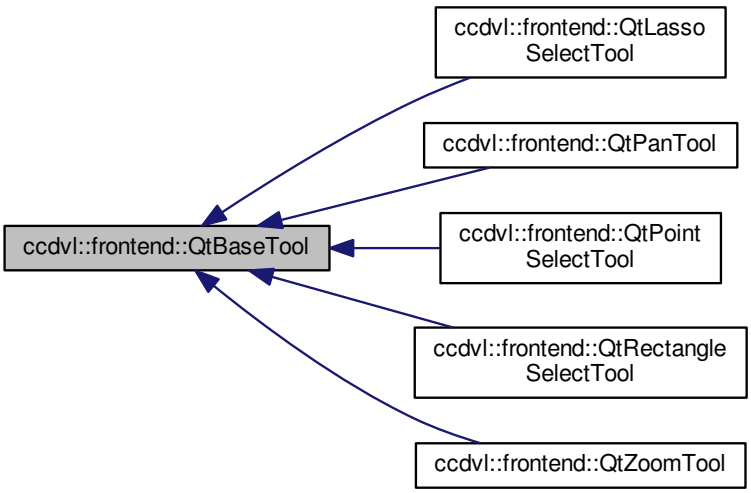
- include/data_set/data_set_visitor_interface.h

7.7 ccdvl::frontend::QtBaseTool Class Reference

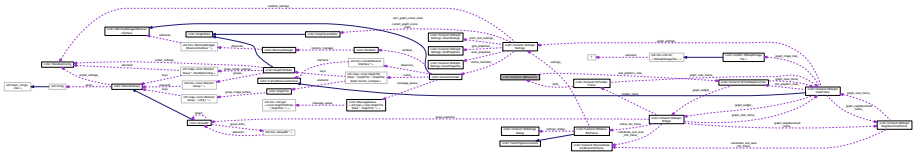
A base class for tools providing basic functionality and event handling functions to be overridden by subclasses.

`#include <qt_base_tool.h>`

Inheritance diagram for ccdvl::frontend::QtBaseTool:



Collaboration diagram for ccdvl::frontend::QtBaseTool:



Public Member Functions

- QtBaseTool (QtToolGraphicsView ∗tool_graphics_view)

  Constructs and initiates an abstract tool with the given QtToolGraphicsView.
- virtual void OnActivate ()

  Should be called when the tool is activated or selected to take care of any initiations and such that may be needed.
- virtual void OnDeactivate ()

  Should be called when the tool is deactivated or deselected to take care of any clean up and such that may be needed.
- virtual bool OnMousePress (QMouseEvent ∗event)

  Handles forwarded mouse button press events.
- virtual bool OnMouseMove (QMouseEvent ∗event)

  Handles forwarded mouse move events.
- virtual bool OnMouseRelease (QMouseEvent ∗event)

  Handles forwarded mouse button release events.
- virtual bool OnWheel (QWheelEvent ∗event)

  Handles forwarded mouse wheel events.
- virtual bool OnKeyPress (QKeyEvent ∗event)

  Handles forwarded key press events.
- virtual bool OnKeyRelease (QKeyEvent ∗event)

  Handles forwarded key release events.
- virtual bool OnEnter (QEvent ∗event)

  Handles forwarded enter events.
- virtual bool OnLeave (QEvent ∗event)

  Handles forwarded leave events.
- virtual bool has_selection ()

  Checks if the tool has selection capabilities.
- virtual GraphPolygonF selection ()

  The current selection made by the tool (in graph coordinates).
- virtual void set_selection (GraphPolygonF selection)

  Sets the current selection made by the tool (in graph coordinates).
- virtual void SetDefaultCursor ()

  Sets the cursor shown in the tool graphics view to default_cursor_.

Protected Member Functions

- ViewPoint RestrictPosToView (const ScenePoint &p)

  Restricts (clamps) the given position p so that it lies within the currently displayed ranges of the graphics view.
- ScenePointF MapToRestrictedScene (const ViewPoint &p)

  Converts a view position to scene coordinates that are within the current visible graphics view.
- GraphPointF MapToRestrictedGraph (const ViewPoint &p)

  Converts a view position to graph coordinates that are within the current visible graphics view.

Protected Attributes

- QtToolGraphicsView * tool_graphics_view_

  The tool graphics view used with the tool.
- QCursor default_cursor_

  The default mouse cursor of the tool.
- QPointF click_pos_

  The point where the tool was first clicked.
- GraphPolygonF selection_

  The last selection of the tool.
- bool has_selection_

  States if the tool has selection capabilities.

7.7.1 Detailed Description

A base class for tools providing basic functionality and event handling functions to be overridden by subclasses.

Provides functions to be called on activation, deactivation, mouse button press, mouse move, mouse buton release, wheel, key press and key release events, as well as enter and leave events regarding the QtToolGraphicsView. All the events are ignored in this implementation.

Functions for storing a selection is also available, and tools with selection capabilities should make sure that has_selection() returns true. has_selection() should also always be checked before attempting to get or set the tool selection. The selection of a tool without selection capabilities is undefined.

There are also some helper functions used to clamp and convert coordinates.

Note

The event handeling functions does not automatically catch the events, but must be explicitly forwarded from the respective Qt event handlers. The return values from the event handling functions tells the caller if the event was accepted or not.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 ccdvl::frontend::QtBaseTool::QtBaseTool (QtToolGraphicsView * *tool_graphics_view*) [explicit]

Constructs and initiates an abstract tool with the given QtToolGraphicsView.

Parameters

in	*tool_graphics_view*	The tool graphics view used with the tool.

7.7.3 Member Function Documentation

7.7.3.1 bool ccdvl::frontend::QtBaseTool::has_selection ( ) [virtual]

Checks if the tool has selection capabilities.

Tools with selection capabilities have valid selections.

Returns

True if the tool has selection capabilities.

See also

selection() and set_selection(GraphPolygonF)

7.7.3.2 GraphPointF ccdvl::frontend::QtBaseTool::MapToRestrictedGraph (const ViewPoint & *p*) [protected]

Converts a view position to graph coordinates that are within the current visible graphics view.

Parameters

in	*p*	The position to restrict and convert.

Returns

The restricted and converted position.

7.7.3.3 ScenePointF ccdvl::frontend::QtBaseTool::MapToRestrictedScene (const ViewPoint & *p*) [protected]

Converts a view position to scene coordinates that are within the current visible graphics view.

Parameters

in	*p*	The position to restrict and convert.

Returns

The restricted and converted position.

7.7.3.4 bool ccdvl::frontend::QtBaseTool::OnEnter (QEvent * *event*) [virtual]

Handles forwarded enter events.

Parameters

in	*event*	The enter event.

Returns

true if the event was accepted.

See also

OnLeave(QEvent*)

Reimplemented in ccdvl::frontend::QtPointSelectTool.

7.7.3.5 bool ccdvl::frontend::QtBaseTool::OnKeyPress (QKeyEvent * *event*) [virtual]

Handles forwarded key press events.

Parameters

in	*event*	The key event.

Returns

true if the event was accepted.

See also

OnKeyRelease(QKeyEvent*)

Reimplemented in ccdvl::frontend::QtZoomTool.

7.7.3.6 bool ccdvl::frontend::QtBaseTool::OnKeyRelease (QKeyEvent * *event*) [virtual]

Handles forwarded key release events.

Parameters

in	*event*	The key event.

Returns

true if the event was accepted.

See also

OnKeyPress(QKeyEvent*)

Reimplemented in ccdvl::frontend::QtZoomTool.

7.7.3.7 bool ccdvl::frontend::QtBaseTool::OnLeave (QEvent * *event*) [virtual]

Handles forwarded leave events.

Parameters

in	*event*	The leave event.

Returns

true if the event was accepted.

See also

OnEnter(QEvent*)

Reimplemented in ccdvl::frontend::QtPointSelectTool.

7.7.3.8 bool ccdvl::frontend::QtBaseTool::OnMouseMove (QMouseEvent * *event*) [virtual]

Handles forwarded mouse move events.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented in ccdvl::frontend::QtLassoSelectTool, ccdvl::frontend::QtZoomTool, ccdvl::frontend::QtPointSelectTool, ccdvl::frontend::QtRectangleSelectTool, and ccdvl::frontend::QtPanTool.

7.7.3.9 bool ccdvl::frontend::QtBaseTool::OnMousePress (QMouseEvent * *event*) [virtual]

Handles forwarded mouse button press events.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented in ccdvl::frontend::QtLassoSelectTool, ccdvl::frontend::QtZoomTool, ccdvl::frontend::QtPointSelectTool, ccdvl::frontend::QtPanTool, and ccdvl::frontend::QtRectangleSelectTool.

7.7.3.10 bool ccdvl::frontend::QtBaseTool::OnMouseRelease (QMouseEvent * *event*) [virtual]

Handles forwarded mouse button release events.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseMove(QMouseEvent*)

Reimplemented in ccdvl::frontend::QtLassoSelectTool, ccdvl::frontend::QtZoomTool, ccdvl::frontend::QtRectangleSelectTool, and ccdvl::frontend::QtPanTool.

7.7.3.11 bool ccdvl::frontend::QtBaseTool::OnWheel (QWheelEvent * *event*) [virtual]

Handles forwarded mouse wheel events.

Parameters

in	*event*	The wheel event.

Returns

true if the event was accepted.

Reimplemented in ccdvl::frontend::QtZoomTool, ccdvl::frontend::QtLassoSelectTool, ccdvl::frontend::QtPointSelectTool, ccdvl::frontend::QtRectangleSelectTool, and ccdvl::frontend::QtPanTool.

7.7.3.12 ViewPoint ccdvl::frontend::QtBaseTool::RestrictPosToView (const ScenePoint & *p*) [protected]

Restricts (clamps) the given position *p* so that it lies within the currently displayed ranges of the graphics view.

Parameters

in	*p*	The position to restrict.

Returns

The restricted position.

7.7.3.13 GraphPolygonF ccdvl::frontend::QtBaseTool::selection ( ) [virtual]

The current selection made by the tool (in graph coordinates).

Returns

The selection or an empty polygon if there is no selection.

See also

set_selection(GraphPolygonF) and has_selection()

7.7.3.14 void ccdvl::frontend::QtBaseTool::set_selection (GraphPolygonF *selection*) [virtual]

Sets the current selection made by the tool (in graph coordinates).

Parameters

selection	[in] The new selection.

See also

selection() and has_selection()

7.7.4 Member Data Documentation

7.7.4.1 QPointF ccdvl::frontend::QtBaseTool::click_pos_ [protected]

The point where the tool was first clicked.

This point is used in different coordinate systems for convenience, depending on the implementation of the subclass.

The documentation for this class was generated from the following files:

- include/qt_frontend/qt_base_tool.h
- src/qt_frontend/qt_base_tool.cc

7.8 ccdvl::frontend::QtCoordinateAndAxesInfoFrame Class Reference

The panel containing information and controls for displaying the x/y-coordinate for mouse over and ranges in the graph.

`#include <qt_coordinate_and_axes_info_frame.h>`

Public Member Functions

- QtCoordinateAndAxesInfoFrame (QWidget *parent=NULL)

  Constructs and initiates a coordinate and axes information frame with the given parent.
- int Init ()

  Initiates the GUI components used for displaying x/y-coordinates and the axes' ranges (min, max and step values).
- void SetXYCoordinates (const GraphPointF &pos)

  Sets the x and y coordinate mouse over label texts with three decimals precision.
- void SetXYCoordinates (GraphDouble x, GraphDouble y)

  A overridden convenience function.
- void SetRangesInfo (GraphDouble x_min, GraphDouble x_max, GraphDouble y_min, GraphDouble y_max, double x_scale, double y_scale, double x_zoom, double y_zoom)

  Sets the displayed ranges info.
- QString FormattedNumberText (GraphDouble xy, double upper_sci_bound, double lower_sci_bound, int normal_precision, int sci_precision)

  Returns a formatted string of the given xy value with the given precision.

Private Member Functions

- void CreateLabels ()

  Creates the labels.

Private Attributes

- QGridLayout * main_layout_

  The main layout containing all components of the object.
- QLabel * x_coordinate_label_

  The x coordinate label displaying the current mouse over graph position.
- QLabel * y_coordinate_label_

  The y coordinate label displaying the current mouse over graph position.
- QLabel * x_min_label_

  The label displaying the scene minimum X graph value.
- QLabel * x_max_label_

  The label displaying the scene maximum X graph value.
- QLabel * x_scale_label_

  The label displaying the graph X scale value.
- QLabel * x_zoom_label_

  The label displaying the graph X zoom value.
- QLabel * y_min_label_

  The label displaying the scene minimum Y graph value.
- QLabel * y_max_label_

  The label displaying the scene maximum Y graph value.
- QLabel * y_scale_label_

  The label displaying the graph Y scale value.
- QLabel * y_zoom_label_

  The label displaying the graph Y zoom value.

7.8.1 Detailed Description

The panel containing information and controls for displaying the x/y-coordinate for mouse over and ranges in the graph.

This panel shows the current x- and y-coordinates under the mouse cursor in the graph view, the coordinates for minimum, maximum and step values of the displayed view.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 ccdvl::frontend::QtCoordinateAndAxesInfoFrame::QtCoordinateAndAxesInfoFrame (QWidget * *parent* = NULL) [explicit]

Constructs and initiates a coordinate and axes information frame with the given parent.

Parameters

in	*parent*	The parent widget.

7.8.3 Member Function Documentation

7.8.3.1 QString ccdvl::frontend::QtCoordinateAndAxesInfoFrame::FormattedNumberText (GraphDouble *xy*, double *upper_sci_bound*, double *lower_sci_bound*, int *normal_precision*, int *sci_precision*)

Returns a formatted string of the given *xy* value with the given precision.

The returned string is an integer if the absolute value of *xy* is less than *upper_sci_bound* and more than *lower_sci_bound*, e.g. `FormattedNumberText(5123, 10000, 0.01)` returns *5123*.

Otherwise it is written with scientific notation with a float followed by times 10 with the exponent in superscript, e.g. *`FormattedNumberText(51234, 10000, 0.01)`* returns

5.123×10^{+03}

and `FormattedNumberText(0.051234, 10000, 0.01)` returns

5.123×10^{-02} .

Parameters

in	*xy*	The coordinate value to be converted.
in	*upper_sci_bound*	The upper bound over which scientific notation is to be used.
in	*lower_sci_bound*	The lower bound under which scientific notation is to be used.
in	*normal_precision*	The precision to be used for normal notation.
in	*sci_precision*	The precision to be used for the scientidic notation.

Returns

The value of *xy* as a formatted string.

7.8.3.2 int ccdvl::frontend::QtCoordinateAndAxesInfoFrame::Init ( )

Initiates the GUI components used for displaying x/y-coordinates and the axes' ranges (min, max and step values).

Returns

0 if the initializiation was successfull.

7.8.3.3 void ccdvl::frontend::QtCoordinateAndAxesInfoFrame::SetRangesInfo (GraphDouble *x_min*, GraphDouble *x_max*, GraphDouble *y_min*, GraphDouble *y_max*, double *x_scale*, double *y_scale*, double *x_zoom*, double *y_zoom*)

Sets the displayed ranges info.

The ranges are displayed with three decimals precision, and the scale and zoom levels are displayed as integral percentage numbers.

Parameters

in	*x_min*	The minimum x value.
in	*x_max*	The maximum x value.
in	*y_min*	The minimum y value.
in	*y_max*	The maximum y value.
in	*x_scale*	The x scale value.
in	*y_scale*	The y scale value.
in	*x_zoom*	The x zoom level.
in	*y_zoom*	The y zoom level.

7.8.3.4 void ccdvl::frontend::QtCoordinateAndAxesInfoFrame::SetXYCoordinates (const GraphPointF & *pos*)

Sets the x and y coordinate mouse over label texts with three decimals precision.

Parameters

in	*pos*	The position as a point.

7.8.3.5 void ccdvl::frontend::QtCoordinateAndAxesInfoFrame::SetXYCoordinates (GraphDouble *x*, GraphDouble *y*)

A overridden convenience function.

Parameters

in	*x*	The x-coordinate of the position.
in	*y*	The y-coordinate of the position.

The documentation for this class was generated from the following files:

- include/qt_frontend/qt_coordinate_and_axes_info_frame.h
- src/qt_frontend/qt_coordinate_and_axes_info_frame.cc

7.9 ccdvl::frontend::QtGraphImageTile Class Reference

A class representing a graph image tile; a `QPixmap` and a flag that keeps track of if the graph image tile has been drawn onto the graph image.

`#include <qt_graph_image_tile.h>`

Public Member Functions

- QtGraphImageTile ()

  Constructs a graph image tile with a "null" image and the drawn flag set to false.
- QtGraphImageTile (const QtGraphImageTile &tile)

  Constructs a copy of the given tile.

- QtGraphImageTile (const QPixmap &image)

  Creates a graph image tile with the given image and the drawn flag set to false.
- QtGraphImageTile (bool drawn)

  Constructs a graph image tile with a "null" image and sets the drawn flag to the given drawn state.
- QtGraphImageTile (const QPixmap &image, bool drawn)

  Constructs a graph image tile and sets the image to the given image and the drawn flag to the given drawn state.
- const QPixmap & image ()

  The image of the graph image tile, which may be "null" if not set or cleared.
- void set_image (const QPixmap &image)

  Sets the image of the graph image tile.
- bool drawn ()

  The drawn flag state, indicating if the graph image tile has been drawn to the graph image or not.
- void set_drawn (bool drawn)

  Sets the state of the drawn flag.
- void Clear ()

  Clears and resets the graph image tile image and drawn flag.

Private Attributes

- QPixmap image_

  The image of the graph image tile.
- bool drawn_

  States if the graph image tile has been drawn to the graph image.

7.9.1 Detailed Description

A class representing a graph image tile; a `QPixmap` and a flag that keeps track of if the graph image tile has been drawn onto the graph image.

7.9.2 Constructor & Destructor Documentation

7.9.2.1 ccdvl::frontend::QtGraphImageTile::QtGraphImageTile ( )

Constructs a graph image tile with a "null" image and the drawn flag set to false.

See also

QtGraphImageTile(const QtGraphImageTile&), QtGraphImageTile(const QPixmap&), QtGraphImageTile(bool) and QtGraphImageTile(const QPixmap&, bool)

7.9.2.2 ccdvl::frontend::QtGraphImageTile::QtGraphImageTile (const QtGraphImageTile & *tile*)

Constructs a copy of the given *tile*.

Parameters

in	*tile*	The graph image tile to copy.

See also

QtGraphImageTile(), QtGraphImageTile(const QPixmap&), QtGraphImageTile(bool) and QtGraphImageTile(const QPixmap&, bool)

7.9.2.3 ccdvl::frontend::QtGraphImageTile::QtGraphImageTile (const QPixmap & *image*)

Creates a graph image tile with the given *image* and the drawn flag set to false.

Parameters

in	*image*	The image to use.

See also

QtGraphImageTile(), QtGraphImageTile(const QtGraphImageTile&), QtGraphImageTile(bool) and QtGraphImageTile(const QPixmap&, bool)

7.9.2.4 ccdvl::frontend::QtGraphImageTile::QtGraphImageTile (bool *drawn*) [explicit]

Constructs a graph image tile with a "null" image and sets the drawn flag to the given *drawn* state.

Parameters

in	*drawn*	The drawn flag state.

See also

QtGraphImageTile(), QtGraphImageTile(const QtGraphImageTile&), QtGraphImageTile(const QPixmap&) and QtGraphImageTile(const QPixmap&, bool)

7.9.2.5 ccdvl::frontend::QtGraphImageTile::QtGraphImageTile (const QPixmap & *image*, bool *drawn*)

Constructs a graph image tile and sets the image to the given *image* and the drawn flag to the given *drawn* state.

Parameters

in	*image*	The image to use.
in	*drawn*	The drawn flag state.

See also

QtGraphImageTile(), QtGraphImageTile(const QtGraphImageTile&), QtGraphImageTile(const QPixmap&) and QtGraphImageTile(bool)

7.9.3 Member Function Documentation

7.9.3.1 bool ccdvl::frontend::QtGraphImageTile::drawn ( )

The drawn flag state, indicating if the graph image tile has been drawn to the graph image or not.

Returns

The state of the drawn flag.

See also

set_drawn(bool)

7.9.3.2 const QPixmap & ccdvl::frontend::QtGraphImageTile::image ( )

The image of the graph image tile, which may be "null" if not set or cleared.

Returns

The tile image.

See also

set_image(QPixmap) and `QPixmap::isNull()`

7.9.3.3 void ccdvl::frontend::QtGraphImageTile::set_drawn (bool *drawn*)

Sets the state of the drawn flag.

Parameters

in	*drawn*	The new state of the drawn flag.

See also

drawn()

7.9.3.4 void ccdvl::frontend::QtGraphImageTile::set_image (const QPixmap & *image*)

Sets the image of the graph image tile.

Parameters

in	*image*	The new image.

See also

image()

The documentation for this class was generated from the following files:

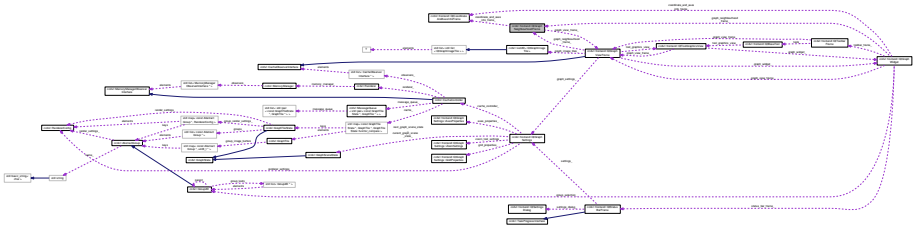
- include/qt_frontend/qt_graph_image_tile.h
- src/qt_frontend/qt_graph_image_tile.cc

7.10 ccdvl::frontend::QtGraphNeighbourhoodFrame Class Reference

Shows a small overview image of the neighbourhood area of the current graph view with a rectangle outlining the current view.

`#include <qt_graph_neighbourhood_frame.h>`

Collaboration diagram for ccdvl::frontend::QtGraphNeighbourhoodFrame:



Public Member Functions

- QtGraphNeighbourhoodFrame (QtCoordinateAndAxesInfoFrame *coordinate_and_axes_info_frame, QWidget *parent=NULL)

  Constructs and initiates a graph neighbourhood frame with the given graph view frame and parent.
- int Init (QtGraphViewFrame *graph_view_frame)

  Creates and initiates the neighbourhood image and neighbourhood group box.
- void ShowLoadingMessage ()

  Changes the displayed image to show a loading message.
- void UpdateNeighbourhoodPixmap (QPixmap graph_image)

  Updates the shown neighbourhood pixmap.
- void UpdateGraphViewOutlinePosition ()

  Redraws the rectangle outline to correspond with the current graph view.

Protected Member Functions

- bool eventFilter (QObject *object, QEvent *event)

  An event filter to capture mouse clicks.

Private Member Functions

- SceneSizeF ScaleFactor ()

  The current scale factor of the neighbourhood / graph view.
- SceneRectF OutlineRect ()

  Calculates the outline rect.

Private Attributes

- QtCoordinateAndAxesInfoFrame * coordinate_and_axes_info_frame_

  The coordinate and axes info frame used to update mouse over graph coordinates.
- QtGraphViewFrame * graph_view_frame_

  The graph view frame to show the neighbourhood of.
- QGridLayout * main_layout_

  The main layout containing all components of the object.
- QLabel * neighbourhood_label_

  The label showing the pixmap.
- QPixmap neighbourhood_pixmap_

  The pixmap showing the neighbourhood image.
- bool loading_

  Keeps track of if the neighbourhood label can be interacted with.

7.10.1 Detailed Description

Shows a small overview image of the neighbourhood area of the current graph view with a rectangle outlining the current view.

Clicking in the neighbourhood image will recenter the graph view on the corresponding position. Clicking and draging the putline rectangel will pan the view correspondingly.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 ccdvl::frontend::QtGraphNeighbourhoodFrame::QtGraphNeighbourhoodFrame (QtCoordinateAndAxesInfoFrame * *coordinate_and_axes_info_frame,* QWidget * *parent* = NULL)

Constructs and initiates a graph neighbourhood frame with the given graph view frame and *parent*.

Parameters

in	*coordinate_and_axes_info_frame*	The coordinate and axes info frame used to update mouse over graph coordinates.
in	*parent*	The parent widget.

7.10.3 Member Function Documentation

7.10.3.1 bool ccdvl::frontend::QtGraphNeighbourhoodFrame::eventFilter (QObject * *object,* QEvent * *event*) [protected]

An event filter to capture mouse clicks.

Parameters

in	*object*	The watched object.
in	*event*	The event to filter.

Returns

true if the event was handled.

7.10.3.2 int ccdvl::frontend::QtGraphNeighbourhoodFrame::Init (QtGraphViewFrame * *graph_view_frame*)

Creates and initiates the neighbourhood image and neighbourhood group box.

Parameters

in	*graph_view_frame*	The graph view frame to show the neighbourhood of.

Returns

0 if the initializiation was successfull.

7.10.3.3 SceneRectF ccdvl::frontend::QtGraphNeighbourhoodFrame::OutlineRect ( ) [private]

Calculates the outline rect.

Returns

The current outline rect.

7.10.3.4 SceneSizeF ccdvl::frontend::QtGraphNeighbourhoodFrame::ScaleFactor ( ) [private]

The current scale factor of the neighbourhood / graph view.

Returns

The scale factor used to resize the image and the outline.

7.10.3.5 void ccdvl::frontend::QtGraphNeighbourhoodFrame::ShowLoadingMessage ( )

Changes the displayed image to show a loading message.

Can be used between updates of the neighbourhood pixmap.

See also

UpdateNeighbourhoodPixmap()

7.10.3.6 void ccdvl::frontend::QtGraphNeighbourhoodFrame::UpdateGraphViewOutlinePosition ( )

Redraws the rectangle outline to correspond with the current graph view.

See also

UpdateNeighbourhoodPixmap()

7.10.3.7 void ccdvl::frontend::QtGraphNeighbourhoodFrame::UpdateNeighbourhoodPixmap (QPixmap *graph_image*)

Updates the shown neighbourhood pixmap.

Resizes the provided graph image, making it fit the neighbourhood display area and masking out the points for better contrast. It also calls UpdateGraphViewOutlinePosition().

Parameters

in	*graph_image*	The graph image to display as a neighbourhood overview.

See also

UpdateGraphViewOutlinePosition()

The documentation for this class was generated from the following files:

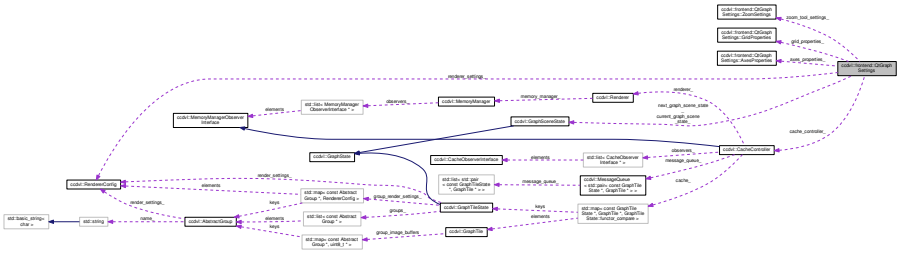
- include/qt_frontend/qt_graph_neighbourhood_frame.h
- src/qt_frontend/qt_graph_neighbourhood_frame.cc

7.11 ccdvl::frontend::QtGraphSettings Class Reference

Holds the configurable settings related to the Qt frontend, with public access to the setting member variables, rather than using mutators, for convenience.

`#include <qt_graph_settings.h>`

Collaboration diagram for ccdvl::frontend::QtGraphSettings:



Classes

- struct AxesProperties

  A struct to hold the axes properties.
- struct GridProperties

  A struct to hold the grid properties.
- struct ZoomSettings

  A struct to hold the zoom settings; x and y zoom steps, zoom step factors and mouse wheel zoom step factors.

Public Types

- enum GridType { kStatic, kRelativeToGraph, kRelativeToZoom }

  The display behaviour of the grid.

Signals

- void SettingsUpdated (bool renderer, bool graph, bool grid, bool axes)

  Signal emited whenever any of these settings change.

Public Member Functions

- QtGraphSettings (QObject *parent=NULL)

  Constructs a graph settings object with the given parent and initiates the settings to default values.
- void ConvertClearColor (uint8_t color[3])

  Convert clear color to RGB888.
- void EmitSettingsUpdated (bool renderer, bool graph, bool grid, bool axes)

  Emits a settings updated signal.

Public Attributes

- GridProperties grid_properties_

  Graph grid properties.
- AxesProperties axes_properties_

  Graph axes properties.
- ZoomSettings zoom_tool_settings_

  Zoom tool settings.
- RendererConfig renderer_settings_

Default render configuration.

- CacheController * cache_controller_

  Currently used graph cache that caches graph images and render them as needed.

- int32_t image_tile_width_

  The graph image tile width.

- int32_t image_tile_height_

  The graph image tile height.

- int image_tile_rows_

  The number of graph image tile rows.

- int image_tile_columns_

  The number of graph image tile columns.

- GraphSceneState * current_graph_scene_state_

  Current graph scene state.

- GraphSceneState * next_graph_scene_state_

  Next graph scene state.

- QColor clear_color_

  Clear color (graph background color).

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (QtGraphSettings)

7.11.1 Detailed Description

Holds the configurable settings related to the Qt frontend, with public access to the setting member variables, rather than using mutators, for convenience.

Note that image_tile_width_ and image_tile_height_ should be changed using QtGraphViewFrame::SetGraphImageTileWidth(int) and QtGraphViewFrame::SetGraphImageTileHeight(int) respectively to maintain a vaid graph state.

7.11.2 Member Enumeration Documentation

7.11.2.1 enum ccdvl::frontend::QtGraphSettings::GridType

The display behaviour of the grid.

Enumerator:

kStatic The grid will be statically drawn at the same line intervals regardless of the current view position and zoom level.

kRelativeToGraph The grid lines will follow the shown graph coordinates and zoom level. Beware of low zoom levels and high scale values, as more grid lines will have to be drawn, slowing down the rendering process (often noticeably).

kRelativeToZoom The grid lines are static, but follow the zoom level. That is, the grid line interval is multiplied with the current zoom level, which is more or less identical to GridType::kRelativeToGraph when both the x and y scales are 100%.

7.11.3 Constructor & Destructor Documentation

7.11.3.1 ccdvl::frontend::QtGraphSettings::QtGraphSettings (QObject * *parent* = NULL) [explicit]

Constructs a graph settings object with the given *parent* and initiates the settings to default values.

Parameters

in	*parent*	The parent widget.

7.11.4 Member Function Documentation

7.11.4.1 void ccdvl::frontend::QtGraphSettings::ConvertClearColor (uint8_t *color[3]*) [inline]

Convert clear color to RGB888.

Parameters

out	*color*	Destination for the converted color.

7.11.4.2 void ccdvl::frontend::QtGraphSettings::EmitSettingsUpdated (bool *renderer,* bool *graph,* bool *grid,* bool *axes*) [inline]

Emits a settings updated signal.

This method must be invoked whenever any settings are changed. It is also always done when user hits apply in the settings dialog.

Parameters

renderer	true iff any of the renderer settings changed. This includes point shape, line, scaling, etc.
graph	true iff any graph view settings changed.
grid	true iff any grid settings changed.
axes	true iff any axis setting changed.

See also

SettingsUpdated()

7.11.4.3 void ccdvl::frontend::QtGraphSettings::SettingsUpdated (bool *renderer,* bool *graph,* bool *grid,* bool *axes*) [signal]

Signal emited whenever any of these settings change.

Parameters

renderer	True iff any of the renderer settings changed. This includes point shape, line, scaling, etc.
graph	true iff any graph view settings changed.
grid	true iff any grid settings changed.
axes	true iff any axis setting changed.

Note

This signal could be emited without any parameters being true. This indicates that one of the settings that have no indicator argument such as maximum number of cached tiles was changed.

7.11.5 Member Data Documentation

7.11.5.1 GraphSceneState* ccdvl::frontend::QtGraphSettings::current_graph_scene_state_

Current graph scene state.

This object represents the current graph scene, is managed elsewhere, and should be replaced by next_graph_-scene_state_ when updated.

See also

next_graph_scene_state_

7.11.5.2 int32_t ccdvl::frontend::QtGraphSettings::image_tile_height_

The graph image tile height.

Use QtGraphViewFrame::SetGraphImageTileHeight(int) when changing this to keep a valid graph state.

See also

QtGraphViewFrame::SetGraphImageTileHeight(int)

7.11.5.3 int32_t ccdvl::frontend::QtGraphSettings::image_tile_width_

The graph image tile width.

Use QtGraphViewFrame::SetGraphImageTileWidth(int) when changing this to keep a valid graph state.

See also

QtGraphViewFrame::SetGraphImageTileWidth(int)

7.11.5.4 GraphSceneState* ccdvl::frontend::QtGraphSettings::next_graph_scene_state_

Next graph scene state.

This object represents the unfinished graph scene, is managed elsewhere, and should be used to update current_-graph_scene_state_ by replacing it.

See also

QtGraphViewFrame::UpdateGraphView() and current_graph_scene_state_

7.11.5.5 RendererConfig ccdvl::frontend::QtGraphSettings::renderer_settings_

Default render configuration.

Settings containing point and line color, size and shape.

The documentation for this class was generated from the following files:

- include/qt_frontend/qt_graph_settings.h
- src/qt_frontend/qt_graph_settings.cc

7.12 ccdvl::frontend::QtGraphSettings::AxesProperties Struct Reference

A struct to hold the axes properties.

`#include <qt_graph_settings.h>`

Public Attributes

- QColor line_color

  Color of axis dashes.
- QString x_label

  X name.
- QString y_label

  Y name.
- GraphInt x_step

  Distance between the big dashes on the x-axis.
- GraphInt y_step

  Distance between the big dashes on the y-axis.
- SceneInt x_spacer

  Distance between the small dashes in pixels.
- SceneInt y_spacer

  Distance between the small dashes in pixels.
- int normal_precision

  Number of decimals shown for non-sci numbers.
- int scientific_precision

  Number of decimals shown for sci numbers.
- double upper_scientific_bound
- double lower_scientific_bound

7.12.1 Detailed Description

A struct to hold the axes properties.

7.12.2 Member Data Documentation

7.12.2.1 double ccdvl::frontend::QtGraphSettings::AxesProperties::lower_scientific_bound

Values between this or its negative counterpart and zero will be displayed with scientific notation.

7.12.2.2 double ccdvl::frontend::QtGraphSettings::AxesProperties::upper_scientific_bound

Values exceeding this (and lower then its negative counterpart) will be displayed with scientific notation.

The documentation for this struct was generated from the following file:

- include/qt_frontend/qt_graph_settings.h

7.13 ccdvl::frontend::QtGraphSettings::GridProperties Struct Reference

A struct to hold the grid properties.

`#include <qt_graph_settings.h>`

Public Attributes

- QColor line_color

  The color of grid lines.
- double x_step

  The step distance between X lines.
- double y_step

  The step distance between Y lines.
- bool visible

  Decides if the grid should be drawn or not.
- GridType type

  Decides how step distance is interpreted.

7.13.1 Detailed Description

A struct to hold the grid properties.

The documentation for this struct was generated from the following file:

- include/qt_frontend/qt_graph_settings.h

7.14 ccdvl::frontend::QtGraphSettings::ZoomSettings Struct Reference

A struct to hold the zoom settings; x and y zoom steps, zoom step factors and mouse wheel zoom step factors.

`#include <qt_graph_settings.h>`

Public Attributes

- double x_zoom_step

  The smallest zoom step in x.
- double y_zoom_step

  The smallest zoom step in y.
- int x_zoom_step_factor

  The number of x zoom steps done when zooming.
- int y_zoom_step_factor

  The number of y zoom steps done when zooming.
- int x_wheel_zoom_step_factor
- int y_wheel_zoom_step_factor

7.14.1 Detailed Description

A struct to hold the zoom settings; x and y zoom steps, zoom step factors and mouse wheel zoom step factors.

7.14.2 Member Data Documentation

7.14.2.1 int ccdvl::frontend::QtGraphSettings::ZoomSettings::x_wheel_zoom_step_factor

The number of x zoom steps performed when zooming using the mouse scroll wheel.

7.14.2.2 int ccdvl::frontend::QtGraphSettings::ZoomSettings::y_wheel_zoom_step_factor

The number of y zoom steps performed when zooming using the mouse scroll wheel.

The documentation for this struct was generated from the following file:

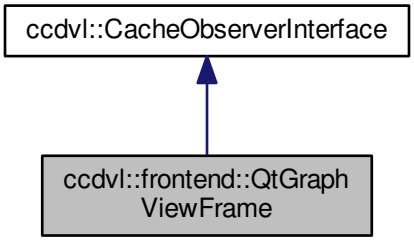
- include/qt_frontend/qt_graph_settings.h

7.15 ccdvl::frontend::QtGraphViewFrame Class Reference

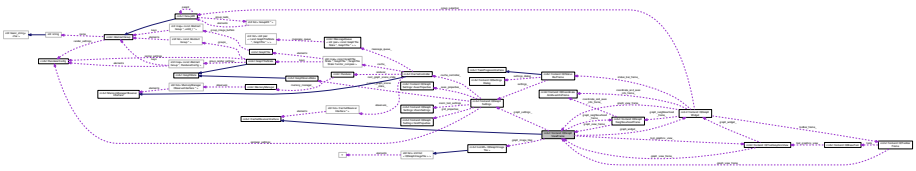
A frame with the rendered plot graphics and axes.

`#include <qt_graph_view_frame.h>`

Inheritance diagram for ccdvl::frontend::QtGraphViewFrame:



Collaboration diagram for ccdvl::frontend::QtGraphViewFrame:



Public Slots

- void UpdateGraphView ()

  Updates the graph states in the graph cache used by the graph image tiles with the current center position, zoom and scale factors. Progressive graph updates are also started if needed.
- void UpdateGraphView (GraphPointF center_pos)

  Updates the graph states in the graph cache used by the graph image tiles with the given center position and the current zoom and scale factors. Progressive graph updates are also started if needed.
- void UpdateGraphView (GraphPointF center_pos, double x_zoom, double y_zoom)

  Updates the graph states in the graph cache used by the graph image tiles with the given center position, zoom and current scale factors. Progressive graph updates are also started if needed.
- void UpdateGraphView (GraphPointF center_pos, double x_zoom, double y_zoom, double x_scale, double y_scale)

  Updates the graph states in the graph cache used by the graph image tiles with the given center position, zoom and scale factors. Progressive graph updates are also started if needed.

- void StartProgressiveGraphUpdates ()

  Starts the graph update timer that triggers progressive graph updates using the kDefaultGraphUpdateInterval.
- void StartProgressiveGraphUpdates (int update_interval)

  Starts or restarts the graph update timer that triggers progressive graph updates using the given update_interval.
- void UpdateBegun ()

  Sets the stop and reload button properties when an update begins.
- void UpdateCanceled ()

  Sets the stop and reload button properties when an update is canceled, and stops the graph view update.
- void UpdateFinished ()

  Sets the stop and reload button properties when an update finishes.
- void StopUpdateButtonClicked ()

  Stops the current graph view update and the renderer if it is running.
- void ReloadUpdateButtonClicked ()

  Reloads the graph view with the current graph state properties.
- void DrawGraphView ()

  Progressively draws and shows the graph image composed of the graph image tiles fetched from the graph cache state image buffers.
- void UpdateGraphGlassPane ()

  Updates the graph glass pane and redraws the grid on it.
- bool AddGraphImageTilesOnResize ()

  Expands the graph scene, if neccessary, by adding enough tiles to cover the current graph view.
- void SynchronizeViewAndAxes ()

  Sets the axes scroll bar positions to the corresponding scroll bar positions of the tool graphics view.
- void SettingsChanged (bool renderer, bool graph, bool grid, bool axes)

  Handles changes of the graph configuration class QtSettings.

Signals

- void BeginUpdate ()

  Sending this signal indicates that an update has been started by the sender.
- void CancelUpdate ()

  Sending this signal cancels any ongoing update of the graph view.
- void FinishUpdate ()

  Sending this signal indicates that the current ongoing update is done and that the GUI should reflect this.

Public Member Functions

- QtGraphViewFrame (Renderer *renderer, int graph_cache_size, SceneInt image_tile_width, SceneInt image_tile_height, QtGraphNeighbourhoodFrame *graph_neighbourhood_frame, QtGraphWidget *graph_widget)

  Constructs and initiates a graph view frame with the given renderer, graph image tile size and graph widget parent.
- int Init (const QString &x_unit_label_text, const QString &y_unit_label_text, bool toolbar_on_top)

  Initiats the labels, toolbar frame, render settings, the graph cache, the graphics scene, the graph axes and the tool graphics view.
- QtToolGraphicsView * tool_graphics_view ()

  Returns the tool graphics view used by the graphics scene of the graph.
- QPixmap graph_image ()

  The image displayed in the tool graphics view.
- QGraphicsPixmapItem * graph_glass_pane ()

  Returns the graph glass pane that can be used to display and draw anything on top of the graph view.
- QtGraphSettings * graph_settings ()

The graph settings.

- SceneRectF CurrentViewRectToScene ()

  The current viewport rectangle of the view in scene coordinates.
- GraphRectF CurrentViewRectToGraph ()

  The current viewport rectangle of the view in graph coordinates.
- GraphPointF CurrentCenterPosition ()

  The current graph view center position in graph coordinates.
- void LockGraphState ()

  Locks the graph state mutex.
- bool TryLockGraphState ()

  Attempts to lock the graph state mutex.
- bool TryLockGraphState (int timeout)

  Attempts to lock the graph state mutex within the given timeout.
- void UnlockGraphState ()

  Unlocks the graph state mutex.
- bool updating_graph ()

  The update state of the graph view.
- virtual void CacheObserverUpdate (CacheController *callee, CacheEvent reason)

  Recieves changes to the cache and renderer states.
- void PanTo (GraphPointF center_pos)

  Pans to (centers graph view on) the given position.
- void PanTriggeredUpdate ()

  Checks if the current center position is close enough to one edge of the scene to trigger a graph update.
- void UpdateDisplayedRangesInfo ()

  Updates the displayed ranges info in the coordinate and axes info frame.
- void ClearGraphImages ()

  Clears all graph image tiles used to compose the graph image.
- void SetGridProperties (QColor line_color, GraphDouble x_step, GraphDouble y_step, bool visible, QtGraphSettings::GridType type)

  Sets the grid line color, grid line distances grid visiblity and grid type.
- void ShowGrid ()

  Redraws the glass pane and shows the grid.
- void HideGrid ()

  Updates the glass pane and hides the grid.
- void SetAxesProperties (QColor line_color, const QString &x_label, const QString &y_label, GraphInt x_step, GraphInt y_step, GraphInt x_spacer, GraphInt y_spacer, double upper_sci_bound, double lower_sci_bound, int normal_precision, int sci_precision)

  Sets the axes line color, the x and y step values and spacers.
- void SetGraphImageTileWidth (int width)

  Sets the graph image tile width in the graph settings, and updates the graph image and scene sizes, as well as the graph state and view.
- void SetGraphImageTileHeight (int height)

  Sets the graph image tile height in the graph settings, and updates the graph image and scene sizes, as well as the graph state and view.
- void SetGraphImageTileSize (int width, int height)

  Sets the graph image tile size in the graph settings, and updates the graph image and scene sizes, as well as the graph state and view.
- void SetGraphImageTileRows (int rows)

  Sets the number of graph image tile rows and updates the view, graph state and graph settings.
- void SetGraphImageTileColumns (int columns)

  Sets the number of graph image tile columns and updates the view, graph state, and settings.
- void SetGraphImageTileRowsAndColumns (int rows, int columns)

Sets the number of graph image tile rows and columns, and updates the view, graph state and settings.

- SceneDouble CurrentSceneWidth ()

  Calculate the current graphics scene width.
- SceneDouble CurrentSceneHeight ()

  Calculate the current graphics scene height.
- SceneRectF CurrentSceneRect ()

  Calculates the current scene rectangle using CurrentSceneWidth() and CurrentSceneHeight().

Static Public Attributes

- static const SceneInt kDefaultPanWidth = 4

  The factor of graphics view widths, to render for the graph image to allow smoother panning and better neighbourhood overview.
- static const SceneInt kDefaultPanHeight = 4

  The factor of graphics view heights, to render for the graph image to allow smoother panning and better neighbourhood overview.
- static const int kDefaultGraphUpdateInterval = 1000

  The interval for the graph update timer in milliseconds.

Protected Member Functions

- void DrawGrid (QPainter &painter)

  Draws the grid according to the curent grid properties on the paint device associated with the painter.

Private Member Functions

- void CreateLabels ()
- void ClearGraphImagesDrawnFlags ()

  Clears all the drawn flags of the graph image tiles used to compose the graph image.
- void SwapStopReloadUpdateButtons (bool show_stop_update_button)

  Shows the stop update button and hides the reload update button if show_stop_update_button is true, otherwise the stop update button is hidden and the reload update button is shown.
- void DrawGraphFirstRedraw ()

  Takes care of updating the current graph state, the current view range data, glass pane and axes during the first graph update timer call to DrawGraphView().
- void DrawGraphDone ()

  Stops the graph update timer, updates the neighbourhood overview and resets the graph update parameters, progressbars and reload button.
- void DrawAxes ()

  Draws the x- and y-axis in the corresponding axis views.
- bool DrawAxesDashAndText (QPainter &painter, ScenePointF curr_dash_start_pos, ScenePointF curr_dash_stop_pos, QStaticText &curr_text, ScenePointF curr_text_start_pos, SceneInt curr_text_start, SceneInt curr_text_stop, SceneInt prev_text_stop, SceneInt next_text_start, SceneInt text_spacer)

  Tries to draw a dash and the given text at the given position on a graph axis, if there is enough available space (in pixels).
- QString AxesDashText (GraphDouble xy, double upper_sci_bound, double lower_sci_bound, int normal_precision, int sci_precision)

  Returns a formatted string of the given xy value with the given precision.
- SceneRectF PanTriggerUpdateBorder ()

  The border, outside which panning should trigger updates of the graph view and graph image, due to being too close to an edge of the graph image in the graphics scene.
- void SetGraphImageAndSceneSize (int width, int height)

Helper function to set the size of the graph image, graphics scene, graph axes and updates the next graph state.

- int GraphImagesRows ()

  The current number of rows in the graph image tiles 2D list.
- int GraphImagesColumns ()

  The current number of columns in the graph image tiles 2D list.
- void PrependRowToGraphImages ()

  Prepends a row containing NULLs to the graph image tiles 2D list.
- void AppendRowToGraphImages ()

  Appends a row containing NULLs to the graph image tiles 2D list.
- void PrependColumnToGraphImages ()

  Prepends a column containing NULLs to the graph image tiles 2D list.
- void AppendColumnToGraphImages ()

  Appends a column containing NULLs to the graph image tiles 2D list.
- void RemoveFirstGraphImagesRow ()

  Removes the first (top) row from the graph image tiles 2D list.
- void RemoveLastGraphImagesRow ()

  Removes the last (bottom) row from the graph image tiles 2D list.
- void RemoveFirstGraphImagesColumn ()

  Removes the first (leftmost) column from the graph image tiles 2D list.
- void RemoveLastGraphImagesColumn ()

  Removes the last column (rightmost) from the graph image tiles 2D list.

Private Attributes

- QtGraphSettings * graph_settings_

  Collection of all settings and other objects with specific configurations.
- QtGraphNeighbourhoodFrame * graph_neighbourhood_frame_

  The graph neighbourhood frame that shows the graph view neighbourhood and view outline.
- QtGraphWidget * graph_widget_

  The graph widget used to access the coordinate and axes info frame.
- QtToolGraphicsView * tool_graphics_view_

  The tool graphics view used by the graphics scene.
- QGraphicsView * x_axis_view_

  The graphics views used to show the x-axis.
- QGraphicsView * y_axis_view_

  The graphics views used to show the y-axis.
- QGraphicsPixmapItem * graph_pixmap_item_

  The graphics item used by the graphics scene to display the graph image.
- QGraphicsPixmapItem * x_axis_pixmap_item_

  The graphics item used by the graphics scene to display the x-axis.
- QGraphicsPixmapItem * y_axis_pixmap_item_

  The graphics item used by the graphics scene to display the y-axis.
- QGraphicsPixmapItem * graph_glass_pane_

  The graphics item used by the graphics scene to display the graph glass pane. The glass pane is an overlay used to display the grid, and it can also be used to display more things if desired.
- QLabel * x_plot_text_label_

  The label used to display x-axis name.
- QLabel * x_plot_arrow_label_

  The label used to display the x-axis arrow.
- QLabel * x_axis_unit_label_

  The label used to display the x-axis unit type.

- QLabel * y_plot_text_label_

  The label used to display y-axis name.
- QLabel * y_plot_arrow_label_

  The label used to display the y-axis arrow.
- QLabel * y_axis_unit_label_

  The label used to display the y-axis unit type.
- QPushButton * stop_update_button_

  The button used to stop an update and the rendering.
- QPushButton * reload_update_button_

  The button used to reload the current graph view.
- QPixmap graph_image_

  The image generated by the graph image tiles and used by the graphics pixmap item displayed in the graphics scene.
- QGridLayout * main_layout_

  The main layout containing all components of the object.
- QTimer * graph_update_timer_

  A timer used to progressively render partial graph image tile states on update.
- QMutex * graph_state_mutex_

  The mutex that makes sure that multiple, possibly parallel, graph state cache updates do not interfere with each other.
- bool cancel_update_

  Checked by graph updating functions to stop an update in progress.
- bool wait_for_renderer_

  UpdateFinished() does nothing as long as this is true.
- bool updating_graph_

  True if the graph view or renderer is currently updating the graph.
- bool draw_graph_init_

  Used once to force centering on the center of the graphics scene during initialization when the scene size is uninitialized.
- bool first_redraw_

  Used to update current graph state and recenter view only once per call to DrawGraphView().
- GraphImageTile2DList graph_image_tiles_

  A clipmap consisting of a 2D list coantaining the graph state image tiles, which are used to compose the currently shown graph image.

Additional Inherited Members

7.15.1 Detailed Description

A frame with the rendered plot graphics and axes.

This class acts as a clipmap using a graph image tile cache. It prepares and composes smaller graph image tiles into a larger graph image showing the plotted graph.

The composed graph image is larger than the actual view to allow panning. Panning to close to an edge or changing the scale or zoom level causes the graph image offset to change and progressively updates the graph image by adding graph image tiles along the closest edges and removing graph image tiles from the opposite edges to retain the size of the panning area. Graph image tiles are also added as needed to make sure that there is enough space to pan.

There are also functions for configuring a grid overlay and managing the graph axes.

Todo TODO(Max): this should be split into several classes and helper classes.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 ccdvl::frontend::QtGraphViewFrame::QtGraphViewFrame (Renderer * *renderer,* int *graph_cache_size,* SceneInt *image_tile_width,* SceneInt *image_tile_height,* QtGraphNeighbourhoodFrame * *graph_neighbourhood_frame,* QtGraphWidget * *graph_widget*)

Constructs and initiates a graph view frame with the given renderer, graph image tile size and graph widget parent.

Parameters

in	*renderer*	The renderer used to draw the graph.
in	*graph_cache_size*	The size of the graph image tile cache (the number of graph image tiles to hold).
in	*image_tile_width*	The width of a state image tile.
in	*image_tile_height*	The height of a state image tile.
in	*graph_neighbourhood_frame*	The graph neighbourhood frame.
in	*graph_widget*	The main widget used by the GUI.

7.15.3 Member Function Documentation

7.15.3.1 bool ccdvl::frontend::QtGraphViewFrame::AddGraphImageTilesOnResize ( ) [slot]

Expands the graph scene, if neccessary, by adding enough tiles to cover the current graph view.

This function should be called on resize of the graphics view and also if the state width or state height change.

Returns

true if the any tiles were added.

7.15.3.2 QString ccdvl::frontend::QtGraphViewFrame::AxesDashText (GraphDouble *xy,* double *upper_sci_bound,* double *lower_sci_bound,* int *normal_precision,* int *sci_precision*) [private]

Returns a formatted string of the given *xy* value with the given precision.

The returned string is an integer if the absolute value of *xy* is less than *upper_sci_bound* and more than *lower_sci_bound*, e.g. `AxesDashText(5123, 10000, 0.01)` returns *5123*.

Otherwise it is written with scientific notation with a line break after the float followed by times 10 with the exponent in superscript, e.g. *AxesDashText(51234, 10000, 0.01)* returns

5.123

×10^{+03}

and `AxesDashText(0.051234, 10000, 0.01)` returns

5.123

×10^{-02} .

Parameters

in	*xy*	The coordinate value to be converted.
in	*upper_sci_bound*	The upper bound over which scientific notation is to be used.
in	*lower_sci_bound*	The lower bound under which scientific notation is to be used.
in	*normal_precision*	The precision to be used for normal notation.

in	*sci_precision*	The precision to be used for the scientidic notation.

Returns

The value of *xy* as a formatted string.

7.15.3.3 void ccdvl::frontend::QtGraphViewFrame::BeginUpdate ( ) [signal]

Sending this signal indicates that an update has been started by the sender.

Note

Sending this signal does **not** start an update, use UpdateGraphView() for this.

See also

UpdateGraphView(), CancelUpdate() and FinishUpdate()

7.15.3.4 void ccdvl::frontend::QtGraphViewFrame::CacheObserverUpdate (CacheController * *callee*, CacheEvent *reason*) [virtual]

Recieves changes to the cache and renderer states.

The cache and renderer states that are handled are: rendering begun, rendering canceled and rendereing finished.

Parameters

in	*callee*	Currently ignored as there is only one ache controller.
in	*reason*	The current state change.

Implements ccdvl::CacheObserverInterface.

7.15.3.5 void ccdvl::frontend::QtGraphViewFrame::CancelUpdate ( ) [signal]

Sending this signal cancels any ongoing update of the graph view.

Note

This does not cancel any work being performed by the renderer.

See also

BeginUpdate() and FinishUpdate()

7.15.3.6 void ccdvl::frontend::QtGraphViewFrame::ClearGraphImages ( )

Clears all graph image tiles used to compose the graph image.

Deletes all the images and sets the drawn flags to false. This will cause all image tiles to be repainted from the graph cache buffers on the next graph view update, i.e. a call to UpdateGraph(), DrawGraph() or StartProgressive-GraphUpdates().

See also

UpdateGraph(), DrawGraph() and StartProgressiveGraphUpdates()

7.15.3.7 void ccdvl::frontend::QtGraphViewFrame::ClearGraphImagesDrawnFlags ( ) [private]

Clears all the *drawn* flags of the graph image tiles used to compose the graph image.

This will cause the graph image tiles to be redrawn on the next graph view update.

See also

ClearGraphImages()

7.15.3.8 void ccdvl::frontend::QtGraphViewFrame::CreateLabels ( ) [private]

Creates and initiates the labels of the graph view frame.

7.15.3.9 GraphPointF ccdvl::frontend::QtGraphViewFrame::CurrentCenterPosition ( )

The current graph view center position in graph coordinates.

Returns

The current center position of the graph view.

7.15.3.10 SceneDouble ccdvl::frontend::QtGraphViewFrame::CurrentSceneHeight ( )

Calculate the current graphics scene height.

The QGraphicsScene::height() may temporarily be invalid due to updates, resizes or transitions, and is thus more accurately calculated by GraphImagesRows() ∗ StateHeight().

Returns

The height of the graphics scene used by the tool graphics view.

See also

CurrentSceneWidth() and CurrentSceneRect()

7.15.3.11 SceneRectF ccdvl::frontend::QtGraphViewFrame::CurrentSceneRect ( )

Calculates the current scene rectangle using CurrentSceneWidth() and CurrentSceneHeight().

Returns

The scene rectangle of the graphics scene used by the tool graphics view.

See also

CurrentSceneWidth() and CurrentSceneHeight()

7.15.3.12 SceneDouble ccdvl::frontend::QtGraphViewFrame::CurrentSceneWidth ( )

Calculate the current graphics scene width.

The QGraphicsScene::width() may temporarily be invalid due to updates, resizes or transitions, and is thus more accurately calculated by GraphImagesColumns() ∗ StateWidth().

Returns

The width of the graphics scene used by the tool graphics view.

See also

CurrentSceneHeight() and CurrentSceneRect()

7.15.3.13 GraphRectF ccdvl::frontend::QtGraphViewFrame::CurrentViewRectToGraph ( )

The current viewport rectangle of the view in graph coordinates.

Returns

The currently shown graph rectangle.

7.15.3.14 SceneRectF ccdvl::frontend::QtGraphViewFrame::CurrentViewRectToScene ( )

The current viewport rectangle of the view in scene coordinates.

Returns

The currently shown scene rectangle.

7.15.3.15 bool ccdvl::frontend::QtGraphViewFrame::DrawAxesDashAndText (QPainter & *painter,* SceneDouble *curr_dash_start_pos,* SceneDouble *curr_dash_stop_pos,* QStaticText & *curr_text,* SceneDouble *curr_text_start_pos,* SceneInt *curr_text_start,* SceneInt *curr_text_stop,* SceneInt *prev_text_stop,* SceneInt *next_text_start,* SceneInt *text_spacer*) [private]

Tries to draw a dash and the given text at the given position on a graph axis, if there is enough available space (in pixels).

Parameters

in	*painter*	The painter to draw the dash and text with.
in	*curr_dash_start_pos*	Dash start position on axis.
in	*curr_dash_stop_pos*	Dash stop position on axis.
in	*curr_text*	The text to draw.
in	*curr_text_start_pos*	Text position in relation to dash.
in	*curr_text_start*	Text start position on axis.
in	*curr_text_stop*	Text stop position on axis.
in	*prev_text_stop*	Previous text stop position on axis.
in	*next_text_start*	Next text start position on axis.
in	*text_spacer*	Minimum pixel distance between texts on axis.

Returns

true if the dash and text was drawn.

7.15.3.16 void ccdvl::frontend::QtGraphViewFrame::DrawGraphFirstRedraw ( ) [private]

Takes care of updating the current graph state, the current view range data, glass pane and axes during the first graph update timer call to DrawGraphView().

See also

DrawGraphView()

7.15.3.17 void ccdvl::frontend::QtGraphViewFrame::DrawGraphView ( ) [slot]

Progressively draws and shows the graph image composed of the graph image tiles fetched from the graph cache state image buffers.

It should continue to be called by the graph update timer, normally started by a call to UpdateGraphView(). It will then stop the graph update timer when all graph image tile graph cache states are valid (fully rendered).

Note

This function should thus preferably not be called directly when the graph update timer is not running, as it will potentially leave the dispalyed graph in a partially updated state if the graph cache does not have completely rendered graph image tiles for the current view.

See also

UpdateGraphView()

7.15.3.18 void ccdvl::frontend::QtGraphViewFrame::DrawGrid (QPainter & *painter*) [protected]

Draws the grid according to the curent grid properties on the paint device associated with the *painter*.

It uses #next_graph_scene_state_ to determine the position of the grid lines. It first draws all grid lines with coordinates larger than zero, and then all the grid lines with coordinates less than zero.

Vertical lines are drawn first and horizontal lines last.

Parameters

in	*painter*	The painter to draw the grid with.

See also

SetGridProperties(QColor, GraphDouble, GraphDouble, bool, GridType)

7.15.3.19 void ccdvl::frontend::QtGraphViewFrame::FinishUpdate ( ) [signal]

Sending this signal indicates that the current ongoing update is done and that the GUI should reflect this.

Note

This does nothing if the renderer process is in the progress of being canceled/stopped/aborted.

See also

BeginUpdate() and CancelUpdate()

7.15.3.20 QGraphicsPixmapItem * ccdvl::frontend::QtGraphViewFrame::graph_glass_pane ( )

Returns the graph glass pane that can be used to display and draw anything on top of the graph view.

By default the graph glass pane is only used to display the grid. Override UpdateGraphGlassPane() to change what is shown.

Returns

The graph glass pane QGraphicsPixmapItem.

See also

UpdateGraphGlassPane()

7.15.3.21 QPixmap ccdvl::frontend::QtGraphViewFrame::graph_image ( )

The image displayed in the tool graphics view.

Returns

The graph image as a pixmap.

7.15.3.22 QtGraphSettings * ccdvl::frontend::QtGraphViewFrame::graph_settings ( )

The graph settings.

Returns

The graph settings.

7.15.3.23 int ccdvl::frontend::QtGraphViewFrame::GraphImagesColumns ( ) [private]

The current number of columns in the graph image tiles 2D list.

Returns

The number of columns of graph tile images.

7.15.3.24 int ccdvl::frontend::QtGraphViewFrame::GraphImagesRows ( ) [private]

The current number of rows in the graph image tiles 2D list.

Returns

The number of rows of graph tile images.

7.15.3.25 void ccdvl::frontend::QtGraphViewFrame::HideGrid ( )

Updates the glass pane and hides the grid.

See also

ShowGrid() and UpdateGraphGlassPane()

7.15.3.26 int ccdvl::frontend::QtGraphViewFrame::Init (const QString & *x_unit_label_text*, const QString & *y_unit_label_text*, bool *toolbar_on_top*)

Initiats the labels, toolbar frame, render settings, the graph cache, the graphics scene, the graph axes and the tool graphics view.

Parameters

in	*x_unit_label_text*	The label for the x-axis unit.
in	*y_unit_label_text*	The label for the y-axis unit.
	toolbar_on_top	Puts the toolbar frame on top of the graph view if true.

Returns

0 if the initializiation was successful.

7.15.3.27 void ccdvl::frontend::QtGraphViewFrame::LockGraphState ( )

Locks the graph state mutex.

If another thread has locked the graph state mutex then this call will block until the graph state mutex is unlocked by that thread with UnlockGraphState().

Note

This function must be called before using the current graph state in any way, otherwise the state is not thread safe and it will have undefined behaviour.

See also

TryLockGraphState(), UnlockGraphState() and current_graph_state().

7.15.3.28 void ccdvl::frontend::QtGraphViewFrame::PanTo (GraphPointF *center_pos*)

Pans to (centers graph view on) the given position.

Calls UpdateGraph(GraphPointF) if *center_pos* is outside the current graph view scene.

Parameters

in	*center_pos*	The position to recenter on.

7.15.3.29 void ccdvl::frontend::QtGraphViewFrame::PanTriggeredUpdate ( )

Checks if the current center position is close enough to one edge of the scene to trigger a graph update.

Updates the graph view by calling UpdateGraph() if the center position is outside the rectangle formed by the

following points:

top left = scene width / tile columns, scene height / tile rows

bottom right = scene witdh ∗ (tile columns - 1) / tile columns, scene height ∗ (tile rows - 1) / tile rows

See also

UpdateGraph()

7.15.3.30 SceneRectF ccdvl::frontend::QtGraphViewFrame::PanTriggerUpdateBorder ( ) [private]

The border, outside which panning should trigger updates of the graph view and graph image, due to being too close to an edge of the graph image in the graphics scene.

The border rectangle is dependent on the current size of the graphics view in that it is formed by subtracting 1.5 ∗ width and 1.5 ∗ height of the graphics view from the width and height of the current scene rectangle respectively.

Returns

The current pan trigger update border rectangle.

7.15.3.31 void ccdvl::frontend::QtGraphViewFrame::ReloadUpdateButtonClicked ( ) [slot]

Reloads the graph view with the current graph state properties.

See also

current_graph_state() and StopUpdateButtonClicked()

7.15.3.32 void ccdvl::frontend::QtGraphViewFrame::SetAxesProperties (QColor *line_color,* const QString & *x_label,* const QString & *y_label,* GraphInt *x_step,* GraphInt *y_step,* GraphInt *x_spacer,* GraphInt *y_spacer,* double *upper_sci_bound,* double *lower_sci_bound,* int *normal_precision,* int *sci_precision*)

Sets the axes line color, the x and y step values and spacers.

DrawAxes() must be called for the changes to be applied.

Parameters

in	*line_color*	The new axes line color.
in	*x_label*	The x-axis unit label.
in	*y_label*	The y-axis unit label.
in	*x_step*	The new graph step distance along the x-axis.
in	*y_step*	The new graph step distance along the y-axis.
in	*x_spacer*	The scene spacer distance between steps and sub-steps along the x-axis.
in	*y_spacer*	The scene spacer distance between steps and sub-steps along the y-axis.
in	*upper_sci_bound*	The upper bound over which scientific notation is to be used.
in	*lower_sci_bound*	The lower bound under which scientific notation is to be used.
in	*normal_precision*	The precision to be used for normal notation.
in	*sci_precision*	The precision to be used for the scientidic notation.

See also

DrawAxes()

7.15.3.33 void ccdvl::frontend::QtGraphViewFrame::SetGraphImageTileColumns (int *columns*)

Sets the number of graph image tile columns and updates the view, graph state, and settings.

More tiles will automatically be added by the graph update to cover the view and panning space if needed.

Parameters

in	*columns*	The number of graph image tiles columns to use. Must be > 2.

See also

SetGraphImageTileRows(int) and SetGraphImageTileRowsAndColumns(int, int)

7.15.3.34 void ccdvl::frontend::QtGraphViewFrame::SetGraphImageTileHeight (int *height*)

Sets the graph image tile height in the graph settings, and updates the graph image and scene sizes, as well as the graph state and view.

Parameters

in	*height*	The new graph image tile height. Must be > 0.

See also

SetGraphImageTileWidth(int) and SetGraphImageTileSize(int, int)

7.15.3.35 void ccdvl::frontend::QtGraphViewFrame::SetGraphImageTileRows (int *rows*)

Sets the number of graph image tile rows and updates the view, graph state and graph settings.

More tiles will automatically be added by the graph update to cover the view and panning space if needed.

Parameters

in	*rows*	The number of graph image tiles rows to use. Must be > 2.

See also

SetGraphImageTileColumns(int) and SetGraphImageTileRowsAndColumns(int, int)

7.15.3.36 void ccdvl::frontend::QtGraphViewFrame::SetGraphImageTileRowsAndColumns (int *rows,* int *columns*)

Sets the number of graph image tile rows and columns, and updates the view, graph state and settings.

More tiles will automatically be added by the graph update to cover the view and panning space if needed.

Parameters

in	*rows*	The number of graph image tiles rows to use. Must be > 2.
in	*columns*	The number of graph image tiles columns to use. Must be > 2.

See also

SetGraphImageTileRows(int) and SetGraphImageTileColumns(int)

7.15.3.37 void ccdvl::frontend::QtGraphViewFrame::SetGraphImageTileSize (int *width*, int *height*)

Sets the graph image tile size in the graph settings, and updates the graph image and scene sizes, as well as the graph state and view.

Parameters

in	*width*	The new graph image tile width. Must be > 0.
in	*height*	The new graph image tile height. Must be > 0.

See also

SetGraphImageTileWidth(int) and SetGraphImageTileHeight(int)

7.15.3.38 void ccdvl::frontend::QtGraphViewFrame::SetGraphImageTileWidth (int *width*)

Sets the graph image tile width in the graph settings, and updates the graph image and scene sizes, as well as the graph state and view.

Parameters

in	*width*	The new graph image tile width. Must be > 0.

See also

SetGraphImageTileHeight(int) and SetGraphImageTileSize(int, int)

7.15.3.39 void ccdvl::frontend::QtGraphViewFrame::SetGridProperties (QColor *line_color*, GraphDouble *x_step*, GraphDouble *y_step*, bool *visible*, QtGraphSettings::GridType *type*)

Sets the grid line color, grid line distances grid visiblity and grid type.

UpdateGraphGlassPane() must be called for the changes to be applied.

Parameters

in	*line_color*	The new grid line color.
in	*x_step*	The new grid line distance along the x-axis.
in	*y_step*	The new grid line distance along the y-axis.
in	*visible*	Determines if the grid should be shown or not.
in	*type*	The type of the grid.

See also

UpdateGraphGlassPane()

7.15.3.40 void ccdvl::frontend::QtGraphViewFrame::SettingsChanged (bool *renderer*, bool *graph*, bool *grid*, bool *axes*) [slot]

Handles changes of the graph configuration class QtSettings.

Parameters

renderer	true iff any of the renderer settings changed. This includes point shape, line, scaling, etc.
graph	true iff any graph view settings changed.
grid	true iff any grid settings changed.
axes	true iff any axis setting changed.

7.15.3.41 void ccdvl::frontend::QtGraphViewFrame::ShowGrid ( )

Redraws the glass pane and shows the grid.

See also

HideGrid() and UpdateGraphGlassPane()

7.15.3.42 void ccdvl::frontend::QtGraphViewFrame::StartProgressiveGraphUpdates ( ) [slot]

Starts the graph update timer that triggers progressive graph updates using the kDefaultGraphUpdateInterval.

See also

StartProgressiveGraphUpdates(int) and kDefaultGraphUpdateInterval

7.15.3.43 void ccdvl::frontend::QtGraphViewFrame::StartProgressiveGraphUpdates (int *update_interval*) [slot]

Starts or restarts the graph update timer that triggers progressive graph updates using the given *update_interval*.

Parameters

in	*update_interval*	The update interval to be used, which must be larger than zero.

See also

StartProgressiveGraphUpdates()

7.15.3.44 void ccdvl::frontend::QtGraphViewFrame::StopUpdateButtonClicked ( ) [slot]

Stops the current graph view update and the renderer if it is running.

Note

This may cause the shown and the current graph scene state to become out of sync. A reload or update of the graph is required to make sure that they are in sync again.

See also

current_graph_state(), UpdateGraphView() and ReloadUpdateButtonClicked()

7.15.3.45 QtToolGraphicsView * ccdvl::frontend::QtGraphViewFrame::tool_graphics_view ( )

Returns the tool graphics view used by the graphics scene of the graph.

Returns

The tool graphics view.

7.15.3.46 bool ccdvl::frontend::QtGraphViewFrame::TryLockGraphState ( )

Attempts to lock the graph state mutex.

If the lock was obtained, the it must be unlocked with UnlockGraphState() before another thread can successfully lock it.

Returns

true if the lock was obtained. If another thread has locked the graph state, this function returns false immediately.

See also

TryLockGraphState(int), LockGraphState(), UnlockGraphState() and current_graph_state().

7.15.3.47 bool ccdvl::frontend::QtGraphViewFrame::TryLockGraphState (int *timeout*)

Attempts to lock the graph state mutex within the given *timeout*.

If another thread has locked the graph state mutex, this function will wait for at most *timeout* milliseconds for the graph state mutex to become available.

If it was obtained, the graph state mutex must be unlocked with UnlockGraphState() before another thread can successfully lock it.

Note

Passing a negative number as the timeout is equivalent to calling LockGraphState(), i.e. this function will wait forever until the graph state mutex can be locked if timeout is negative.

Parameters

in	*timeout*	The number of milliseconds to wait for the graph state mutex to become available.

Returns

true if the lock was successfully locked by this function within the given *timeout*.

See also

LockGraphState(), UnlockGraphState() and current_graph_state().

7.15.3.48 void ccdvl::frontend::QtGraphViewFrame::UnlockGraphState ( )

Unlocks the graph state mutex.

Attempting to unlock the graph state mutex in a different thread to the one that locked it results in an error. Unlocking the graph state mutex when it is not locked results in undefined behavior.

See also

LockGraphState(), TryLockGraphState() and current_graph_state().

7.15.3.49 void ccdvl::frontend::QtGraphViewFrame::UpdateBegun ( ) [slot]

Sets the stop and reload button properties when an update begins.

See also

UpdateCanceled() and UpdateFinished()

7.15.3.50 void ccdvl::frontend::QtGraphViewFrame::UpdateCanceled ( ) [slot]

Sets the stop and reload button properties when an update is canceled, and stops the graph view update.

See also

UpdateBegun() and UpdateFinished()

7.15.3.51 void ccdvl::frontend::QtGraphViewFrame::UpdateFinished ( ) [slot]

Sets the stop and reload button properties when an update finishes.

See also

UpdateBegun and UpdateCanceled()

7.15.3.52 void ccdvl::frontend::QtGraphViewFrame::UpdateGraphGlassPane ( ) [slot]

Updates the graph glass pane and redraws the grid on it.

This function can be overriden in a sub class to allow more items than the grid to be shown on the glass pane.

See also

DrawGrid(QPainter&)

7.15.3.53 void ccdvl::frontend::QtGraphViewFrame::UpdateGraphView ( ) [slot]

Updates the graph states in the graph cache used by the graph image tiles with the current center position, zoom and scale factors. Progressive graph updates are also started if needed.

See also

UpdateGraph(GraphPointF), UpdateGraph(GraphPointF, double, double) and UpdateGraph(GraphPointF, double, double, double, double).

7.15.3.54 void ccdvl::frontend::QtGraphViewFrame::UpdateGraphView (GraphPointF *center_pos*) [slot]

Updates the graph states in the graph cache used by the graph image tiles with the given center position and the current zoom and scale factors. Progressive graph updates are also started if needed.

Parameters

in	*center_pos*	The new center position to recenter on.

See also

UpdateGraph(), UpdateGraph(GraphPointF, double, double) and UpdateGraph(GraphPointF, double, double, double, double).

7.15.3.55 void ccdvl::frontend::QtGraphViewFrame::UpdateGraphView (GraphPointF *center_pos,* double *x_zoom,* double *y_zoom*) [slot]

Updates the graph states in the graph cache used by the graph image tiles with the given center position, zoom and current scale factors. Progressive graph updates are also started if needed.

Parameters

in	*center_pos*	The new center position to recenter on.
in	*x_zoom*	The x (or width) zoom factor of the graph view.
in	*y_zoom*	The y (or height) zoom factor of the graph view.

See also

UpdateGraph(), UpdateGraph(GraphPointF) and UpdateGraph(GraphPointF, double, double, double, double).

7.15.3.56 void ccdvl::frontend::QtGraphViewFrame::UpdateGraphView (GraphPointF *center_pos,* double *x_zoom,* double *y_zoom,* double *x_scale,* double *y_scale*) [slot]

Updates the graph states in the graph cache used by the graph image tiles with the given center position, zoom and scale factors. Progressive graph updates are also started if needed.

Parameters

in	*center_pos*	The new center position to recenter on.
in	*x_zoom*	The x (or width) zoom factor of the graph view.
in	*y_zoom*	The y (or height) zoom factor of the graph view.
in	*x_scale*	The x (or width) scale factor of the graph view.
in	*y_scale*	The y (or height) scale factor of the graph view.

See also

UpdateGraph(), UpdateGraph(GraphPointF) and UpdateGraph(GraphPointF, double, double, double, double).

7.15.3.57 bool ccdvl::frontend::QtGraphViewFrame::updating_graph ( )

The update state of the graph view.

Returns

true if the graph view is being updated.

7.15.4 Member Data Documentation

7.15.4.1 bool ccdvl::frontend::QtGraphViewFrame::cancel_update_ [private]

Checked by graph updating functions to stop an update in progress.

See also

UpdateGraphView(), DrawGraphView(), CancelUpdate(), StopUpdateButtonClicked() and ReloadUpdateButtonClicked()

7.15.4.2 QGraphicsPixmapItem* ccdvl::frontend::QtGraphViewFrame::graph_glass_pane_ [private]

The graphics item used by the graphics scene to display the graph glass pane. The glass pane is an overlay used to display the grid, and it can also be used to display more things if desired.

See also

graph_glass_pane() and UpdateGraphGlassPane()

7.15.4.3 GraphImageTile2DList ccdvl::frontend::QtGraphViewFrame::graph_image_tiles_ [private]

A clipmap consisting of a 2D list coantaining the graph state image tiles, which are used to compose the currently shown graph image.

The list contains rows, with each row represented by a list. *begin()->begin()* thus returns an iterator for the values in the first row, and so on.

See also

GraphImagesRows(), GraphImagesColumns(), SetGraphImageTileRows(int), SetGraphImageTileColumns(int), SetGraphImageTileRowsAndColumns(int, int), PrependRowToGraphImages(), AppendRowToGraphImages(), PrependColumnToGraphImages(), AppendColumnToGraphImages(), RemoveFirstGraphImagesRow(), RemoveLastGraphImagesRow(), RemoveFirstGraphImagesColumn() and RemoveLastGraphImagesColumn().

7.15.4.4 QGraphicsPixmapItem* ccdvl::frontend::QtGraphViewFrame::graph_pixmap_item_ [private]

The graphics item used by the graphics scene to display the graph image.

See also

graph_image_

7.15.4.5 QMutex* ccdvl::frontend::QtGraphViewFrame::graph_state_mutex_ [private]

The mutex that makes sure that multiple, possibly parallel, graph state cache updates do not interfere with each other.

Must be used when dealing with the graph cache or any kind of graph state. This makes sure that the #current_graph_scene_state_ remains valid and in sync with the displayed graph image during update.

See also

#current_graph_scene_state_, LockGraphState(), TryLockGraphState(), TryLockGraphState(int) and UnlockGraphState()

7.15.4.6 QTimer* ccdvl::frontend::QtGraphViewFrame::graph_update_timer_ [private]

A timer used to progressively render partial graph image tile states on update.

Started by Init() and UpdateGraphView(). Stopped by DrawGraphView() when all image tile states are valid (fully rendered).

See also

Init(), UpdateGraphView() and DrawGraphView()

7.15.4.7 const int ccdvl::frontend::QtGraphViewFrame::kDefaultGraphUpdateInterval = 1000 [static]

The interval for the graph update timer in milliseconds.

Setting this too low will slow down the response time of the interface, especially on computers with a slow or single core processor. Setting this too high will make the graph updates real slow, but might lessen any potential jerkiness of the GUI.

See also

graph_update_timer_ and StartProgressiveGraphUpdates()

7.15.4.8 const SceneInt ccdvl::frontend::QtGraphViewFrame::kDefaultPanHeight = 4 [static]

The factor of graphics view heights, to render for the graph image to allow smoother panning and better neighbourhood overview.

Note

This must be $>$ 2 to work at all, but it is not usefull unless $>$ 3.

7.15.4.9 const SceneInt ccdvl::frontend::QtGraphViewFrame::kDefaultPanWidth = 4 [static]

The factor of graphics view widths, to render for the graph image to allow smoother panning and better neighbourhood overview.

Note

This must be $>$ 2 to work at all, but it is not usefull unless $>$ 3.

7.15.4.10 QPushButton* ccdvl::frontend::QtGraphViewFrame::reload_update_button_ [private]

The button used to reload the current graph view.

See also

ReloadUpdateButtonClicked()

7.15.4.11 QPushButton* ccdvl::frontend::QtGraphViewFrame::stop_update_button_ [private]

The button used to stop an update and the rendering.

See also

StopUpdateButtonClicked()

7.15.4.12 bool ccdvl::frontend::QtGraphViewFrame::wait_for_renderer_ [private]

UpdateFinished() does nothing as long as this is true.

Is set to true by StopUpdateButtonClicked() if the renderer is not idle at the time. Is set to false when CacheObserverUpdate() recieves a kRendererCanceled message, which also triggers a call to UpdateFinished().

See also

UpdateFinished(), StopUpdateButtonClicked() and CacheObserverUpdate()

The documentation for this class was generated from the following files:

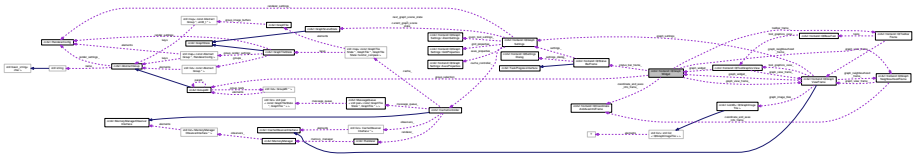
- include/qt_frontend/qt_graph_view_frame.h
- src/qt_frontend/qt_graph_view_frame.cc

7.16 ccdvl::frontend::QtGraphWidget Class Reference

The main widget for the graph GUI, responsible for creating, displaying and updating the GUI components.

`#include <qt_graph_widget.h>`

Collaboration diagram for ccdvl::frontend::QtGraphWidget:



Public Slots

- void show ()

  Displays a loading data message when showing the window.
- void ResizeGraphUpdate ()

  Checks if the graph has to be updated due to resize events.

Public Member Functions

- QtGraphWidget (Renderer *renderer, bool mac_brushed_metal_look, QWidget *parent=NULL)

  Constructs and initiates a graph widget with the given width, height, renderer and parent.
- int Init (const QString &window_title, const QString &x_unit_label_text, const QString &y_unit_label_text, int width, int height, bool toolbar_on_top)

  Initiates the GUI components and resize event timer.
- QtGraphViewFrame * graph_view_frame ()

  Returns the graph view frame of the widget.
- QtCoordinateAndAxesInfoFrame * coordinate_and_axes_info_frame ()

  Returns the coordinate and axes info frame of the widget.
- QtGraphNeighbourhoodFrame * neighbourhood_frame ()

  Returns the neighbourhood frame of the widget.
- QtStatusBarFrame * status_bar_frame ()

  Returns the status bar frame of the widget.
- QtToolbarFrame * toolbar_frame ()

Returns the toolbar frame of the widget.

- Group2D * GroupSelection ()

  Get selection as a group.

Protected Member Functions

- virtual bool event (QEvent *event)

  Accepts status tip events and displays the status tips in the satus bar of the status bar frame. All other events are ignored and passed on to the parent of the widget.
- virtual void resizeEvent (QResizeEvent *event)

  Accepts and handles resize events.

Private Attributes

- bool init_done_

  Disables multiple graph state updates during initiation.
- Group2D * group_selection_

  Current group.
- QTimer * resize_event_graph_update_timer_

  The resize event timer used to delay graph updates.
- QtGraphViewFrame * graph_view_frame_

  The graph view frame of the widget.
- QtCoordinateAndAxesInfoFrame * coordinate_and_axes_info_frame_

  The coordinate and axes info frame of the widget.
- QtGraphNeighbourhoodFrame * graph_neighbourhood_frame_
- QtStatusBarFrame * status_bar_frame_

  The status bar frame of the widget.
- QtToolbarFrame * toolbar_frame_

  The toolbar frame of the widget.
- QHBoxLayout * main_layout_

  The main layout containing all components of the widget.

Related Functions

(Note that these are not member functions.)

- void CCDVLQtInitializeResources ()

  Forces the resources (in a static library) to be initiated. See `http://doc.qt.nokia.com/latest/resources.-html` *for more information.*

7.16.1 Detailed Description

The main widget for the graph GUI, responsible for creating, displaying and updating the GUI components.

See also

`QWidget`

7.16.2 Constructor & Destructor Documentation

7.16.2.1 ccdvl::frontend::QtGraphWidget::QtGraphWidget (Renderer * *renderer,* bool *mac_brushed_metal_look,* QWidget * *parent* = NULL)

Constructs and initiates a graph widget with the given *width*, *height*, *renderer* and *parent*.

The *mac_brushed_metal_look* decides if the widget window should have the native brushed metal look when running on Mac OS X. This is not supported by other operating systems.

Parameters

in	*renderer*	The renderer to be usded by the main widget.
in	*mac_brushed_metal_look*	Brushed metal look on Mac OS X.
in	*parent*	The parent widget.

7.16.3 Member Function Documentation

7.16.3.1 QtCoordinateAndAxesInfoFrame * ccdvl::frontend::QtGraphWidget::coordinate_and_axes_info_frame ( )

Returns the coordinate and axes info frame of the widget.

Returns

The coordinate and axes info frame.

7.16.3.2 bool ccdvl::frontend::QtGraphWidget::event (QEvent * *event*) [protected], [virtual]

Accepts status tip events and displays the status tips in the satus bar of the status bar frame. All other events are ignored and passed on to the parent of the widget.

Parameters

in	*event*	An event.

7.16.3.3 QtGraphViewFrame * ccdvl::frontend::QtGraphWidget::graph_view_frame ( )

Returns the graph view frame of the widget.

Returns

The graph view frame.

7.16.3.4 Group2D * ccdvl::frontend::QtGraphWidget::GroupSelection ( )

Get selection as a group.

Obtain the one and only supported group.

Returns

A group or NULL if there is no selection.

7.16.3.5 int ccdvl::frontend::QtGraphWidget::Init (const QString & *window_title,* const QString & *x_unit_label_text,* const QString & *y_unit_label_text,* int *width,* int *height,* bool *toolbar_on_top*)

Initiates the GUI components and resize event timer.

Parameters

in	*window_title*	The title of the widget window.
in	*x_unit_label_text*	The label for the x-axis unit.
in	*y_unit_label_text*	The label for the y-axis unit.
in	*width*	Initial main widget width.
in	*height*	Initial main widget height.
in	*toolbar_on_top*	Decides if the toolbar frame should be on top of the graph view or not.

Returns

0 if the initializiation was successfull.

7.16.3.6 QtGraphNeighbourhoodFrame* ccdvl::frontend::QtGraphWidget::neighbourhood_frame ( )

Returns the neighbourhood frame of the widget.

Returns

The neighbourhood frame.

7.16.3.7 void ccdvl::frontend::QtGraphWidget::resizeEvent (QResizeEvent * *event*) [protected], [virtual]

Accepts and handles resize events.

Delays the update of the graph view image to 0.5 seconds after the last recieved resize event to avoid unnecessary repaints and heavy recalculations of the image while resizing the widget window. Multiple resize events may be triggered during the resize depending on the behaviour of the OS window manager (at least Mac OS X does this).

Parameters

event	A resize event.

7.16.3.8 void ccdvl::frontend::QtGraphWidget::show ( ) [slot]

Displays a loading data message when showing the window.

See also

QWidget::show()

7.16.3.9 QtStatusBarFrame * ccdvl::frontend::QtGraphWidget::status_bar_frame ( )

Returns the status bar frame of the widget.

Returns

The status bar frame.

7.16.3.10 QtToolbarFrame * ccdvl::frontend::QtGraphWidget::toolbar_frame ( )

Returns the toolbar frame of the widget.

Returns

The toolbar frame.

7.16.4 Member Data Documentation

7.16.4.1 QtGraphNeighbourhoodFrame* ccdvl::frontend::QtGraphWidget::graph_neighbourhood_frame_ [private]

The neighbourhood frame of the widget.

The documentation for this class was generated from the following files:

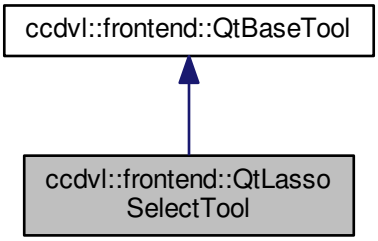
- include/qt_frontend/qt_graph_widget.h
- src/qt_frontend/qt_graph_widget.cc

7.17 ccdvl::frontend::QtLassoSelectTool Class Reference

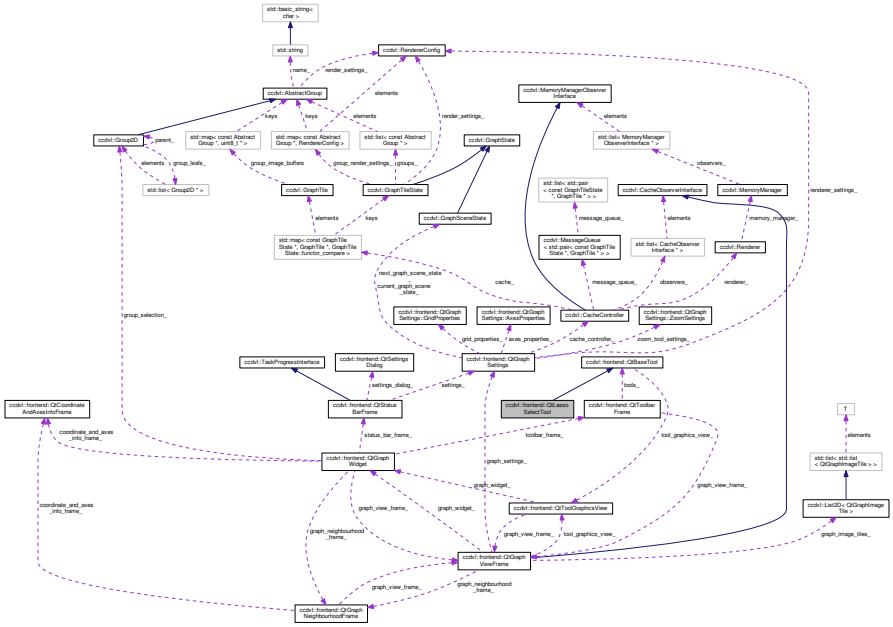
Provides a reusable lasso select rubber band for the graphics scene of the given QtToolGraphicsView.

`#include <qt_lasso_select_tool.h>`

Inheritance diagram for ccdvl::frontend::QtLassoSelectTool:



Collaboration diagram for ccdvl::frontend::QtLassoSelectTool:



Public Member Functions

- QtLassoSelectTool (QtToolGraphicsView *tool_graphics_view)

  Constructs and initiates a deactivated lasso select tool object with the given QtToolGraphicsView and image pointer.
- virtual bool OnMousePress (QMouseEvent *event)

  Starts a new lasso selection.
- virtual bool OnMouseMove (QMouseEvent *event)

  Adds a line between the previous position and the position given by the mouse event, and changes the mouse cursor depending in the cursor background.
- virtual bool OnMouseRelease (QMouseEvent *event)

  Deactivates and hides the lasso selection rubber band on left mouse button release if it is active.
  Other mouse events are ignored.
- virtual bool OnWheel (QWheelEvent *event)

  Accepts all wheel events to disable manual scrolling.

Public Attributes

- const QCursor kLassoCursor
- const QCursor kLassoCursorInverted

Static Public Attributes

- static const int kMinLineLength = 4

  The minimum polygon line segment length in pixels.

Private Member Functions

- void UpdatePolygonPainterPath ()

Updates the polygon painter path used to display the lasso rubber band selection.

- void UpdateViewImage ()

  Updates the view_image_ if necessary.

- QColor ViewPixelColor (ViewPoint point)

  Does a color lookup of the pixel at the given point.

Private Attributes

- bool active_

  Keeps track of if the lasso select tool is active.

- QGraphicsPolygonItem * graphics_polygon_item_

  The graphics polygon item used to display the lasso selection in the graphics scene.

- QPainterPath lasso_painter_path_

  The current selection consisting of a series of connected lines.

- QPainterPath polygon_painter_path_

  The final selection consisting of a series of connected lines.

- SceneRect prev_view_scene_rect_

  The previous graphics view rectangle, in scene coordinates, used in combination with prev_view_graph_rect_ to determine if the view of the graphics view has changed.

- GraphRect prev_view_graph_rect_

  The previous graphics view rectangle, in graph coordinates, used in combination with prev_view_scene_rect_ to determine if the view of the graphics view has changed.

- QImage view_image_

  The current portion of the graphics scene as shown by the graphics view, represented as an image used for pixel color lookup.

Additional Inherited Members

7.17.1 Detailed Description

Provides a reusable lasso select rubber band for the graphics scene of the given QtToolGraphicsView.

The mouse cursor icon color inverts when going to and from lighter and darker background in the graph image, to allow better visability.

The rubber band is used in conjunction with mouse, wheel and key events.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 ccdvl::frontend::QtLassoSelectTool::QtLassoSelectTool (QtToolGraphicsView * *tool_graphics_view*)

Constructs and initiates a deactivated lasso select tool object with the given QtToolGraphicsView and image pointer.

Parameters

in	*tool_graphics_view*	The tool graphics view that uses the lasso select tool.

7.17.3 Member Function Documentation

7.17.3.1 bool ccdvl::frontend::QtLassoSelectTool::OnMouseMove (QMouseEvent * *event*) [virtual]

Adds a line between the previous position and the position given by the mouse event, and changes the mouse cursor depending in the cursor background.

Only triggers if the lasso select tool is active, the left mouse button is pressed (other mouse events are ignored) and if the length of the added line would be greater than QtLassoSelectTool::kMinLineLength.

It also always sets the cursor to either kLassoCursor or kLassoCursorInverted depending on the color of the pixel below the cursor.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.17.3.2 bool ccdvl::frontend::QtLassoSelectTool::OnMousePress (QMouseEvent * *event*) [virtual]

Starts a new lasso selection.

Only triggers on left mouse press and uses the position given by the mouse event, activates and shows the lasso select rubber band.

Right mouse press cancels the selection and deactivates the lasso select rubber band.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMouseMove(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.17.3.3 bool ccdvl::frontend::QtLassoSelectTool::OnMouseRelease (QMouseEvent * *event*) [virtual]

Deactivates and hides the lasso selection rubber band on left mouse button release if it is active.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseMove(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.17.3.4 bool ccdvl::frontend::QtLassoSelectTool::OnWheel (QWheelEvent * *event*) [virtual]

Accepts all wheel events to disable manual scrolling.

Parameters

in	*event*	The wheel event.

Returns

true if the event was accepted.

Reimplemented from ccdvl::frontend::QtBaseTool.

7.17.3.5 QColor ccdvl::frontend::QtLassoSelectTool::ViewPixelColor (ViewPoint *point*) [private]

Does a color lookup of the pixel at the given *point*.

Parameters

in	*point*	The pixel position to lookup.

Returns

The color of the pixel if it is valid, otherwise an invalid color.

7.17.4 Member Data Documentation

7.17.4.1 const QCursor ccdvl::frontend::QtLassoSelectTool::kLassoCursor

The normal (black) lasso cursor used over lighter areas.

7.17.4.2 const QCursor ccdvl::frontend::QtLassoSelectTool::kLassoCursorInverted

The inverted (white) lasso cursor used over darker areas.

7.17.4.3 const int ccdvl::frontend::QtLassoSelectTool::kMinLineLength = 4 [static]

The minimum polygon line segment length in pixels.

This is used to decrease the number of lines making up the lasso selection polygon. Lower values make a smoother polygon with a slower update rate and higher values make a sharper polygon with a faster update rate.

7.17.4.4 GraphRect ccdvl::frontend::QtLassoSelectTool::prev_view_graph_rect_ [private]

The previous graphics view rectangle, in graph coordinates, used in combination with prev_view_scene_rect_ to determine if the view of the graphics view has changed.

See also

prev_view_scene_rect_ and OnMouseMove(QMouseEvent*)

7.17.4.5 SceneRect ccdvl::frontend::QtLassoSelectTool::prev_view_scene_rect_ [private]

The previous graphics view rectangle, in scene coordinates, used in combination with prev_view_graph_rect_ to determine if the view of the graphics view has changed.

See also

prev_view_graph_rect_ and OnMouseMove(QMouseEvent*)

7.17.4.6 QImage ccdvl::frontend::QtLassoSelectTool::view_image_ [private]

The current portion of the graphics scene as shown by the graphics view, represented as an image used for pixel color lookup.

See also

ViewPixelColor(ViewPoint) and OnMouseMove(QMouseEvent*)

The documentation for this class was generated from the following files:

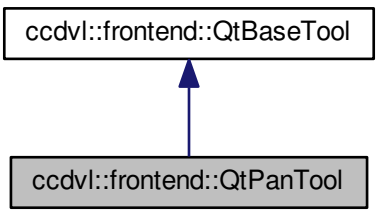
- include/qt_frontend/qt_lasso_select_tool.h
- src/qt_frontend/qt_lasso_select_tool.cc

7.18 ccdvl::frontend::QtPanTool Class Reference

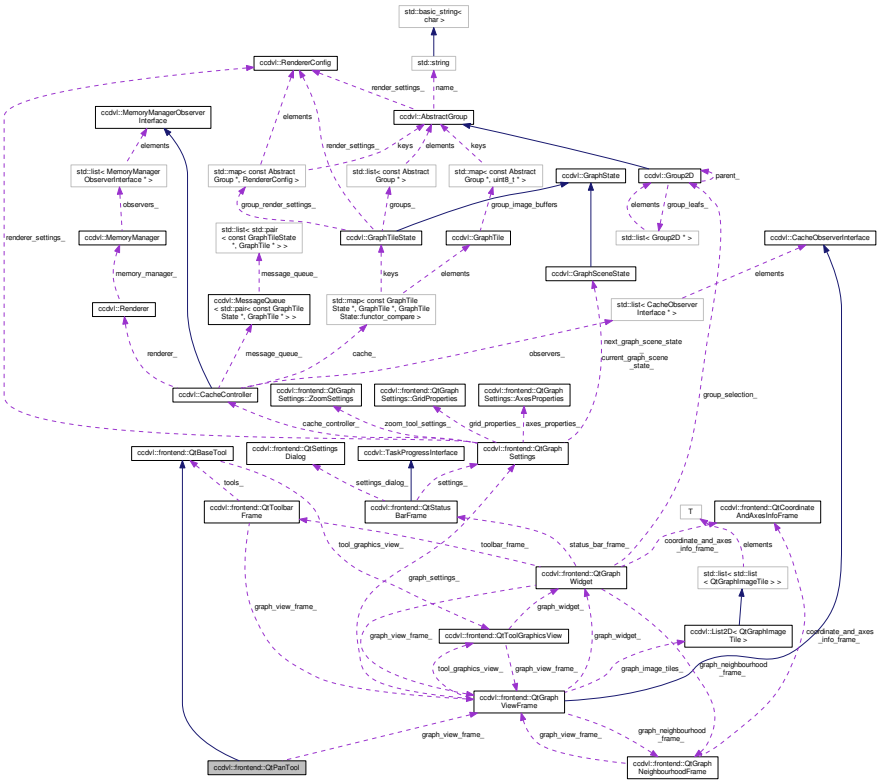
A small reusable pan tool class that pans in a QtToolGraphicsView.

`#include <qt_pan_tool.h>`

Inheritance diagram for ccdvl::frontend::QtPanTool:



Collaboration diagram for ccdvl::frontend::QtPanTool:



Public Member Functions

- QtPanTool (QtGraphViewFrame *graph_view_frame, QtToolGraphicsView *tool_graphics_view)

  Constructs and initiates a pan tool object with the given QtToolGraphicsView.
- virtual bool OnMousePress (QMouseEvent *event)

  Starts panning on left mouse button press at the position given by the mouse event.
  Other mouse events are ignored.
- virtual bool OnMouseMove (QMouseEvent *event)

  Pans from the start position using the position given by the mouse event.
- virtual bool OnMouseRelease (QMouseEvent *event)

  Resets the start position on left mouse button release.
  Other mouse events are ignored.
- virtual bool OnWheel (QWheelEvent *event)

  Accepts all wheel events to disable manual scrolling.

Private Attributes

- QtGraphViewFrame * graph_view_frame_

  The graph view frame.

Additional Inherited Members

7.18.1 Detailed Description

A small reusable pan tool class that pans in a QtToolGraphicsView.

The pan tool is used in conjunction with mouse, wheel and key events.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 ccdvl::frontend::QtPanTool::QtPanTool (QtGraphViewFrame * *graph_view_frame,* QtToolGraphicsView * *tool_graphics_view*)

Constructs and initiates a pan tool object with the given QtToolGraphicsView.

Parameters

in	*graph_view_frame*	The graph view frame that uses the tool graphics view.
in	*tool_graphics_view*	The tool graphics view that uses the pan tool.

7.18.3 Member Function Documentation

7.18.3.1 bool ccdvl::frontend::QtPanTool::OnMouseMove (QMouseEvent * *event*) [virtual]

Pans from the start position using the position given by the mouse event.

Only triggers if the left mouse button is pressed.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.18.3.2 bool ccdvl::frontend::QtPanTool::OnMousePress (QMouseEvent * *event*) [virtual]

Starts panning on left mouse button press at the position given by the mouse event.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMouseMove(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.18.3.3 bool ccdvl::frontend::QtPanTool::OnMouseRelease (QMouseEvent * *event*) [virtual]

Resets the start position on left mouse button release.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseMove(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.18.3.4 bool ccdvl::frontend::QtPanTool::OnWheel (QWheelEvent * *event*) [virtual]

Accepts all wheel events to disable manual scrolling.

Parameters

in	*event*	The wheel event.

Returns

true to indicate that the event was accepted.

Reimplemented from ccdvl::frontend::QtBaseTool.

The documentation for this class was generated from the following files:

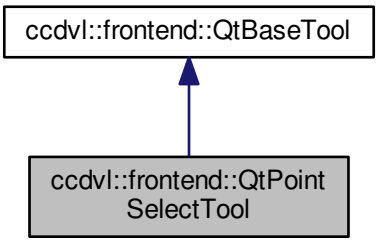
- include/qt_frontend/qt_pan_tool.h
- src/qt_frontend/qt_pan_tool.cc

7.19 ccdvl::frontend::QtPointSelectTool Class Reference

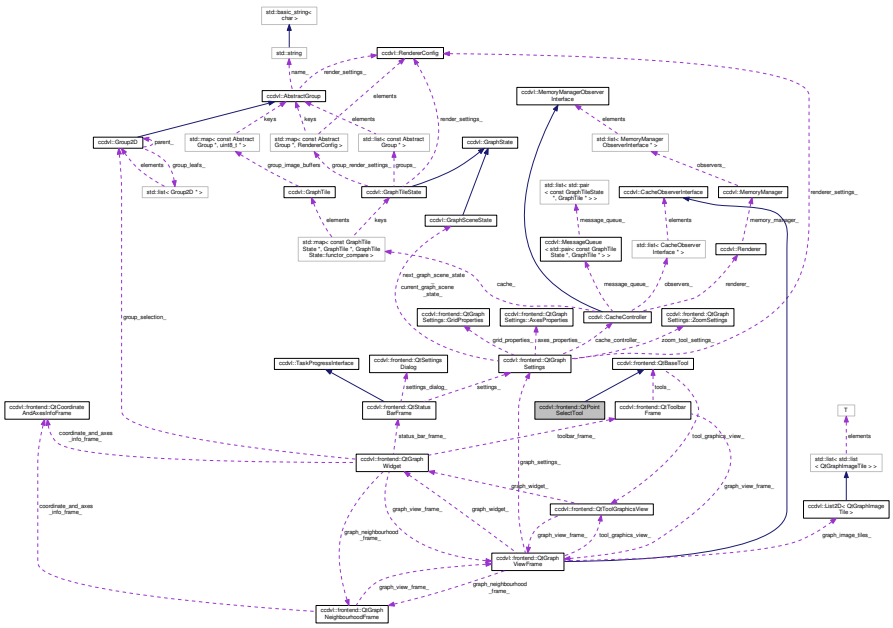
A reusable point selection tool class that selects "single points" in a QtToolGraphicsView.

`#include <qt_point_select_tool.h>`

Inheritance diagram for ccdvl::frontend::QtPointSelectTool:



Collaboration diagram for ccdvl::frontend::QtPointSelectTool:



Public Member Functions

- QtPointSelectTool (QtToolGraphicsView ∗tool_graphics_view)

  Constructs and initiates a point select tool object with the given QtToolGraphicsView.
- virtual void OnActivate ()

  Shows the cursor helper lines or mouse cursor if enabled.
- virtual void OnDeactivate ()

  Hides the cursor helper lines.
- virtual bool OnMousePress (QMouseEvent ∗event)

  Selects a point on left mouse button press at the position given by the mouse event. Right mouse press cancels the selection.
  Other mouse events are ignored.
- virtual bool OnMouseMove (QMouseEvent ∗event)

Updates the position of and redraws the cursor crosshair helper lines, originating from the position given by the mouse event.

- virtual bool OnWheel (QWheelEvent ∗event)

  Accepts all wheel events to disable manual scrolling.
- virtual bool OnEnter (QEvent ∗event)

  Shows the cursor crosshair helper lines if enabled.
- virtual bool OnLeave (QEvent ∗event)

  Hides the cursor crosshair helper lines if enabled.
- bool mouse_cursor ()

  Tells if the mouse cursor is visible or not.
- void set_mouse_cursor (bool enabled)

  Shows or hides the mouse cursor.
- bool helper_lines ()

  Tells if the helper lines are visible or not.
- void set_helper_lines (bool enabled)

  Shows or hides the helper lines.

Private Member Functions

- void DrawHelperLines (QPainter &painter, ViewPointF center_pos)

  Draws the crosshair helper lines.

Private Attributes

- bool mouse_cursor_

  Display mouse cursor.
- bool helper_lines_

  Display helper lines.
- QGraphicsPixmapItem ∗ helper_lines_pixmap_item_

  The graphics pixmap item used to show the cursor helper lines.

Additional Inherited Members

7.19.1 Detailed Description

A reusable point selection tool class that selects "single points" in a QtToolGraphicsView.

A small selection area is used to try and cover a point under it, since it is not possible to actually select a specific point with the current graph image implementation.

Toggleable helper crosshair lines originating from and following the cursor points to the graph view edges for extra visual support.

The point select tool is used in conjunction with mouse, wheel and key events.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 ccdvl::frontend::QtPointSelectTool::QtPointSelectTool (QtToolGraphicsView ∗ *tool_graphics_view*) [explicit]

Constructs and initiates a point select tool object with the given QtToolGraphicsView.

Parameters

in	*tool_graphics_view*	The tool graphics view that uses the point select tool.

7.19.3 Member Function Documentation

7.19.3.1 void ccdvl::frontend::QtPointSelectTool::DrawHelperLines (QPainter & *painter*, ViewPointF *center_pos*) [private]

Draws the crosshair helper lines.

Parameters

in	*painter*	The painter to draw the lines with.
in	*center_pos*	The center position of the crosshair.

7.19.3.2 bool ccdvl::frontend::QtPointSelectTool::helper_lines ( )

Tells if the helper lines are visible or not.

Returns

True if the helper lines are shown.

See also

set_helper_lines()

7.19.3.3 bool ccdvl::frontend::QtPointSelectTool::mouse_cursor ( )

Tells if the mouse cursor is visible or not.

Returns

True if the mouse cursor is shown.

See also

set_mouse_cursor()

7.19.3.4 void ccdvl::frontend::QtPointSelectTool::OnActivate ( ) [virtual]

Shows the cursor helper lines or mouse cursor if enabled.

See also

OnDeactivate()

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.5 void ccdvl::frontend::QtPointSelectTool::OnDeactivate ( ) [virtual]

Hides the cursor helper lines.

See also

OnActivate()

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.6 bool ccdvl::frontend::QtPointSelectTool::OnEnter (QEvent * *event*) [virtual]

Shows the cursor crosshair helper lines if enabled.

Parameters

in	*event*	The enter event.

Returns

true if the event was accepted.

See also

OnLeave(QEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.7 bool ccdvl::frontend::QtPointSelectTool::OnLeave (QEvent * *event*) [virtual]

Hides the cursor crosshair helper lines if enabled.

Parameters

in	*event*	The leave event.

Returns

true if the event was accepted.

See also

OnEnter(QEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.8 bool ccdvl::frontend::QtPointSelectTool::OnMouseMove (QMouseEvent * *event*) [virtual]

Updates the position of and redraws the cursor crosshair helper lines, originating from the position given by the mouse event.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.9 bool ccdvl::frontend::QtPointSelectTool::OnMousePress (QMouseEvent * *event*) [virtual]

Selects a point on left mouse button press at the position given by the mouse event. Right mouse press cancels the selection.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMouseMove(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.10 bool ccdvl::frontend::QtPointSelectTool::OnWheel (QWheelEvent * *event*) [virtual]

Accepts all wheel events to disable manual scrolling.

Parameters

in	*event*	The wheel event.

Returns

true if the event was accepted.

Reimplemented from ccdvl::frontend::QtBaseTool.

7.19.3.11 void ccdvl::frontend::QtPointSelectTool::set_helper_lines (bool *enabled*)

Shows or hides the helper lines.

Parameters

in	*enabled*	Sets the visibility of the helper lines.

See also

helper_lines()

7.19.3.12 void ccdvl::frontend::QtPointSelectTool::set_mouse_cursor (bool *enabled*)

Shows or hides the mouse cursor.

Parameters

in	*enabled*	Sets the visibility of the mouse cursor.

See also

mouse_cursor()

The documentation for this class was generated from the following files:

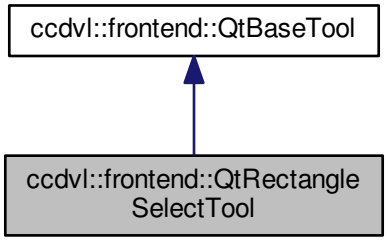
- include/qt_frontend/qt_point_select_tool.h
- src/qt_frontend/qt_point_select_tool.cc

7.20 ccdvl::frontend::QtRectangleSelectTool Class Reference

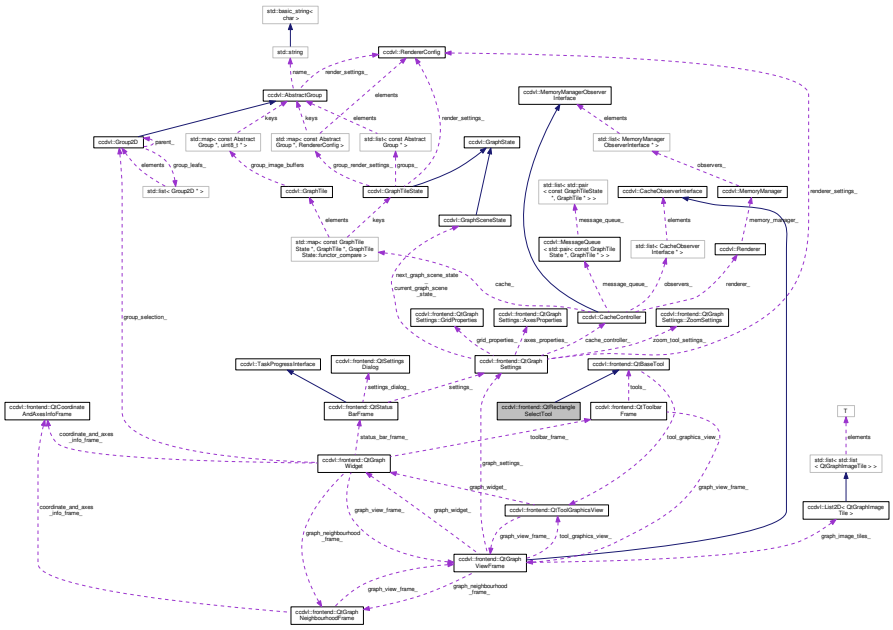
A rectangle selection rubberband that keeps track of the start position to make it simpler to set new geometry.

`#include <qt_rectangle_select_tool.h>`

Inheritance diagram for ccdvl::frontend::QtRectangleSelectTool:



Collaboration diagram for ccdvl::frontend::QtRectangleSelectTool:



Public Member Functions

- QtRectangleSelectTool (QtToolGraphicsView ∗tool_graphics_view)

  Constructs and initiates a rectangle select tool with the given QtToolGraphicsView.

- virtual bool OnMousePress (QMouseEvent ∗event)

  Starts a new rectangle selection.

- virtual bool OnMouseMove (QMouseEvent ∗event)

  Expands the rectangle selection to position given by the mouse event.

- virtual bool OnMouseRelease (QMouseEvent ∗event)

  Deactivates and hides the rectangle selection rubber band on left mouse button release if it is active. Other mouse events are ignored.

- virtual bool OnWheel (QWheelEvent ∗event)

  Accepts all wheel events to disable manual scrolling.

- QRubberBand ∗ rubber_band ()

  Returns the underlying QRubberBand used for the selection.

Private Attributes

- bool active_

  Keeps track of if the rectangle select tool is active.

- QRubberBand ∗ rubber_band_

  The Qt rubber band used to make the selection.

- GraphRectF selection_rect_

  The last selection of the rectangle select rubber band.

Additional Inherited Members

7.20.1 Detailed Description

A rectangle selection rubberband that keeps track of the start position to make it simpler to set new geometry.

The rubber band is used in conjunction with mouse, wheel and key events.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 ccdvl::frontend::QtRectangleSelectTool::QtRectangleSelectTool (QtToolGraphicsView * *tool_graphics_view*) [explicit]

Constructs and initiates a rectangle select tool with the given QtToolGraphicsView.

Parameters

in	*tool_graphics_view*	The tool graphics view that uses the rectangle select tool.

7.20.3 Member Function Documentation

7.20.3.1 bool ccdvl::frontend::QtRectangleSelectTool::OnMouseMove (QMouseEvent * *event*) [virtual]

Expands the rectangle selection to position given by the mouse event.

Only triggers if the rectangle select tool is active and the left mouse button is depressed.

Right mouse press cancels the selection and deactivates the rubber band.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.20.3.2 bool ccdvl::frontend::QtRectangleSelectTool::OnMousePress (QMouseEvent * *event*) [virtual]

Starts a new rectangle selection.

Only triggers on left mouse press and uses the position given by the mouse event, activates and shows the rectangle select rubber band.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMouseMove(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.20.3.3 bool ccdvl::frontend::QtRectangleSelectTool::OnMouseRelease (QMouseEvent * *event*) [virtual]

Deactivates and hides the rectangle selection rubber band on left mouse button release if it is active.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseMove(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.20.3.4 bool ccdvl::frontend::QtRectangleSelectTool::OnWheel (QWheelEvent * *event*) [virtual]

Accepts all wheel events to disable manual scrolling.

Parameters

in	*event*	The wheel event.

Returns

true to indicate that the event was accepted.

Reimplemented from ccdvl::frontend::QtBaseTool.

7.20.3.5 QRubberBand * ccdvl::frontend::QtRectangleSelectTool::rubber_band ( )

Returns the underlying QRubberBand used for the selection.

Returns

The rubber band used by the rectangle select tool.

The documentation for this class was generated from the following files:

- include/qt_frontend/qt_rectangle_select_tool.h
- src/qt_frontend/qt_rectangle_select_tool.cc

7.21 ccdvl::frontend::QtSettingsDialog Class Reference

Graph view settings configuration dialog.

`#include <qt_settings_dialog.h>`

Signals

- void clearcache ()

  Signal emitted when the empty cache button is clicked.

Public Member Functions

- QtSettingsDialog (QWidget *parent=NULL)

  Creates a new graph settings dialog with the given parent.
- void SetRendererPointShape (RendererConfig::PointShape shape)

  Set renderer point shape.
- RendererConfig::PointShape GetRendererPointShape ()

  Get renderer point shape.
- void SetRendererPointSize (int16_t size)

  Set renderer point size.
- int16_t GetRendererPointSize ()

  Get renderer point size.
- void SetRendererLineWidth (int16_t width)

  Set renderer line width.
- int16_t GetRendererLineWidth ()

  Get renderer line width.
- void SetRendererPointColor (const QColor &color)

  Set renderer point color.
- const QColor & GetRendererPointColor ()

  Get renderer point color.
- void SetRendererLineColor (const QColor &color)

  Set renderer line color.
- const QColor & GetRendererLineColor ()

  Get renderer line color.
- void SetRendererClearColor (const QColor &color)

  Set renderer clear color.
- const QColor & GetRendererClearColor ()

  Get renderer clear color.
- void SetScaleXValue (double scale)

  Set x-axis scale value.
- double GetScaleXValue ()

  Get x-axis scale value.
- void SetScaleYValue (double scale)

  Set y-axis scale value.
- double GetScaleYValue ()

  Get y-axis scale value.
- void SetScaleXMethod (GraphState::ScaleMethod method)

  Set x-axis scale method.
- GraphState::ScaleMethod GetScaleXMethod ()

  Get x-axis scale method.

- void SetScaleYMethod (GraphState::ScaleMethod method)

  Set y-axis scale method.
- GraphState::ScaleMethod GetScaleYMethod ()

  Get y-axis scale method.
- void SetCacheMaxTileCount (int32_t max_count)

  Set number of tiles to cache.
- int32_t GetCacheMaxTileCount ()

  Set number of tiles to cache.
- void SetCacheUsed (size_t usage)

  Set used cache information.

Private Slots

- void ButtonClicked (QAbstractButton *button)

  Dialog button events. Apply, discard and restore.
- void ColorButtonClicked ()

  Color selection button clicked.
- void ViewXPixelCountChanged ()

  Update x pixel count for view.
- void ViewYPixelCountChanged ()

  Update y pixel count for view.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (QtSettingsDialog)
- const QColor & GetButtonColor (QAbstractButton *button) const

  Obtain the backgroud color from a button.
- void SetButtonColor (QAbstractButton *button, const QColor &color)

  Sets the background color of a button.

Private Attributes

- QVBoxLayout * main_layout_

  View holding tabbed widget.
- QTabWidget * tabbed_view_

  Tabbed view of all settings.
- QWidget * tab1_

  Tab 1, used for basic graph settings.
- QVBoxLayout * layout_tab1_

  Dialog layout for the first tab.
- QWidget * tab2_

  Tab 2, used for view settings.
- QVBoxLayout * layout_tab2_

  Dialog layout for the second tab.
- QWidget * tab3_

  Tab 3, used for advanced controls.
- QVBoxLayout * layout_tab3_

  Dialog layout for the third tab.
- QGroupBox * renderer_settings_group_

  Renderer settings group.

- QHBoxLayout * renderer_layout_

  Renderer settings layout.
- QFormLayout * renderer_layout_1_

  Renderer settings layout part 1.
- QFormLayout * renderer_layout_2_

  Renderer settings layout part 2.
- QSpinBox * renderer_point_size_value_

  Renderer point size.
- QPushButton * renderer_point_color_

  Renderer point color.
- QComboBox * renderer_point_shape_

  Renderer point shape.
- QPushButton * renderer_clear_color_

  Tile clear color.
- QSpinBox * renderer_line_width_

  Renderer line width.
- QPushButton * renderer_line_color_

  Renderer line color.
- QGroupBox * scale_settings_group_

  Scale settings group.
- QHBoxLayout * scale_layout_

  Scale settings layout.
- QFormLayout * scale_layout_1_

  Scale settings layout part 1.
- QFormLayout * scale_layout_2_

  Scale layout part 2.
- QDoubleSpinBox * scale_x_value_

  X-axis scaling value.
- QDoubleSpinBox * scale_y_value_

  Y-axis scaling value.
- QComboBox * scale_x_method_

  X-axis scaling method.
- QComboBox * scale_y_method_

  Y-axis scaling method.
- QGroupBox * axis_settings_group_

  Axis setting group.
- QHBoxLayout * axis_layout_

  Axis settings layout.
- QFormLayout * axis_layout_1_

  Axis settings layout part 1.
- QFormLayout * axis_layout_2_

  Axis settings layout part 2.
- QLineEdit * axis_x_type_

  X-axis label.
- QLineEdit * axis_y_type_

  Y-axis label.
- QSpinBox * axis_x_step_

  Graph distance between every large X step.
- QSpinBox * axis_y_step_

  Graph distance between every large Y step.
- QSpinBox * axis_x_spacer_

Number of pixels between every small X step.

- QSpinBox * axis_y_spacer_

  Number of pixels between every small Y step.

- QPushButton * axis_line_color_

  Color of drawn axis dashes.

- QGroupBox * grid_settings_group_

  Grid settings group.

- QHBoxLayout * grid_layout_

  Grid settings layout.

- QFormLayout * grid_layout_1_

  Grid settings layout part 1.

- QFormLayout * grid_layout_2_

  Grid settings layout part 2.

- QSpinBox * grid_x_step_

  Distance between x grid lines.

- QSpinBox * grid_y_step_

  Distance between y grid lines.

- QPushButton * grid_line_color_

  Color of grid lines.

- QComboBox * grid_type_

  Grid type.

- QGroupBox * view_settings_group_

  View settings group.

- QHBoxLayout * view_layout_

  View settings layout.

- QFormLayout * view_layout_1_

  View settings layout part 1.

- QFormLayout * view_layout_2_

  View settings layout part 2.

- QVBoxLayout * view_layout_3_

  View settings layout part 3.

- QSpinBox * view_x_count_

  View width, in tiles.

- QSpinBox * view_x_size_

  The width of a tile.

- QLabel * view_x_final_

  View x size in pixels.

- QSpinBox * view_y_count_

  View height in tiles.

- QSpinBox * view_y_size_

  The height of a tile.

- QLabel * view_y_final_

  View y size in pixels.

- QGroupBox * cache_settings_group_

  Cache settings view.

- QHBoxLayout * cache_layout_

  Cache settings layout.

- QFormLayout * cache_layout_1_

  Cache settings layout part 1.

- QVBoxLayout * cache_layout_2_

  Cache settings layout part 2.

- QSpinBox * cache_tile_count_

  Mamximum number of tiles to cache.
- QLabel * cache_used_

  Amount of memory used by cached tiles.
- QPushButton * cache_empty_

  Clear cached tiles.
- QColorDialog * color_dialog_

  Color selection popup dialog.
- QDialogButtonBox * button_box_

  Dialog buttons.

7.21.1 Detailed Description

Graph view settings configuration dialog.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 ccdvl::frontend::QtSettingsDialog::QtSettingsDialog (QWidget * *parent* = NULL) [explicit]

Creates a new graph settings dialog with the given *parent*.

Parameters

in	*parent*	The parent widget of the graph settings dialog.

7.21.3 Member Function Documentation

7.21.3.1 void ccdvl::frontend::QtSettingsDialog::ButtonClicked (QAbstractButton * *button*) [private], [slot]

Dialog button events. Apply, discard and restore.

Excludes all set color buttons.

Parameters

in	*button*	The button clicked.

See also

ColorButtonClicked()

7.21.3.2 void ccdvl::frontend::QtSettingsDialog::ColorButtonClicked ( ) [private], [slot]

Color selection button clicked.

See also

ButtonClicked(QAbstractButton*)

7.21.3.3 const QColor & ccdvl::frontend::QtSettingsDialog::GetButtonColor (QAbstractButton * *button*) const [private]

Obtain the backgroud color from a button.

Parameters

in	*button*	Button.

Returns

Button background color.

See also

SetButtonColor(QAbstractButton*, const QColor&)

7.21.3.4 int32_t ccdvl::frontend::QtSettingsDialog::GetCacheMaxTileCount ( )

Set number of tiles to cache.

Returns

The maximum number of tiles to cache.

See also

SetCacheMaxTileCount(int8_t)

7.21.3.5 const QColor & ccdvl::frontend::QtSettingsDialog::GetRendererClearColor ( )

Get renderer clear color.

Returns

Selected clear color.

See also

SetRendererClearColor(const QColor&)

7.21.3.6 const QColor & ccdvl::frontend::QtSettingsDialog::GetRendererLineColor ( )

Get renderer line color.

Returns

Selected line color.

See also

SetRendererLineColor(const QColor&)

7.21.3.7 int16_t ccdvl::frontend::QtSettingsDialog::GetRendererLineWidth ( )

Get renderer line width.

Returns

Selected line width.

See also

SetRendererLineWidth(int16_t)

7.21.3.8 const QColor & ccdvl::frontend::QtSettingsDialog::GetRendererPointColor ( )

Get renderer point color.

Returns

Selected point color.

See also

SetRendererPointColor(const QColor&)

7.21.3.9 RendererConfig::PointShape ccdvl::frontend::QtSettingsDialog::GetRendererPointShape ( )

Get renderer point shape.

Returns

Selected renderer point shape.

See also

SetRendererPointShape()

7.21.3.10 int16_t ccdvl::frontend::QtSettingsDialog::GetRendererPointSize ( )

Get renderer point size.

Returns

Selected point size.

See also

SetRendererPointSize(int16_t)

7.21.3.11 GraphState::ScaleMethod ccdvl::frontend::QtSettingsDialog::GetScaleXMethod ( )

Get x-axis scale method.

Returns

x scale method.

See also

SetScaleXMethod()

7.21.3.12 double ccdvl::frontend::QtSettingsDialog::GetScaleXValue ( )

Get x-axis scale value.

Returns

x scale value.

See also

SetScaleXValue(double)

7.21.3.13 GraphState::ScaleMethod ccdvl::frontend::QtSettingsDialog::GetScaleYMethod ( )

Get y-axis scale method.

Returns

y scale method.

See also

SetScaleYMethod()

7.21.3.14 double ccdvl::frontend::QtSettingsDialog::GetScaleYValue ( )

Get y-axis scale value.

Returns

y scale value.

See also

SetScaleYValue(double)

7.21.3.15 void ccdvl::frontend::QtSettingsDialog::SetButtonColor (QAbstractButton * *button*, const QColor & *color*) [private]

Sets the background color of a button.

Parameters

in	*button*	Button.
	color	Color.

See also

GetButtonColor(QAbstractButton*)

7.21.3.16 void ccdvl::frontend::QtSettingsDialog::SetCacheMaxTileCount (int32_t *max_count*)

Set number of tiles to cache.

Parameters

max_count	The maximum number of tiles to cache.

See also

GetCacheMaxTileCount()

7.21.3.17 void ccdvl::frontend::QtSettingsDialog::SetCacheUsed (size_t *usage*)

Set used cache information.

Parameters

usage	Memory used by cache, in bytes.

7.21.3.18 void ccdvl::frontend::QtSettingsDialog::SetRendererClearColor (const QColor & *color*)

Set renderer clear color.

Parameters

in	*color*	Current clear color.

See also

GetRendererClearColor()

7.21.3.19 void ccdvl::frontend::QtSettingsDialog::SetRendererLineColor (const QColor & *color*)

Set renderer line color.

Parameters

in	*color*	Current line color.

See also

GetRendererLineColor()

7.21.3.20 void ccdvl::frontend::QtSettingsDialog::SetRendererLineWidth (int16_t *width*)

Set renderer line width.

Parameters

width	Current line width.

See also

GetRendererLineWidth()

7.21.3.21 void ccdvl::frontend::QtSettingsDialog::SetRendererPointColor (const QColor & *color*)

Set renderer point color.

Parameters

in	*color*	Current point color.

See also

GetRendererPointColor()

7.21.3.22 void ccdvl::frontend::QtSettingsDialog::SetRendererPointShape (RendererConfig::PointShape *shape*)

Set renderer point shape.

Parameters

in	*shape*	Current renderer point shape.

See also

GetRendererPointShape()

7.21.3.23 void ccdvl::frontend::QtSettingsDialog::SetRendererPointSize (int16_t *size*)

Set renderer point size.

Parameters

size	Curent point size.

See also

GetRendererPointSize()

7.21.3.24 void ccdvl::frontend::QtSettingsDialog::SetScaleXMethod (GraphState::ScaleMethod *method*)

Set x-axis scale method.

Parameters

method	X scale method.

See also

GetScaleXMethod()

7.21.3.25 void ccdvl::frontend::QtSettingsDialog::SetScaleXValue (double *scale*)

Set x-axis scale value.

Parameters

scale	x scale value.

See also

GetScaleXValue()

7.21.3.26 void ccdvl::frontend::QtSettingsDialog::SetScaleYMethod (GraphState::ScaleMethod *method*)

Set y-axis scale method.

Parameters

method	Y scale method.

See also

GetScaleYMethod()

7.21.3.27 void ccdvl::frontend::QtSettingsDialog::SetScaleYValue (double *scale*)

Set y-axis scale value.

Parameters

scale	Y scale value.

See also

GetScaleYValue()

7.21.4 Member Data Documentation

7.21.4.1 QPushButton* ccdvl::frontend::QtSettingsDialog::axis_line_color_ [private]

Color of drawn axis dashes.

See also

QtGraphSettings::AxesProperties::line_color

7.21.4.2 QSpinBox* ccdvl::frontend::QtSettingsDialog::axis_x_spacer_ [private]

Number of pixels between every small X step.

See also

QtGraphSettings::AxesProperties::x_spacer

7.21.4.3 QSpinBox* ccdvl::frontend::QtSettingsDialog::axis_x_step_ [private]

Graph distance between every large X step.

See also

QtGraphSettings::AxesProperties::x_step

7.21.4.4 QLineEdit* ccdvl::frontend::QtSettingsDialog::axis_x_type_ [private]

X-axis label.

See also

QtGraphSettings::AxesProperties::x_label

7.21.4.5 QSpinBox* ccdvl::frontend::QtSettingsDialog::axis_y_spacer_ [private]

Number of pixels between every small Y step.

See also

QtGraphSettings::AxesProperties::y_spacer

7.21.4.6 QSpinBox* ccdvl::frontend::QtSettingsDialog::axis_y_step_ [private]

Graph distance between every large Y step.

See also

QtGraphSettings::AxesProperties::y_step

7.21.4.7 QLineEdit* ccdvl::frontend::QtSettingsDialog::axis_y_type_ [private]

Y-axis label.

See also

QtGraphSettings::AxesProperties::y_label

7.21.4.8 QPushButton* ccdvl::frontend::QtSettingsDialog::cache_empty_ [private]

Clear cached tiles.

See also

CacheController::Clear()

7.21.4.9 QSpinBox* ccdvl::frontend::QtSettingsDialog::cache_tile_count_ [private]

Mamximum number of tiles to cache.

See also

CacheController::SetMaxCacheSize(), CacheController::GetMaxCacheSize(), CacheController::cache_size_

7.21.4.10 QLabel* ccdvl::frontend::QtSettingsDialog::cache_used_ [private]

Amount of memory used by cached tiles.

See also

CacheController::GetMemoryUsage()

7.21.4.11 QPushButton* ccdvl::frontend::QtSettingsDialog::grid_line_color_ [private]

Color of grid lines.

See also

QtGraphSettings::GridProperties::line_color

7.21.4.12 QComboBox* ccdvl::frontend::QtSettingsDialog::grid_type_ [private]

Grid type.

See also

QtGraphSettings::GridProperties::type

7.21.4.13 QSpinBox* ccdvl::frontend::QtSettingsDialog::grid_x_step_ [private]

Distance between x grid lines.

See also

QtGraphSettings::GridProperties::x_step

7.21.4.14 QSpinBox* ccdvl::frontend::QtSettingsDialog::grid_y_step_ [private]

Distance between y grid lines.

See also

QtGraphSettings::GridProperties::y_step

7.21.4.15 QPushButton* ccdvl::frontend::QtSettingsDialog::renderer_clear_color_ [private]

Tile clear color.

See also

GraphTileState::clear_color_

7.21.4.16 QPushButton* ccdvl::frontend::QtSettingsDialog::renderer_line_color_ [private]

Renderer line color.

See also

RendererConfig::line_color_

7.21.4.17 QSpinBox* ccdvl::frontend::QtSettingsDialog::renderer_line_width_ [private]

Renderer line width.

See also

RendererConfig::line_width_

7.21.4.18 QPushButton* ccdvl::frontend::QtSettingsDialog::renderer_point_color_ [private]

Renderer point color.

See also

RendererConfig::point_color_

7.21.4.19 QComboBox* ccdvl::frontend::QtSettingsDialog::renderer_point_shape_ [private]

Renderer point shape.

See also

RendererConfig::point_shape_

7.21.4.20 QSpinBox* ccdvl::frontend::QtSettingsDialog::renderer_point_size_value_ [private]

Renderer point size.

See also

RendererConfig::point_size_

7.21.4.21 QComboBox* ccdvl::frontend::QtSettingsDialog::scale_x_method_ [private]

X-axis scaling method.

See also

GraphState::scale_method_

7.21.4.22 QDoubleSpinBox* ccdvl::frontend::QtSettingsDialog::scale_x_value_ [private]

X-axis scaling value.

See also

GraphState::scale_

7.21.4.23 QComboBox* ccdvl::frontend::QtSettingsDialog::scale_y_method_ [private]

Y-axis scaling method.

See also

GraphState::scale_method_

7.21.4.24 QDoubleSpinBox* ccdvl::frontend::QtSettingsDialog::scale_y_value_ [private]

Y-axis scaling value.

See also

GraphState::scale_

7.21.4.25 QLabel* ccdvl::frontend::QtSettingsDialog::view_x_final_ [private]

View x size in pixels.

See also

view_x_count_, view_x_size_

7.21.4.26 QLabel* ccdvl::frontend::QtSettingsDialog::view_y_final_ [private]

View y size in pixels.

See also

view_y_count_, view_y_size_

The documentation for this class was generated from the following files:

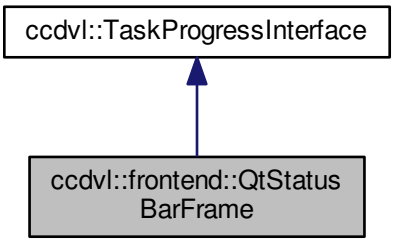
- include/qt_frontend/qt_settings_dialog.h
- src/qt_frontend/qt_settings_dialog.cc

7.22 ccdvl::frontend::QtStatusBarFrame Class Reference

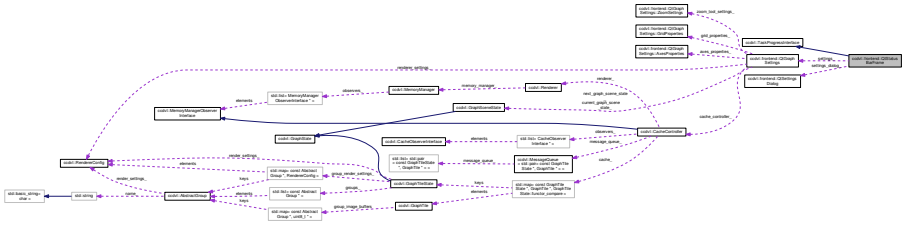
A status bar displaying mouse-over information, the progress bar and the settings button.

`#include <qt_status_bar_frame.h>`

Inheritance diagram for ccdvl::frontend::QtStatusBarFrame:



Collaboration diagram for ccdvl::frontend::QtStatusBarFrame:



Public Member Functions

- QtStatusBarFrame (QtGraphSettings *settings, QWidget *parent=NULL)

  Constructs and initiates a status bar frame with the given parent.
- int Init ()

  Initiates the layout, status bar and progress bar.
- void HideAndResetTaskProgress ()

Hides and resets the progress bar label and prograss bar.

- virtual void SetTaskProgress (QString text, int value, int max)

  Overloaded function to support Qt translation of strings.
- virtual void SetTaskProgress (const char *text, int value, int max)

  Sets the text of the progress bar label and updates the progress bar.
- QStatusBar * status_bar ()

  Returns the status bar used to display status tips.
- QProgressBar * progress_bar ()

  Returns the progress bar used to display task progress.

Private Slots

- void OpenSettingsDialog ()

  Open a settings dialog.
- void ClearCache ()

  Clears cache.

Private Attributes

- QtGraphSettings * settings_

  Qt frontend settings and graph configuration.
- QGridLayout * main_layout_

  Main widget layout.
- QHBoxLayout * left_layout_

  The left sub-layout of the status bar frame.
- QHBoxLayout * right_layout_

  The right sub-layout of the status bar frame.
- QStatusBar * status_bar_

  The status bar of the status bar frame.
- QProgressBar * progress_bar_

  The progress bar of the status bar frame.
- QLabel * progress_bar_label_

  The progress bar label of the status bar frame.
- QPushButton * settings_button_

  The settings button of the status bar frame that opens the settings dialog.
- QtSettingsDialog * settings_dialog_

  The settings dialog shown when the settings button is pushed.

7.22.1 Detailed Description

A status bar displaying mouse-over information, the progress bar and the settings button.

It also handles the settings dailog and clear cache signal emitted from said dialog.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 ccdvl::frontend::QtStatusBarFrame::QtStatusBarFrame (QtGraphSettings * *settings*, QWidget * *parent* = NULL) [explicit]

Constructs and initiates a status bar frame with the given parent.

Parameters

in	*settings*	Shared settings used for rendering graphs.
in	*parent*	The parent widget.

7.22.3 Member Function Documentation

7.22.3.1 void ccdvl::frontend::QtStatusBarFrame::HideAndResetTaskProgress ( )

Hides and resets the progress bar label and prograss bar.

See also

SetTaskProgress(QString, int, int) and SetTaskProgress(constc char*, int, int)

7.22.3.2 int ccdvl::frontend::QtStatusBarFrame::Init ( )

Initiates the layout, status bar and progress bar.

Returns

0 if the initializiation was successfull.

7.22.3.3 QProgressBar * ccdvl::frontend::QtStatusBarFrame::progress_bar ( )

Returns the progress bar used to display task progress.

Returns

The progress bar.

7.22.3.4 void ccdvl::frontend::QtStatusBarFrame::SetTaskProgress (QString *text,* int *value,* int *max*) [virtual]

Overloaded function to support Qt translation of strings.

See also

SetTaskProgress(const char*, int, int)

7.22.3.5 void ccdvl::frontend::QtStatusBarFrame::SetTaskProgress (const char * *text,* int *value,* int *max*) [virtual]

Sets the text of the progress bar label and updates the progress bar.

The progressbar is hidden if *value* and *max* is equal.

If *max* is not positive it is set to zero.

Parameters

in	*text*	The new label text.
in	*value*	The new value of the progress bar.
in	*max*	The maximum value of the progress bar.

See also

SetTaskProgress(QString, int, int)

Implements ccdvl::TaskProgressInterface.

7.22.3.6 QStatusBar * ccdvl::frontend::QtStatusBarFrame::status_bar ( )

Returns the status bar used to display status tips.

Returns

The status bar.

The documentation for this class was generated from the following files:

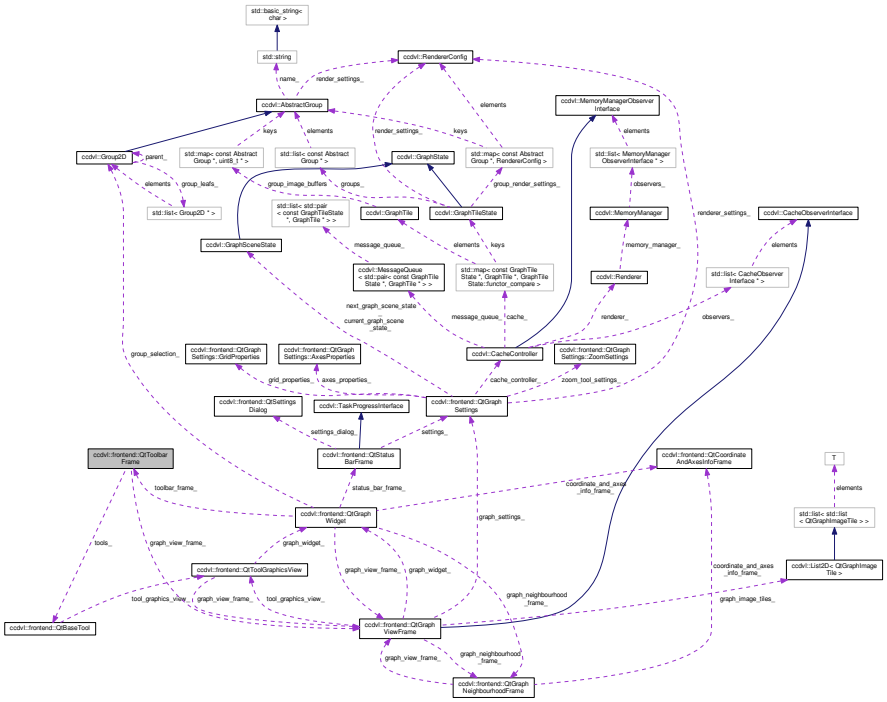
- include/qt_frontend/qt_status_bar_frame.h
- src/qt_frontend/qt_status_bar_frame.cc

7.23 ccdvl::frontend::QtToolbarFrame Class Reference

The toolbar, containing the tools and tool buttons.

`#include <qt_toolbar_frame.h>`

Collaboration diagram for ccdvl::frontend::QtToolbarFrame:



Public Types

- enum Tool {
  kCurrent, **kPointSelect**, **kRectangleSelect**, **kLassoSelect**,
  kPan, **kZoom** }

  Keeps track of the different tools.

Public Member Functions

- QtToolbarFrame (QWidget *parent=NULL)

  Constructs and initiates a toolbar frame with the given parent.
- int Init (QtGraphViewFrame *graph_view_frame)

  Creates and initiates the toolbar buttons, tools and layout.
- QtBaseTool * current_tool ()

  The current tool selected through the tool buttons.
- void set_current_tool (Tool tool)

  Sets the current tool as well as selecting the corresponding tool button.
- QtBaseTool * GetTool (Tool tool)

  Fetches the specified tool.

Static Public Attributes

- static const int kNumberOfTools = 6

  The number of graph interaction tools.

Private Slots

- void ToolButtonClicked (int id)

  Updates the currently selected tool when a tool button is clicked.
- void ResetZoomLevel ()

  Resets the zoom level of the tool graphics view to 100%.

Private Member Functions

- void ToolButtonDoubleClicked (int id)

  Perform a double click action for the specified tool.
- void CreateToolButtons ()

  Creates and initiates the tool buttons of the toolbar frame.
- void CreateTools ()

  Creates and initiates the tools of the toolbar frame.

Private Attributes

- QtGraphViewFrame * graph_view_frame_

  The graphics view used by the graphics scene.
- QtBaseTool * tools_ [kNumberOfTools]

  The graph interaction tools.
- QButtonGroup * tool_button_group_

  Keeps track of the currently selected tool button and makes sure that only one button is selected at a time.
- QHBoxLayout * main_layout_

  The main layout containing all components of the object.
- QPushButton * tool_buttons_ [kNumberOfTools]

  The tool buttons.
- QPushButton * reset_zoom_button_

  The reset zoom button.
- QTimer * double_click_timer_

  A timer for registering double clicks on the tool buttons.

7.23.1 Detailed Description

The toolbar, containing the tools and tool buttons.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 ccdvl::frontend::QtToolbarFrame::QtToolbarFrame (QWidget * *parent* = NULL) [explicit]

Constructs and initiates a toolbar frame with the given parent.

Parameters

in	*parent*	The parent widget.

7.23.3 Member Function Documentation

7.23.3.1 QtBaseTool * ccdvl::frontend::QtToolbarFrame::current_tool ( )

The current tool selected through the tool buttons.

Returns

The currently selected tool.

7.23.3.2 QtBaseTool * ccdvl::frontend::QtToolbarFrame::GetTool (Tool *tool*)

Fetches the specified *tool*.

Parameters

in	*tool*	The tool to be fetched.

Returns

A pointer to the specified *tool*.

7.23.3.3 int ccdvl::frontend::QtToolbarFrame::Init (QtGraphViewFrame * *graph_view_frame*)

Creates and initiates the toolbar buttons, tools and layout.

Parameters

in	*graph_view_frame*	The graph view frame that uses the tools.

Returns

0 if the initializiation was successfull.

7.23.3.4 void ccdvl::frontend::QtToolbarFrame::set_current_tool (Tool *tool*)

Sets the current tool as well as selecting the corresponding tool button.

Parameters

in	*tool*	The tool to be selected.

7.23.3.5 void ccdvl::frontend::QtToolbarFrame::ToolButtonClicked (int *id*) [private], [slot]

Updates the currently selected tool when a tool button is clicked.

Also checks for double clicks on the tool buttons, and calls ToolButtonDoubleClicked(int) if there a double click was detected.

Parameters

in	*id*	The id corresponding to the clicked tool button.

7.23.3.6 void ccdvl::frontend::QtToolbarFrame::ToolButtonDoubleClicked (int *id*) [private]

Perform a double click action for the specified tool.

Parameters

in	*id*	The id corresponding to the double clicked tool button.

7.23.4 Member Data Documentation

7.23.4.1 const int ccdvl::frontend::QtToolbarFrame::kNumberOfTools = 6 [static]

The number of graph interaction tools.

This should be the number of tools inheriting QtBaseTool + 1 to also hold the currently selected tool.

The documentation for this class was generated from the following files:

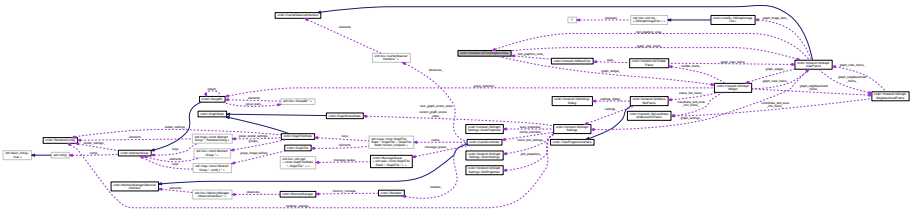
- include/qt_frontend/qt_toolbar_frame.h
- src/qt_frontend/qt_toolbar_frame.cc

7.24 ccdvl::frontend::QtToolGraphicsView Class Reference

A graphics view that handles mouse button, wheel and key events according to the currently selected tool in the toolbar frame.

`#include <qt_tool_graphics_view.h>`

Collaboration diagram for ccdvl::frontend::QtToolGraphicsView:



Public Member Functions

- QtToolGraphicsView (QGraphicsScene *graphics_scene, QtGraphWidget *graph_widget, QtGraphViewFrame *graph_view_frame)

  Constructs and initiates a tool graphics view with the given graphics scene, graph widget and graph view frame.
- int Init ()

  Initiates the object.
- void centerOn (const ScenePointF &pos)

  Only recenters the view if pos is more than 1 pixel away from the current center position to avoid minor jerking due to rounding errors.
- GraphPointF CalculateBottomLeftFromCenterPosition (const GraphPointF &center_pos, const GraphSceneState &state)

  Calculates the bottom left position for the given center position in the given graph scene state.
- GraphPointF mapToGraph (const ScenePointF &point, const GraphSceneState &state)

  Translates the given point from graphics scene coordinates to graph coordinates with double precision.
- GraphPointF mapToGraph (const ScenePointF &point)

  Overloaded function for convenience using the current graph scene state.
- ScenePointF mapFromGraph (const GraphPointF &point, const GraphSceneState &state)

  Translates the given point from graph coordinates to graphics scene coordinates with double precision.
- ScenePointF mapFromGraph (const GraphPointF &point)

  Overloaded function for convenience using the current graph scene state.
- GraphRectF mapToGraph (const SceneRectF &rect, const GraphSceneState &state)

  Translates the given rectangle from graphics scene coordinates to graph coordinates with double precision.
- GraphRectF mapToGraph (const SceneRectF &rect)

  Overloaded function for convenience using the current graph scene state.
- SceneRectF mapFromGraph (const GraphRectF &rect, const GraphSceneState &state)

  Translates the given rectangle from graph coordinates to graphics scene coordinates with double precision.
- SceneRectF mapFromGraph (const GraphRectF &rect)

  Overloaded function for convenience using the current graph scene state.
- GraphPolygonF mapToGraph (const ScenePolygonF &poly, const GraphSceneState &state)

  Translates the given polygon from graphics scene coordinates to graph coordinates with double precision.
- GraphPolygonF mapToGraph (const ScenePolygonF &poly)

  Overloaded function for convenience using the current graph scene state.
- ScenePolygonF mapFromGraph (const GraphPolygonF &poly, const GraphSceneState &state)

  Translates the given polygon from graph coordinates to graphics scene coordinates with double precision.
- ScenePolygonF mapFromGraph (const GraphPolygonF &poly)

  Overloaded function for convenience using the current graph scene state.
- GraphPairVector CurrentSelectionToStdVector ()

  Converts the current selection polygon to a standard vector of standard pairs in graph coordinates with double precision.
- GraphPolygonF current_selection ()

  The current selection polygon in graph coordinates.
- void set_current_selection (const GraphPolygonF &selection)

  Sets the current selection polygon and updates the shown selection.
- void UpdateShownCurrentSelection ()

  Updates the shown selection in the graphics scene to match the current selection.
- void HideCurrentSelection ()

  Hides the current selection.
- void ShowCurrentSelection ()

  Shows the intersection between the current selection and the scene.

Protected Member Functions

- virtual void mousePressEvent (QMouseEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void mouseMoveEvent (QMouseEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void mouseReleaseEvent (QMouseEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void wheelEvent (QWheelEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void keyPressEvent (QKeyEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void keyReleaseEvent (QKeyEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void enterEvent (QEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).
- virtual void leaveEvent (QEvent ∗event)

  Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Private Attributes

- QtGraphViewFrame ∗ graph_view_frame_

  The parent graph view frame.
- QtGraphWidget ∗ graph_widget_

  The graph widget GUI, used to access the toolbar frame.
- QGraphicsPolygonItem ∗ selection_polygon_item_

  The graphics polygon item used to display the last selection in the graphics scene (or the intersecting part of the selection).
- GraphPolygonF current_selection_

  The current selection polygon (in graph coordinates).
- int wheel_event_restricted_

  Used to restrict (ignore) two out of three wheel events to not trigger too many zoom updates.

7.24.1 Detailed Description

A graphics view that handles mouse button, wheel and key events according to the currently selected tool in the toolbar frame.

It also manages the current selection made by the selection tools, and displays this as a semi-transparent overlay polygon.

See also

QtToolbarFrame

7.24.2 Constructor & Destructor Documentation

7.24.2.1 ccdvl::frontend::QtToolGraphicsView::QtToolGraphicsView (QGraphicsScene * *graphics_scene*, QtGraphWidget * *graph_widget*, QtGraphViewFrame * *graph_view_frame*)

Constructs and initiates a tool graphics view with the given graphics scene, graph widget and graph view frame.

Parameters

in	*graphics_scene*	The graphics scene used by the graph view frame.
in	*graph_widget*	The main widget of the GUI.
in	*graph_view_frame*	The parent graph view frame.

7.24.3 Member Function Documentation

7.24.3.1 GraphPointF ccdvl::frontend::QtToolGraphicsView::CalculateBottomLeftFromCenterPosition (const GraphPointF & *center_pos*, const GraphSceneState & *state*)

Calculates the bottom left position for the given center position in the given graph scene state.

The calculated position is strictly relative to the given center position, not the graph image tile grid.

Parameters

in	*center_pos*	The center position of the new bottom left.
in	*state*	The graph scene state to calculate the bottom left position for.

Returns

The calculated bottom left position.

7.24.3.2 void ccdvl::frontend::QtToolGraphicsView::centerOn (const ScenePointF & *pos*)

Only recenters the view if *pos* is more than 1 pixel away from the current center position to avoid minor jerking due to rounding errors.

Parameters

in	*pos*	The position to center the view on.

See also

`QGraphicsView::centerOn()`

7.24.3.3 GraphPolygonF ccdvl::frontend::QtToolGraphicsView::current_selection ( )

The current selection polygon in graph coordinates.

Returns

The current selection polygon.

See also

set_current_selection()

7.24.3.4 GraphPairVector ccdvl::frontend::QtToolGraphicsView::CurrentSelectionToStdVector ( )

Converts the current selection polygon to a standard vector of standard pairs in graph coordinates with double precision.

Returns

The current selection as a vector.

See also

current_selection()

7.24.3.5 void ccdvl::frontend::QtToolGraphicsView::enterEvent (QEvent * *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The enter event.

See also

leaveEvent(QEvent*)

7.24.3.6 void ccdvl::frontend::QtToolGraphicsView::HideCurrentSelection ( )

Hides the current selection.

See also

ShowCurrentSelection(), current_selection() and set_current_selection(GraphPolygonF)

7.24.3.7 int ccdvl::frontend::QtToolGraphicsView::Init ( )

Initiates the object.

Must be called diretly after object creation.

Returns

0 if the initializiation was successfull.

7.24.3.8 void ccdvl::frontend::QtToolGraphicsView::keyPressEvent (QKeyEvent * *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The key event.

See also

OnKeyRelease(QKeyEvent∗)

7.24.3.9 void ccdvl::frontend::QtToolGraphicsView::keyReleaseEvent (QKeyEvent ∗ *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The key event.

See also

OnKeyPress(QKeyEvent∗)

7.24.3.10 void ccdvl::frontend::QtToolGraphicsView::leaveEvent (QEvent ∗ *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Also sets the displayed current coordinates under the mouse cursor to zero when leaving the graphics view.

Parameters

in	*event*	The leave event.

See also

enterEvent(QEvent∗)

7.24.3.11 ScenePointF ccdvl::frontend::QtToolGraphicsView::mapFromGraph (const GraphPointF & *point,* const GraphSceneState & *state*)

Translates the given point from graph coordinates to graphics scene coordinates with double precision.

Parameters

in	*point*	The double precision point to be translated.
in	*state*	The graph scene state to be used in the calculations.

Returns

The translated point in graphics scene coordinates.

See also

MapFromGraph(GraphPointF), MapToGraph(ScenePointF) and MapToGraph(ScenePointF, const GraphTileState &)

7.24.3.12 ScenePointF ccdvl::frontend::QtToolGraphicsView::mapFromGraph (const GraphPointF & *point*)

Overloaded function for convenience using the current graph scene state.

See also

MapFromGraph(GraphPointF, &GraphSceneState), MapToGraph(ScenePointF) and MapToGraph(ScenePointF, const GraphSceneState &)

7.24.3.13 SceneRectF ccdvl::frontend::QtToolGraphicsView::mapFromGraph (const GraphRectF & *rect,* const GraphSceneState & *state*)

Translates the given rectangle from graph coordinates to graphics scene coordinates with double precision.

Parameters

in	*rect*	The double precision rectangle to be translated.
in	*state*	The graph scene state to be used in the calculations.

Returns

The translated rectangle in graphics scene coordinates.

See also

MapFromGraph(GraphRectF), MapToGraph(SceneRectF) and MapToGraph(SceneRectF, GraphTileState∗)

7.24.3.14 SceneRectF ccdvl::frontend::QtToolGraphicsView::mapFromGraph (const GraphRectF & *rect*)

Overloaded function for convenience using the current graph scene state.

See also

MapFromGraph(GraphRectF, const GraphSceneState &), MapToGraph(SceneRectF) and MapToGraph(SceneRectF, const GraphSceneState &)

7.24.3.15 ScenePolygonF ccdvl::frontend::QtToolGraphicsView::mapFromGraph (const GraphPolygonF & *poly,* const GraphSceneState & *state*)

Translates the given polygon from graph coordinates to graphics scene coordinates with double precision.

Parameters

in	*poly*	The double precision polygon to be translated.
in	*state*	The graph scene state to be used in the calculations.

Returns

The translated polygon in graphics scene coordinates.

See also

MapFromGraph(GraphPolygonF), MapToGraph(ScenePolygonF) and MapToGraph(ScenePolygonF, const GraphSceneState &)

7.24.3.16 ScenePolygonF ccdvl::frontend::QtToolGraphicsView::mapFromGraph (const GraphPolygonF & *poly*)

Overloaded function for convenience using the current graph scene state.

See also

MapFromGraph(GraphPolygonF, const GraphSceneState &), MapToGraph(ScenePolygonF) and MapToGraph(ScenePolygonF, const GraphSceneState &)

7.24.3.17 GraphPointF ccdvl::frontend::QtToolGraphicsView::mapToGraph (const ScenePointF & *point,* const GraphSceneState & *state*)

Translates the given point from graphics scene coordinates to graph coordinates with double precision.

Parameters

in	*point*	The double precision point to be translated.
in	*state*	The graph scene state to be used in the calculations.

Returns

The translated point in graph coordinates.

See also

MapToGraph(ScenePointF), MapFromGraph(GraphPointF) and MapFromGraph(GraphPointF, &GraphSceneState)

7.24.3.18 GraphPointF ccdvl::frontend::QtToolGraphicsView::mapToGraph (const ScenePointF & *point*)

Overloaded function for convenience using the current graph scene state.

See also

MapToGraph(ScenePointF, &GraphSceneState), MapFromGraph(GraphPointF) and MapFromGraph(GraphPointF, const GraphSceneState &)

7.24.3.19 GraphRectF ccdvl::frontend::QtToolGraphicsView::mapToGraph (const SceneRectF & *rect,* const GraphSceneState & *state*)

Translates the given rectangle from graphics scene coordinates to graph coordinates with double precision.

Parameters

in	*rect*	The double precision rectangle to be translated.
in	*state*	The graph scene state to be used in the calculations.

Returns

The translated rectangle in graphics scene coordinates.

See also

MapToGraph(SceneRectF), MapFromGraph(GraphRectF) and MapFromGraph(GraphRectF, const GraphSceneState &)

7.24.3.20 GraphRectF ccdvl::frontend::QtToolGraphicsView::mapToGraph (const SceneRectF & *rect*)

Overloaded function for convenience using the current graph scene state.

See also

MapToGraph(SceneRectF, const GraphSceneState &), MapFromGraph(GraphRectF) and MapFromGraph(GraphRectF, const GraphSceneState &)

7.24.3.21 GraphPolygonF ccdvl::frontend::QtToolGraphicsView::mapToGraph (const ScenePolygonF & *poly,* const GraphSceneState & *state*)

Translates the given polygon from graphics scene coordinates to graph coordinates with double precision.

Parameters

in	*poly*	The double precision polygon to be translated.
in	*state*	The graph scene state to be used in the calculations.

Returns

The translated polygon in graphics scene coordinates.

See also

MapToGraph(ScenePolygonF), MapFromGraph(GraphPolygonF) and MapFromGraph(GraphPolygonF, const GraphSceneState &)

7.24.3.22 GraphPolygonF ccdvl::frontend::QtToolGraphicsView::mapToGraph (const ScenePolygonF & *poly*)

Overloaded function for convenience using the current graph scene state.

See also

MapToGraph(ScenePolygonF, const GraphSceneState &), MapFromGraph(GraphPolygonF) and MapFromGraph(GraphPolygonF, const GraphSceneState &)

7.24.3.23 void ccdvl::frontend::QtToolGraphicsView::mouseMoveEvent (QMouseEvent * *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The mouse event.

See also

OnMousePress(QMouseEvent∗) and OnMouseRelease(QMouseEvent∗)

7.24.3.24 void ccdvl::frontend::QtToolGraphicsView::mousePressEvent (QMouseEvent ∗ *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The mouse event.

See also

OnMouseMove(QMouseEvent∗) and OnMouseRelease(QMouseEvent∗)

7.24.3.25 void ccdvl::frontend::QtToolGraphicsView::mouseReleaseEvent (QMouseEvent ∗ *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The mouse event.

See also

OnMousePress(QMouseEvent∗) and OnMouseMove(QMouseEvent∗)

7.24.3.26 void ccdvl::frontend::QtToolGraphicsView::set_current_selection (const GraphPolygonF & *selection*)

Sets the current selection polygon and updates the shown selection.

Parameters

in	*selection*	The new selection.

See also

current_selection()

7.24.3.27 void ccdvl::frontend::QtToolGraphicsView::ShowCurrentSelection ( )

Shows the intersection between the current selection and the scene.

The current selection is only shown if the part of the selection that intersects the scene is non-empty.

See also

HideCurrentSelection(), current_selection() and set_current_selection(GraphPolygonF)

7.24.3.28 void ccdvl::frontend::QtToolGraphicsView::wheelEvent (QWheelEvent * *event*) [protected], [virtual]

Forwards the event to the corresponding function in the currently selected tool (fetched from the toolbar frame via the graph widget).

Parameters

in	*event*	The wheel event.

The documentation for this class was generated from the following files:

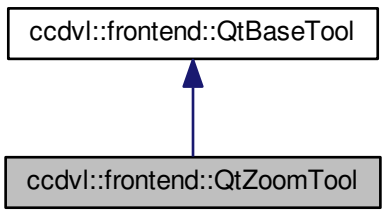
- include/qt_frontend/qt_tool_graphics_view.h
- src/qt_frontend/qt_tool_graphics_view.cc

7.25 ccdvl::frontend::QtZoomTool Class Reference

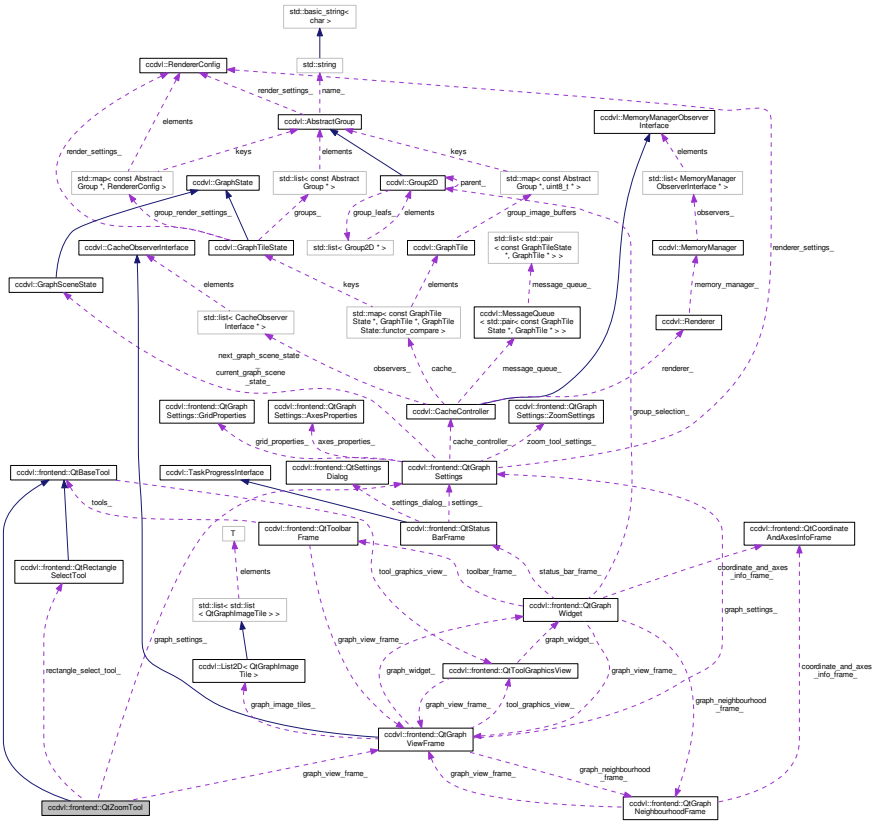
A small reusable zoom tool class that zooms in or out in a QtToolGraphicsView.

`#include <qt_zoom_tool.h>`

Inheritance diagram for ccdvl::frontend::QtZoomTool:



Collaboration diagram for ccdvl::frontend::QtZoomTool:



Public Types

- enum ZoomDirection { kIn, kOut }

  Represents the zoom level direction.

Public Member Functions

- QtZoomTool (QtGraphViewFrame *graph_view_frame, QtToolGraphicsView *tool_graphics_view)

  Constructs and initiates a zoom tool object with the given graph view frame and tool graphics view.

Public Attributes

- const QCursor kZoomPlusCursor

  A magnifying glass with a plus sign.
- const QCursor kZoomMinusCursor

  A magnifying glass with a minus sign.

Protected Member Functions

- virtual bool OnMousePress (QMouseEvent *event)

  Starts a new zoom selection and forwards the event to the internal zoom-select rectangle tool.
- virtual bool OnMouseMove (QMouseEvent *event)

Forwards the event to the internal zoom-select rectangle tool.

- virtual bool OnMouseRelease (QMouseEvent ∗event)

  Zooms on left mouse button release at the position given by the mouse event if the zoom-select rectangle has not been expanded, otherwise zoom is performed to fit all the contents of the zoom-select rectangle.
  Other mouse events are ignored.

- virtual bool OnKeyPress (QKeyEvent ∗event)

  Changes the cursor to kZoomMinusCursor (zoom out cursor) if the alt key is pressed.
  Other key events are ignored.

- virtual bool OnKeyRelease (QKeyEvent ∗event)

  Changes the cursor to kZoomPlusCursor (zoom in cursor) if the alt key is released.
  Other key events are ignored.

- virtual bool OnWheel (QWheelEvent ∗event)

  Zooms according to the wheel direction.

Private Member Functions

- void DoZoom (ZoomDirection zoom, int x_step_factor, int y_step_factor)

  Perform the zoom operation.

Private Attributes

- QtGraphViewFrame ∗ graph_view_frame_

  The graph view frame.

- QtGraphSettings ∗ graph_settings_

  The graph settings.

- QtRectangleSelectTool ∗ rectangle_select_tool_

  The rectangle select tool used to select-zoom.

- bool active_

  Keeps track of if the zoom rectangle select is active.

Additional Inherited Members

7.25.1 Detailed Description

A small reusable zoom tool class that zooms in or out in a QtToolGraphicsView.

Left clicking zooms in, and alt-left clicking zooms out. Clicking and dragging zooms to the area shown by the drawn selection rectangle on mouse button release. Everything inside the selection rectangle is zoomed to and fitted as snuggly as possible.

The zoom tool is used in conjunction with mouse, wheel and key events.

7.25.2 Member Enumeration Documentation

7.25.2.1 enum ccdvl::frontend::QtZoomTool::ZoomDirection

Represents the zoom level direction.

Enumerator:

kIn Increase zoom level.

kOut Decrease zoom level.

7.25.3 Constructor & Destructor Documentation

7.25.3.1 ccdvl::frontend::QtZoomTool::QtZoomTool (QtGraphViewFrame * *graph_view_frame,* QtToolGraphicsView * *tool_graphics_view*)

Constructs and initiates a zoom tool object with the given graph view frame and tool graphics view.

Parameters

in	*tool_graphics_view*	The tool graphics view that uses the zoom tool.
in	*graph_view_frame*	The graph view frame that uses the tool graphics view.

See also

QtGraphSettings

7.25.4 Member Function Documentation

7.25.4.1 void ccdvl::frontend::QtZoomTool::DoZoom (QtZoomTool::ZoomDirection *zoom,* int *x_step_factor,* int *y_step_factor*) [private]

Perform the zoom operation.

The zoom factors give the possibility to compose several operations into one.

Parameters

in	*zoom*	The zoom direction.
in	*x_step_factor*	The x zoom step factor.
in	*y_step_factor*	The y zoom step factor.

7.25.4.2 bool ccdvl::frontend::QtZoomTool::OnKeyPress (QKeyEvent * *event*) [protected], [virtual]

Changes the cursor to kZoomMinusCursor (zoom out cursor) if the alt key is pressed.

Other key events are ignored.

Parameters

in	*event*	The key event.

Returns

true if the event was accepted.

See also

OnKeyRelease(QKeyEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.25.4.3 bool ccdvl::frontend::QtZoomTool::OnKeyRelease (QKeyEvent * *event*) [protected], [virtual]

Changes the cursor to kZoomPlusCursor (zoom in cursor) if the alt key is released.

Other key events are ignored.

Parameters

in	*event*	The key event.

Returns

true if the event was accepted.

See also

OnKeyPress(QKeyEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.25.4.4 bool ccdvl::frontend::QtZoomTool::OnMouseMove (QMouseEvent * *event*) [protected], [virtual]

Forwards the event to the internal zoom-select rectangle tool.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.25.4.5 bool ccdvl::frontend::QtZoomTool::OnMousePress (QMouseEvent * *event*) [protected], [virtual]

Starts a new zoom selection and forwards the event to the internal zoom-select rectangle tool.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMouseMove(QMouseEvent*) and OnMouseRelease(QMouseEvent*)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.25.4.6 bool ccdvl::frontend::QtZoomTool::OnMouseRelease (QMouseEvent * *event*) [protected], [virtual]

Zooms on left mouse button release at the position given by the mouse event if the zoom-select rectangle has not been expanded, otherwise zoom is performed to fit all the contents of the zoom-select rectangle.

Other mouse events are ignored.

Parameters

in	*event*	The mouse event.

Returns

true if the event was accepted.

See also

OnMousePress(QMouseEvent∗) and OnMouseMove(QMouseEvent∗)

Reimplemented from ccdvl::frontend::QtBaseTool.

7.25.4.7 bool ccdvl::frontend::QtZoomTool::OnWheel (QWheelEvent ∗ *event*) [protected], [virtual]

Zooms according to the wheel direction.

Parameters

in	*event*	The wheel event.

Returns

true if the event was accepted.

Reimplemented from ccdvl::frontend::QtBaseTool.

The documentation for this class was generated from the following files:

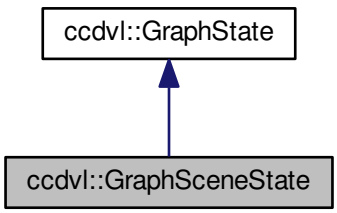
- include/qt_frontend/qt_zoom_tool.h
- src/qt_frontend/qt_zoom_tool.cc

7.26 ccdvl::GraphSceneState Class Reference

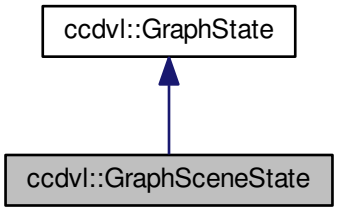
A class that keeps track of a two dimensional graph scene state, i.e. the values associated with a whole graph image composed of graph image tiles.

`#include <graph_scene_state.h>`

Inheritance diagram for ccdvl::GraphSceneState:



Collaboration diagram for ccdvl::GraphSceneState:



Public Member Functions

- GraphSceneState (int32_t width, int32_t height)

  Constructs a default graph scene state.
- GraphSceneState (const GraphSceneState &instance)

  Copy constructor.
- virtual ~GraphSceneState ()

  Destroys the object and frees any allocated resources.

Additional Inherited Members

7.26.1 Detailed Description

A class that keeps track of a two dimensional graph scene state, i.e. the values associated with a whole graph image composed of graph image tiles.

The values that make up a graph scene state are the bottom left graph coordinates of the graph image, graph image width and height, scale and zoom factors, scale method and number of dimensions.

See also

GraphTileState

7.26.2 Constructor & Destructor Documentation

7.26.2.1 ccdvl::GraphSceneState::GraphSceneState (int32_t *width*, int32_t *height*)

Constructs a default graph scene state.

Parameters

width	The scene graph image width.
height	The scene graph image height.

The documentation for this class was generated from the following files:

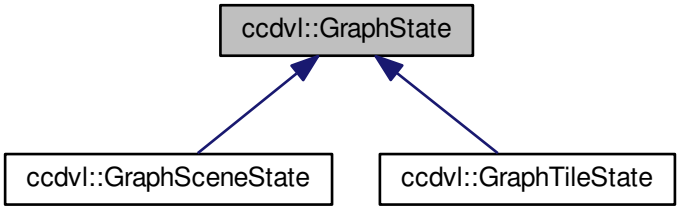
- include/graph_scene_state.h
- src/graph_scene_state.cc

7.27 ccdvl::GraphState Class Reference

A class that keeps track of a graph scene state, i.e. the values associated with a whole graph image composed of graph image tiles.

`#include <graph_state.h>`

Inheritance diagram for ccdvl::GraphState:



Public Types

- enum ScaleMethod { kInvalid = -1, kLinear, kLogarithmic }

  Enumeration of scaling methods. kInvalid Indicate an unknown or invalid scaling method, rendered image result is undefined with the exception of background color. kLinear Regular linear scaling, scaling constant is multiplied with point. kLogarithmic Logarithmic scaling, scaling constant represent logarithm used.

Public Member Functions

- GraphState (int8_t dimensions, int32_t width, int32_t height)

  Constructs a graph state.
- GraphState (const GraphState &instance)

  Copy constructor.
- virtual ~GraphState ()

  Destroys the object and frees any allocated resources.

Public Attributes

- const int8_t dimensions_

  Number of dimensions supported.
- int32_t width_

  Scene graph image width.
- int32_t height_

  Scene graph image height.
- double zoom_ [2]

  Zoom value for graph image.
- GraphDouble * bottom_left_

  Start value for each axis in graph.
- double * scale_

  Scaling value for each axis.

- ScaleMethod * scale_method_

  Scaling method used for each axis.

Protected Member Functions

- void Allocate ()

  Allocates memory for bottom_left_, scale_ and scale_method_.

7.27.1 Detailed Description

A class that keeps track of a graph scene state, i.e. the values associated with a whole graph image composed of graph image tiles.

The values that make up a graph scene state are the bottom left graph coordinates of the graph image, graph image width and height, scale and zoom factors, scale method and number of dimensions.

See also

GraphTileState

7.27.2 Member Enumeration Documentation

7.27.2.1 enum ccdvl::GraphState::ScaleMethod

Enumeration of scaling methods. *kInvalid* Indicate an unknown or invalid scaling method, rendered image result is undefined with the exception of background color. *kLinear* Regular linear scaling, scaling constant is multiplied with point. *kLogarithmic* Logarithmic scaling, scaling constant represent logarithm used.

Enumerator:

kInvalid Invalid scaling.

kLinear Linear scaling, scaling value is a multiplication constant.

kLogarithmic Logarithmic scaling, scaling value is the nth-logarithm.

7.27.3 Constructor & Destructor Documentation

7.27.3.1 ccdvl::GraphState::GraphState (int8_t *dimensions*, int32_t *width*, int32_t *height*)

Constructs a graph state.

Parameters

in	*dimensions*	The number of dimensions. Must be positive and larger then zero.
	width	The graph scene image width.
	height	The graph scene image height.

7.27.4 Member Data Documentation

7.27.4.1 GraphDouble* ccdvl::GraphState::bottom_left_

Start value for each axis in graph.

Its a point which points to the lower left corner of the graph image.

7.27.4.2 double* ccdvl::GraphState::scale_

Scaling value for each axis.

The effect of this value depends on scale_method_.

See also

ScaleMethod for effects.

7.27.4.3 ScaleMethod* ccdvl::GraphState::scale_method_

Scaling method used for each axis.

See also

ScaleMethod for effects.

7.27.4.4 double ccdvl::GraphState::zoom_[2]

Zoom value for graph image.

This setting is equal to scale iff scale method is set to linear, however in the case of other scaling methods the effect of this will be applied after scaling and it will always be a linear transform. Finally in higher dimensions (or with rotation) this value must be used to compute the data point search box.

The documentation for this class was generated from the following files:

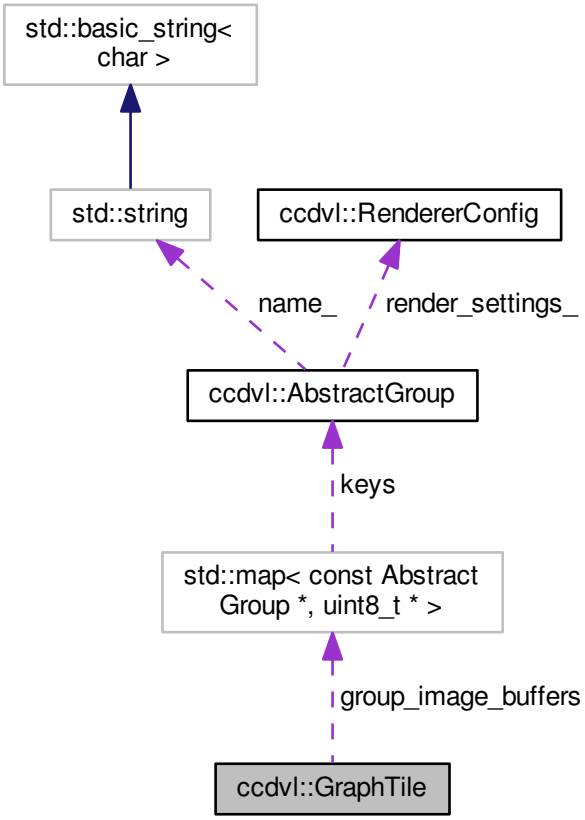
- include/graph_state.h
- src/graph_state.cc

7.28 ccdvl::GraphTile Class Reference

Structure which composes a rendered graph tile.

`#include <graph_tile.h>`

Collaboration diagram for ccdvl::GraphTile:



Public Member Functions

- GraphTile ()

  Default constructor.
- virtual ~GraphTile ()

  Destroys the object and frees any allocated resources.

Public Attributes

- time_t last_used

  Tile time-stamp for cache.
- bool completed

  True iff the renderer has finished.
- uint8_t * image_buffer

  Raw pixelbuffer in the format used by the renderer.
- std::map< const AbstractGroup *, uint8_t * > group_image_buffers

  Raw pixelbuffer overlays with an alpha channel for groups.

7.28.1 Detailed Description

Structure which composes a rendered graph tile.

The documentation for this class was generated from the following file:

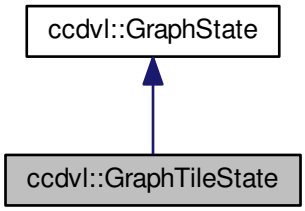
- include/graph_tile.h

7.29 ccdvl::GraphTileState Class Reference

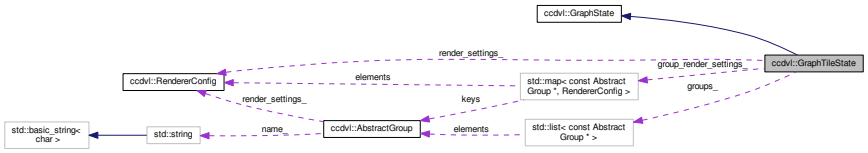
Holds configuration for rendering a graph tile.

`#include <graph_tile_state.h>`

Inheritance diagram for ccdvl::GraphTileState:



Collaboration diagram for ccdvl::GraphTileState:



Classes

- struct functor_compare

  Functor to compare graph tile states.

Public Member Functions

- GraphTileState (int8_t dimensions, int32_t width, int32_t height)

  Constructs a default graph tile state.
- GraphTileState (const GraphSceneState &graph_scene_state)

  Constructor that uses the values from the provided graph scene state.
- GraphTileState (const GraphTileState &instance)

  Copy constructor.

- virtual ~GraphTileState ()

  Destructor.
- bool LessThan (const GraphTileState &graphstate) const

  Less-than comparator for this class.

Public Attributes

- uint8_t clear_color_ [3]

  Graph clear color.
- RendererConfig * render_settings_

  Default render settings, used iff point-in-polygon missed on all groups.
- std::list< const AbstractGroup * > groups_

  Groups.
- std::map< const AbstractGroup *, RendererConfig > group_render_settings_

  Group render settings.

Private Member Functions

- void operator= (const GraphTileState &)

  Assignment stub operatror.

Additional Inherited Members

7.29.1 Detailed Description

Holds configuration for rendering a graph tile.

It represents an arbitrary configuration for a graph image slice.

Todo 3D require rotation of view. (quaternions)

7.29.2 Constructor & Destructor Documentation

7.29.2.1 ccdvl::GraphTileState::GraphTileState (int8_t *dimensions*, int32_t *width*, int32_t *height*)

Constructs a default graph tile state.

Parameters

in	*dimensions*	The number of dimensions. Must be positive and larger then zero.
	width	The buffer image width.
	height	The buffer image height.

7.29.2.2 ccdvl::GraphTileState::GraphTileState (const GraphSceneState & *graph_scene_state*) [explicit]

Constructor that uses the values from the provided graph scene state.

Parameters

in	*graph_scene_state*	The graph scene state to use.

7.29.3 Member Function Documentation

7.29.3.1 bool ccdvl::GraphTileState::LessThan (const GraphTileState & *graphstate*) const

Less-than comparator for this class.

Parameters

in	*graphstate*	Graph tile state to compare with.

Returns

True iff this graph tile state is "less-than" the argument.

7.29.3.2 void ccdvl::GraphTileState::operator= (const GraphTileState &) [private]

Assignment stub operatror.

Dissallows assignment of this class type.

7.29.4 Member Data Documentation

7.29.4.1 uint8_t ccdvl::GraphTileState::clear_color_[3]

Graph clear color.

Standard RGB888 color.

Note

Deviation from code standard, unsigned. Qt's QColor accepts signed values while AGG expects unsigned 8bit integers.
Keeping color as unsigned 8bit integers avoids using regular integers and extra typecasting.

The documentation for this class was generated from the following files:

- include/graph_tile_state.h
- src/graph_tile_state.cc

7.30 ccdvl::GraphTileState::functor_compare Struct Reference

Functor to compare graph tile states.

`#include <graph_tile_state.h>`

Public Member Functions

- bool operator() (const GraphTileState *lhs, const GraphTileState *rhs) const

  Less-than replacement operator.

7.30.1 Detailed Description

Functor to compare graph tile states.

7.30.2 Member Function Documentation

7.30.2.1 bool ccdvl::GraphTileState::functor_compare::operator() (const GraphTileState * *lhs*, const GraphTileState * *rhs*) const

Less-than replacement operator.

Parameters

in	*lhs*	Argument on left hand side of operator.
in	*rhs*	Argument on right hand side of operator.

Returns

True iff lhs is less than rhs.

The documentation for this struct was generated from the following files:

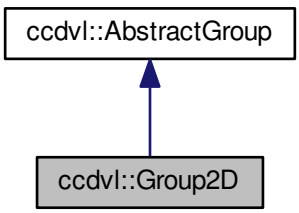
- include/graph_tile_state.h
- src/graph_tile_state.cc

7.31 ccdvl::Group2D Class Reference

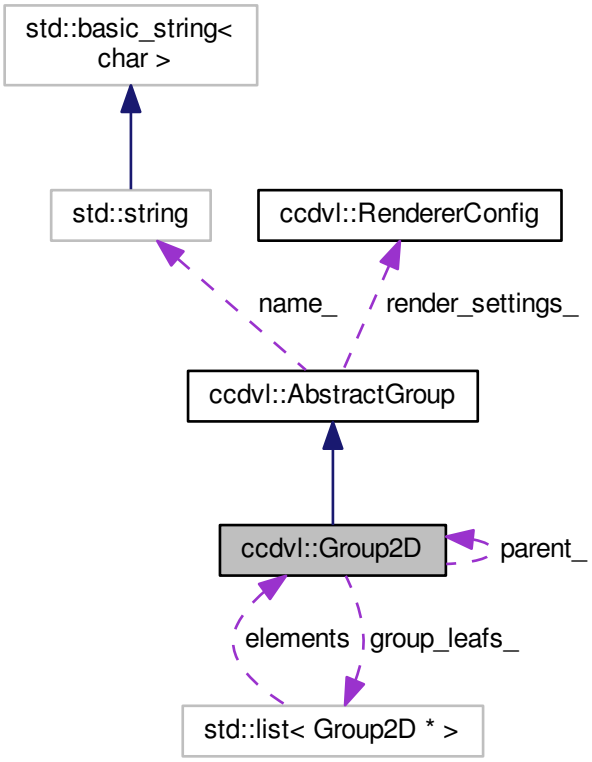
A 2D group selection of graph space; or simply a polygon.

`#include <group_2d.h>`

Inheritance diagram for ccdvl::Group2D:



Collaboration diagram for ccdvl::Group2D:



Public Member Functions

- Group2D ()

  Construct a 2D selection group.
- Group2D (const GraphPairVector &vertices)

  Construct a 2D selection group which operates in two dimensions.
- ~Group2D ()

  Destroys the object and frees any allocated resources.
- virtual bool PointInGroup (const GraphPoint &point) const

  Test if a point is within/selected by this group.
- virtual void GetBoundingBox (double *box) const

  Compute selection polygon bounding box.
- virtual const std::list< const AbstractGroup * > & GetLeafs () const

  Get the list of leaf group nodes.
- int32_t AddSubGroup (Group2D *g)

  Add a group selection within this group.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (Group2D)

- bool PointInPolygon (const double *vertex_x, const double *vertex_y, int32_t vertex_count, double test_x, double test_y) const

  Internal point-in-polygon algorithm.

Private Attributes

- std::list< Group2D * > group_leafs_

  List of group leafs.
- Group2D * parent_

  Group parent.
- int32_t vertex_count_

  Number of vertices in selection polygon.
- GraphDouble * vertex_x_

  X-coordinates in selection polygon.
- GraphDouble * vertex_y_

  Y-coordinates in selection polygon.

Additional Inherited Members

7.31.1 Detailed Description

A 2D group selection of graph space; or simply a polygon.

Todo Implement proper set operations for groups.

7.31.2 Constructor & Destructor Documentation

7.31.2.1 ccdvl::Group2D::Group2D (const GraphPairVector & *vertices*) [explicit]

Construct a 2D selection group which operates in two dimensions.

Parameters

vertices	The vertex vector representing a 2D polygon.

7.31.3 Member Function Documentation

7.31.3.1 int32_t ccdvl::Group2D::AddSubGroup (Group2D * *g*)

Add a group selection within this group.

Sub-Groups may not overlap.

Returns

zero on success.

Note

Basically a stub implementation; avoid use and implement proper set operations for groups instead.

7.31.3.2 void ccdvl::Group2D::GetBoundingBox (double * *box*) const [virtual]

Compute selection polygon bounding box.

Parameters

out	*box*	The polygon bounding box. Array pointer given as x_start, y_start, ..., k_start, x_end, y_end, ..., k_end So its length is equal to dimensions_ times two.

Implements ccdvl::AbstractGroup.

7.31.3.3 const std::list< const **AbstractGroup** * > & ccdvl::Group2D::GetLeafs ( ) const [virtual]

Get the list of leaf group nodes.

Returns

The std::list of leafs.

Implements ccdvl::AbstractGroup.

7.31.3.4 bool ccdvl::Group2D::PointInGroup (const **GraphPoint** & *point*) const [virtual]

Test if a point is within/selected by this group.

Parameters

in	*point*	A point of dimensions_ dimensions to test.

Returns

True iff the point is within/selected by this group.

Implements ccdvl::AbstractGroup.

7.31.3.5 bool ccdvl::Group2D::PointInPolygon (const double * *vertex_x*, const double * *vertex_y*, int32_t *vertex_count*, double *test_x*, double *test_y*) const [private]

Internal point-in-polygon algorithm.

Parameters

in	*vertex_x*	X-coordinates of vertices.
in	*vertex_y*	Y-coordinates of vertices.
	vertex_count	Number of vertices.
	test_x	X-coordinate of point to test.
	test_y	Y-coordinate of point to test.

7.31.4 Member Data Documentation

7.31.4.1 std::list<**Group2D***> ccdvl::Group2D::group_leafs_ [private]

List of group leafs.

These leaf groups polygons which must not intersect, since they are hierarchical.

The documentation for this class was generated from the following files:

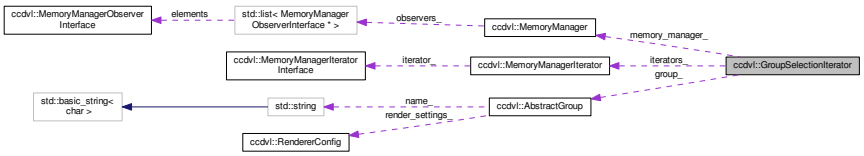
- include/group_2d.h
- src/group_2d.cc

7.32 ccdvl::GroupSelectionIterator Class Reference

Generic group selection iterator which fetches points selected by a group.

`#include <group_selection_iterator.h>`

Collaboration diagram for ccdvl::GroupSelectionIterator:



Public Member Functions

- GroupSelectionIterator ()

  Constructs a new invalid group iterator.
- GroupSelectionIterator (MemoryManager ∗memory_manager, MemoryManagerIterator memory_manager_-iterator, const AbstractGroup ∗group)

  Constructs and initialize a new iterator.
- GroupSelectionIterator (const GroupSelectionIterator &instance)

  Basic copy constructor.
- ∼GroupSelectionIterator ()

  Destroys the object and frees any allocated resources.
- GroupSelectionIterator & operator= (const GroupSelectionIterator &iterator)

  Basic assignment operator.
- GroupSelectionIterator & operator++ ()

  Basic post-increment operator.
- GroupSelectionIterator operator++ (int32_t)

  Basic pre-increment operator.
- bool operator== (const GroupSelectionIterator &rhs)

  Equality test.
- bool operator!= (const GroupSelectionIterator &rhs)

  Inequality test.
- std::vector< GraphDouble > operator∗ ()

  Dereference operator.

Private Attributes

- int8_t dimensions_

  Number of dimensions of data in memory manager.
- MemoryManager ∗ memory_manager_

  The memory manager to search in.

- const AbstractGroup * group_

  The group selection to search for.
- size_t at_

  The data point location in dataset.
- MemoryManagerIterator * iterators_

  The memory manager iterator that point to the current dataset.

7.32.1 Detailed Description

Generic group selection iterator which fetches points selected by a group.

Changing the group or memory manager used is not supported. Adding data to the memorymanager could cause data to be rearranged and the result may include double selection of data points or could case some data points to be lost. This iterator is also meant to be similar to std:iterator.

7.32.2 Constructor & Destructor Documentation

7.32.2.1 ccdvl::GroupSelectionIterator::GroupSelectionIterator ( )

Constructs a new invalid group iterator.

Intended to be assigned a real position.

7.32.2.2 ccdvl::GroupSelectionIterator::GroupSelectionIterator (MemoryManager * *memory_manager,* MemoryManagerIterator *memory_manager_iterator,* const AbstractGroup * *group*)

Constructs and initialize a new iterator.

Neither Memory manager; memory manager iterator or group instance should be destroyed before this iterator.

Parameters

in	*memory_manager*	The memory manager to select from.
in	*memory_manager_iterator*	The memory manager iterator to begin in.
in	*group*	The group to search from.

7.32.2.3 ccdvl::GroupSelectionIterator::GroupSelectionIterator (const GroupSelectionIterator & *instance*)

Basic copy constructor.

Creates a new copy of the provided group selection iterator.

Parameters

in	*instance*	The instance to copy.

7.32.3 Member Function Documentation

7.32.3.1 bool ccdvl::GroupSelectionIterator::operator!= (const GroupSelectionIterator & *rhs*)

Inequality test.

Test if two iterators references the same data location.

Returns

False iff both iterators point to the same data location.

7.32.3.2 AbstractGroup::GraphPoint ccdvl::GroupSelectionIterator::operator∗ ( )

Dereference operator.

Returns

The data point at the current data location.

7.32.3.3 GroupSelectionIterator & ccdvl::GroupSelectionIterator::operator++ ( )

Basic post-increment operator.

Increments the iterator to the next location.

Returns

The incremented iterator (this).

7.32.3.4 GroupSelectionIterator ccdvl::GroupSelectionIterator::operator++ (int32_t)

Basic pre-increment operator.

Increments the iterator to the next location.

Returns

The current (before it is incremented) iterator.

7.32.3.5 GroupSelectionIterator & ccdvl::GroupSelectionIterator::operator= (const GroupSelectionIterator & *iterator*)

Basic assignment operator.

Reassigns the internal state from the provided iterator.

Parameters

in	*iterator*	The iterator to copy from.

7.32.3.6 bool ccdvl::GroupSelectionIterator::operator== (const GroupSelectionIterator & *rhs*)

Equality test.

Test if two iterators references the same data location.

Returns

True iff both iterators point to the same data location.

The documentation for this class was generated from the following files:

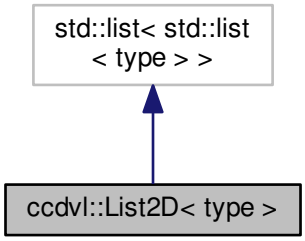
- include/group_selection_iterator.h
- src/group_selection_iterator.cc

7.33 ccdvl::List2D< type > Class Template Reference

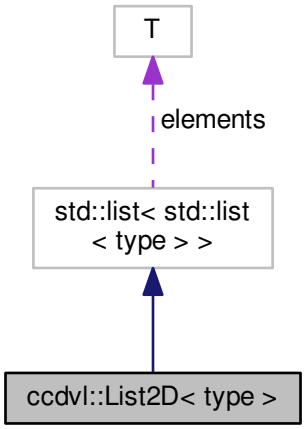
A two dimensional standard C++ list template.

`#include <list_2d.h>`

Inheritance diagram for ccdvl::List2D< type >:



Collaboration diagram for ccdvl::List2D< type >:



Public Member Functions

- List2D ()

  Constructs an empty two dimensional list.
- List2D (int rows, int columns, const type &value=type())

  Constructs a two dimensional list with the given number of rows and columns.

Additional Inherited Members

7.33.1 Detailed Description

template<typename type>class ccdvl::List2D< type >

A two dimensional standard C++ list template.

7.33.2 Constructor & Destructor Documentation

7.33.2.1 template<typename type> ccdvl::List2D< type >::List2D (int *rows*, int *columns*, const type & *value* = type()) [inline]

Constructs a two dimensional list with the given number of *rows* and *columns*.

Parameters

in	*rows*	The number of rows.
in	*columns*	The number of columns.
in	*value*	Default fill value.

The documentation for this class was generated from the following file:

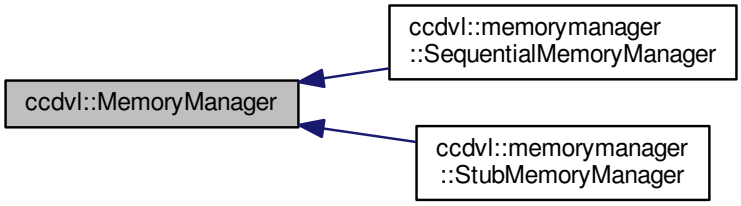
- include/list_2d.h

7.34 ccdvl::MemoryManager Class Reference

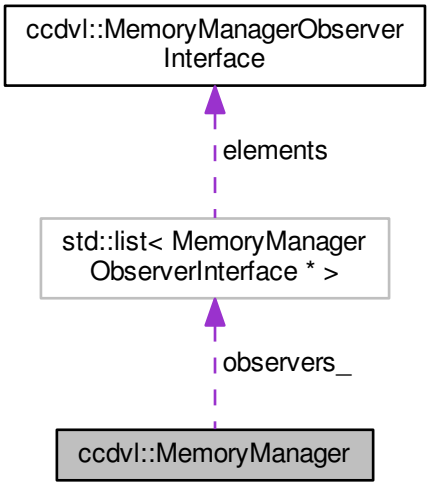
Abstract class for arbitrary memory managers.

`#include <memory_manager.h>`

Inheritance diagram for ccdvl::MemoryManager:



Collaboration diagram for ccdvl::MemoryManager:



Public Types

- enum MemoryManagerError { kEUnkown = -1, kEOk = 0, kENotSupported, kEOutOfMemory }

  Enumeration of error codes.
- typedef MemoryManagerIterator iterator

  The iterator type for any memory manager.

Public Member Functions

- MemoryManager (int8_t dimensions, bool search_support)

  Initialize constants for subclasses.
- virtual ~MemoryManager ()

  Destroys the object and frees any allocated resources.
- void AddObserver (MemoryManagerObserverInterface *observer)

  Adds an observer for new data.
- virtual MemoryManagerError AddData (const AbstractDataSet **data)=0

  Adds data.
- virtual MemoryManagerError Clear ()=0

  Remove all managed data.
- virtual iterator begin ()=0

  Get an iterator for the full dataset managed.
- virtual iterator GetRange (const std::vector< GraphDouble > &start, const std::vector< GraphDouble > &stop)=0

  Get an iterator for a managed data subset.
- virtual iterator end ()=0

  Get an iterator pointing past the last element of the dataset.

Public Attributes

- const int8_t dimensions_

  Number of dimensions supported.
- const bool search_support_

  True iff GetRange is fully supported by this memory manager.

Protected Member Functions

- void NotifyCleared ()

  Notifies all observers that all data was cleared.
- void NotifyNew (const AbstractDataSet **new_data)

  Notifies all observers that a new dataset was added.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (MemoryManager)

Private Attributes

- std::list
  < MemoryManagerObserverInterface * > observers_

  List of registered observers.

7.34.1 Detailed Description

Abstract class for arbitrary memory managers.

A memory manager manage memory allocation and storage for datasets.

7.34.2 Member Enumeration Documentation

7.34.2.1 enum ccdvl::MemoryManager::MemoryManagerError

Enumeration of error codes.

Enumerator:

kEUnkown An unknown error.

kEOk No error.

kENotSupported Operation not supported.

kEOutOfMemory An out-of-memory error occurred (possibly no enough virtual memory) or not enough disk space.

7.34.3 Constructor & Destructor Documentation

7.34.3.1 ccdvl::MemoryManager::MemoryManager (int8_t *dimensions*, bool *search_support*)

Initialize constants for subclasses.

Parameters

dimensions	The number of dimensions. Must be positive and larger then zero.
search_support	Flag which indicates if a memory manager support the GetRange method properly.

7.34.3.2 ccdvl::MemoryManager::~MemoryManager ( ) [virtual]

Destroys the object and frees any allocated resources.

Invalidates all iterators using this memory manager. Excluding the data swap file; if used.

7.34.4 Member Function Documentation

7.34.4.1 virtual MemoryManagerError ccdvl::MemoryManager::AddData (const AbstractDataSet ** *data*) [pure virtual]

Adds data.

This method will copy provided input.

Parameters

in	*data*	The data to manage, should be an array and its length must match dimensions_.

Returns

A MemoryManagerError code, kEOk iff the operation was successful.

Todo Change parameter to std::vector<const AbstractDataSet*>.

Implemented in ccdvl::memorymanager::SequentialMemoryManager, and ccdvl::memorymanager::StubMemoryManager.

7.34.4.2 void ccdvl::MemoryManager::AddObserver (MemoryManagerObserverInterface * *observer*)

Adds an observer for new data.

Observers can be safely destroyed as long as no additional data is added or Clear() is invoked.

Parameters

in	*observer*	A new observer to inform.

7.34.4.3 virtual iterator ccdvl::MemoryManager::begin ( ) [pure virtual]

Get an iterator for the full dataset managed.

Returns

An iterator for all data managed.

Implemented in ccdvl::memorymanager::SequentialMemoryManager, and ccdvl::memorymanager::StubMemoryManager.

7.34.4.4 virtual MemoryManagerError ccdvl::MemoryManager::Clear ( ) [pure virtual]

Remove all managed data.

Memory will also be freed.

Returns

A MemoryManagerError code, kEOk iff the operation was successful.

Implemented in ccdvl::memorymanager::SequentialMemoryManager, and ccdvl::memorymanager::StubMemoryManager.

7.34.4.5 virtual iterator ccdvl::MemoryManager::end ( ) [pure virtual]

Get an iterator pointing past the last element of the dataset.

Returns

An iterator which is pointing past the last element.

Implemented in ccdvl::memorymanager::SequentialMemoryManager, and ccdvl::memorymanager::StubMemoryManager.

7.34.4.6 virtual iterator ccdvl::MemoryManager::GetRange (const std::vector< GraphDouble > & *start*, const std::vector< GraphDouble > & *stop*) [pure virtual]

Get an iterator for a managed data subset.

Any memory manager fully supporting this must set search_support_ to true.

This will imply some restrictions on how data sets are ordered and it has not been decided how to address this.

Parameters

in	*start*	The start point of the bounding box for elements to iterate, its length must be equal to the number of dimensions.
in	*stop*	The stop point of the bounding box for elements to iterate, its length must be equal to the number of dimensions.

Returns

An iterator which iterates all elements within the provided bounding box.

Implemented in ccdvl::memorymanager::SequentialMemoryManager, and ccdvl::memorymanager::StubMemoryManager.

7.34.4.7 void ccdvl::MemoryManager::NotifyNew (const AbstractDataSet ** *new_data*) [protected]

Notifies all observers that a new dataset was added.

Parameters

in	*new_data*	The newly added dataset.

The documentation for this class was generated from the following files:

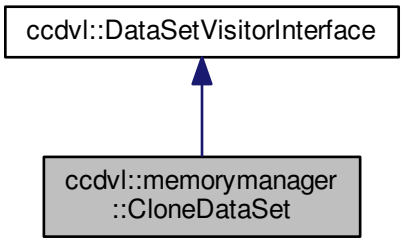
- include/memory_manager.h
- src/memory_manager.cc

7.35 ccdvl::memorymanager::CloneDataSet Class Reference

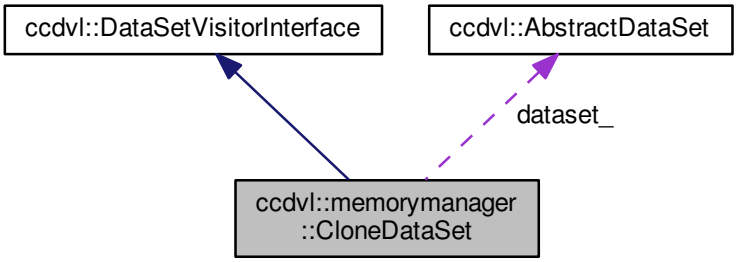
AbstractDataSet clone allocator for SequentialMemoryManager.

`#include <clone_data_set.h>`

Inheritance diagram for ccdvl::memorymanager::CloneDataSet:



Collaboration diagram for ccdvl::memorymanager::CloneDataSet:



Public Member Functions

- virtual void Visit (TypedDataSet< uint8_t > *uint8)
  Callback for TypedDataSet holding uint8_t.
- virtual void Visit (TypedDataSet< int8_t > *int8)
  Callback for TypedDataSet holding int8_t.
- virtual void Visit (TypedDataSet< uint16_t > *uint16)
  Callback for TypedDataSet holding uint16_t.
- virtual void Visit (TypedDataSet< int16_t > *int16)
  Callback for TypedDataSet holding int16_t.
- virtual void Visit (TypedDataSet< uint32_t > *uint32)
  Callback for TypedDataSet holding uint32_t.
- virtual void Visit (TypedDataSet< int32_t > *int32)
  Callback for TypedDataSet holding int32_t.

- virtual void Visit (TypedDataSet< uint64_t > *uint64)

  Callback for TypedDataSet holding uint64_t.
- virtual void Visit (TypedDataSet< int64_t > *int64)

  Callback for TypedDataSet holding int64_t.
- virtual void Visit (TypedDataSet< float > *tfloat)

  Callback for TypedDataSet holding regular floats.
- virtual void Visit (TypedDataSet< double > *tdouble)

  Callback for TypedDataSet holding regular doubles.
- CloneDataSet (void **destination)

  Constructs a new AbstractDataSet cloning visitor.
- ~CloneDataSet ()

  Destroys the object and frees any allocated resources.
- void SetCopyDestination (AbstractDataSet **dataset)

  Sets return location for new datasets.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (CloneDataSet)
- template<typename T >
  void Visit (TypedDataSet< T > *t)

  Template aggregation of the overloaded visit method.

Private Attributes

- void ** destination_

  Pointer to memory allocation.
- ptrdiff_t written_

  Bytes of memory already consumed.
- AbstractDataSet ** dataset_

  Destination for new dataset.

7.35.1 Detailed Description

AbstractDataSet clone allocator for SequentialMemoryManager.

Copies AbstractDataSet objects into provided memory areas.

See also

SequentialMemoryManager and AbstractDataSet.

7.35.2 Constructor & Destructor Documentation

7.35.2.1 ccdvl::memorymanager::CloneDataSet::CloneDataSet (void ** *destination*) [explicit]

Constructs a new AbstractDataSet cloning visitor.

Parameters

out	*destination*	memory pointer.

7.35.3 Member Function Documentation

7.35.3.1 void ccdvl::memorymanager::CloneDataSet::SetCopyDestination (AbstractDataSet ** *dataset*)

Sets return location for new datasets.

Parameters

out	*dataset*	Destination pointer.

7.35.3.2 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< uint8_t > * *uint8*) [virtual]

Callback for TypedDataSet holding uint8_t.

Parameters

in,out	*uint8*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.3 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< int8_t > * *int8*) [virtual]

Callback for TypedDataSet holding int8_t.

Parameters

in,out	*int8*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.4 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< uint16_t > * *uint16*) [virtual]

Callback for TypedDataSet holding uint16_t.

Parameters

in,out	*uint16*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.5 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< int16_t > * *int16*) [virtual]

Callback for TypedDataSet holding int16_t.

Parameters

in,out	*int16*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.6 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< uint32_t > * *uint32*) [virtual]

Callback for TypedDataSet holding uint32_t.

Parameters

in,out	*uint32*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.7 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< int32_t > * *int32*) [virtual]

Callback for TypedDataSet holding int32_t.

Parameters

in,out	*int32*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.8 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< uint64_t > * *uint64*) [virtual]

Callback for TypedDataSet holding uint64_t.

Parameters

in,out	*uint64*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.9 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< int64_t > * *int64*) [virtual]

Callback for TypedDataSet holding int64_t.

Parameters

in,out	*int64*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.10 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< float > * *tfloat*) [virtual]

Callback for TypedDataSet holding regular floats.

Parameters

in,out	*tfloat*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.11 void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< double > * *tdouble*) [virtual]

Callback for TypedDataSet holding regular doubles.

Parameters

in,out	*tdouble*	Calling class instance.

Implements ccdvl::DataSetVisitorInterface.

7.35.3.12 template<typename T > void ccdvl::memorymanager::CloneDataSet::Visit (TypedDataSet< T > * t) [inline], [private]

Template aggregation of the overloaded visit method.

Template Parameters

T	Numerical data type held by a TypedDataSet to handle.

Parameters

in	*t*	TypedDataSet to visit.

See also

TypedDataSet

Note

Reduces copy-paste code.

The documentation for this class was generated from the following files:

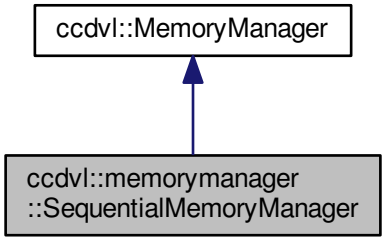
- include/sequential_memory_manager/clone_data_set.h
- src/sequential_memory_manager/clone_data_set.cc

7.36 ccdvl::memorymanager::SequentialMemoryManager Class Reference

A regular memory manager which stores data to be accessed in sequence.

`#include <sequential_memory_manager.h>`

Inheritance diagram for ccdvl::memorymanager::SequentialMemoryManager:



Collaboration diagram for ccdvl::memorymanager::SequentialMemoryManager:

Classes

- struct MappedMemory

  Memory map structure, including matching datasets.

Public Types

- typedef std::list< MappedMemory * > MemoryAllocationList

  List type for allocated memory.
- typedef MemoryAllocationList::iterator MemoryAllocationListIterator

  Iterator type for allocated memory.

Public Member Functions

- SequentialMemoryManager (int8_t dimensions)

  Constructs a new SequentialMemoryManager.
- SequentialMemoryManager::MemoryManagerError Init ()

  Initializes this memory manager in anonymous swap mode.
- SequentialMemoryManager::MemoryManagerError Init (const char *file)

  Initializes this memory manager in file swap mode.
- virtual ~SequentialMemoryManager ()

  Destroys the object and frees any allocated resources.
- virtual MemoryManagerError AddData (const AbstractDataSet **data)

  Adds data.
- virtual MemoryManagerError Clear ()

  Remove all managed data.
- virtual iterator begin ()

  Get an iterator for the full dataset managed.
- virtual iterator end ()

  Get an iterator pointing past the last element of the dataset.
- virtual iterator GetRange (const std::vector< GraphDouble > &start, const std::vector< GraphDouble > &stop)

  Get an iterator for a managed data subset.
- void PageControlLoad (MappedMemory &mm)

  Informs OS that a certain MappedMemory area must be loaded.
- void PageControlUnload (MappedMemory &mm)

  Informs OS that a certain MappedMemory area is no longer needed.

Public Attributes

- MemoryAllocationList mapped_space_

  List of all data buckets, and their allocation if any.

Private Member Functions

- void PageControlAllocate (MappedMemory &mm)

  Create a new MappedMemory area or acquire a new allocation for one.
- void PageControlDeallocate (MappedMemory &mm)

  Deallocate a previously allocated MappedMemory area.
- void PageControlDelete (MappedMemory &mm)

  Deletes a MappedMemory area.

Private Attributes

- size_t page_size_

  OS page size.
- bool anonymous_swap_

  Memory swap mode.
- int32_t memory_map_flags_

  mmap allocation flags.
- int32_t file_descriptor_

  Descriptor for accessing the swap file, if any.
- pthread_mutex_t load_mutex_

  Data load/unload mutex.

Additional Inherited Members

7.36.1 Detailed Description

A regular memory manager which stores data to be accessed in sequence.

It uses a reference counting iterator, this allows data to be (flushed to disk and) deallocated when its no longer needed.

7.36.2 Constructor & Destructor Documentation

7.36.2.1 ccdvl::memorymanager::SequentialMemoryManager::SequentialMemoryManager (int8_t *dimensions*) [explicit]

Constructs a new SequentialMemoryManager.

Remember to invoke one of the initialization methods.

Parameters

dimensions	The number of dimensions. Must be positive and larger then zero.

See also

Init()

7.36.3 Member Function Documentation

7.36.3.1 SequentialMemoryManager::MemoryManagerError ccdvl::memorymanager::SequentialMemoryManager::AddData (const AbstractDataSet ** *data*) [virtual]

Adds data.

This method will copy provided input.

Parameters

in	*data*	The data to manage, should be an array and its length must match dimensions_.

Returns

A MemoryManagerError code, kEOk iff the operation was successful.

Todo Change parameter to std::vector<const AbstractDataSet*>.

Implements ccdvl::MemoryManager.

7.36.3.2 SequentialMemoryManager::iterator ccdvl::memorymanager::SequentialMemoryManager::begin ( ) [virtual]

Get an iterator for the full dataset managed.

Returns

An iterator for all data managed.

Implements ccdvl::MemoryManager.

7.36.3.3 SequentialMemoryManager::MemoryManagerError ccdvl::memorymanager::SequentialMemoryManager::Clear ( ) [virtual]

Remove all managed data.

Memory will also be freed.

Returns

A MemoryManagerError code, kEOk iff the operation was successful.

Implements ccdvl::MemoryManager.

7.36.3.4 SequentialMemoryManager::iterator ccdvl::memorymanager::SequentialMemoryManager::end ( ) [virtual]

Get an iterator pointing past the last element of the dataset.

Returns

An iterator which is pointing past the last element.

Implements ccdvl::MemoryManager.

7.36.3.5 SequentialMemoryManager::iterator ccdvl::memorymanager::SequentialMemoryManager::GetRange (const std::vector< GraphDouble > & *start*, const std::vector< GraphDouble > & *stop*) [virtual]

Get an iterator for a managed data subset.

Any memory manager fully supporting this must set search_support_ to true.

This will imply some restrictions on how data sets are ordered and it has not been decided how to address this.

Parameters

in	*start*	The start point of the bounding box for elements to iterate, its length must be equal to the number of dimensions.
in	*stop*	The stop point of the bounding box for elements to iterate, its length must be equal to the number of dimensions.

Returns

An iterator which iterates all elements within the provided bounding box.

This method return the same iterator as begin.

Implements ccdvl::MemoryManager.

7.36.3.6 SequentialMemoryManager::MemoryManagerError ccdvl::memorymanager::SequentialMemoryManager::Init (const char * *file*)

Initializes this memory manager in file swap mode.

Parameters

in	*file*	Swap filename.

7.36.3.7 void ccdvl::memorymanager::SequentialMemoryManager::PageControlAllocate (MappedMemory & *mm*) [private]

Create a new MappedMemory area or acquire a new allocation for one.

Don't bother with this, use the iterator instead.

Parameters

in,out	*mm*	MappedMemory to allocate.

7.36.3.8 void ccdvl::memorymanager::SequentialMemoryManager::PageControlDeallocate (MappedMemory & *mm*) [private]

Deallocate a previously allocated MappedMemory area.

If possible, pages will be flushed to disk and then removed to free virtual memory. Don't bother with this, use the iterator instead.

Parameters

in,out	*mm*	MappedMemory area to deallocate.

7.36.3.9 void ccdvl::memorymanager::SequentialMemoryManager::PageControlDelete (MappedMemory & *mm*) [private]

Deletes a MappedMemory area.

This will delete a MappedMemory area without disk flush, unlike PageControlDeallocate. Don't bother with this, use the iterator instead.

Parameters

in,out	*mm*	MappedMemory area to remove.

7.36.3.10 void ccdvl::memorymanager::SequentialMemoryManager::PageControlLoad (MappedMemory & *mm*)

Informs OS that a certain MappedMemory area must be loaded.

Don't bother with this, use the iterator instead. This also increases the reference counter.

Parameters

in,out	*mm*	MappedMemory area to load.

7.36.3.11 void ccdvl::memorymanager::SequentialMemoryManager::PageControlUnload (MappedMemory & *mm*)

Informs OS that a certain MappedMemory area is no longer needed.

Don't bother with this, use the iterator instead. This also decreases the reference counter.

Parameters

in	*mm*	MappedMemory area to unload.

7.36.4 Member Data Documentation

7.36.4.1 MemoryAllocationList ccdvl::memorymanager::SequentialMemoryManager::mapped_space_

List of all data buckets, and their allocation if any.

Note

This is public so that it can be accessed by the iterator.

Todo It is tempting to define SequentialMemoryManagerIterator a friend class, Alternatively provide the needed functionality as methods which is already true for page control methods.

The documentation for this class was generated from the following files:

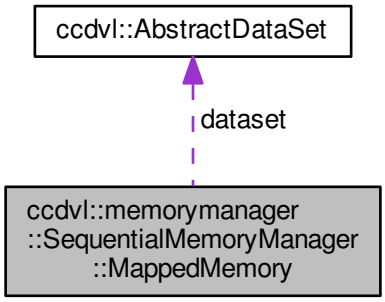
- include/sequential_memory_manager/sequential_memory_manager.h
- src/sequential_memory_manager/sequential_memory_manager.cc

7.37 ccdvl::memorymanager::SequentialMemoryManager::MappedMemory Struct Reference

Memory map structure, including matching datasets.

`#include <sequential_memory_manager.h>`

Collaboration diagram for ccdvl::memorymanager::SequentialMemoryManager::MappedMemory:



Public Attributes

- void * memory_pointer

  Memory pointer for dataset.
- size_t allocated_bytes

  Amount of bytes allocated for data.
- off_t cache_file_offset

  Offset to data in cache file.
- size_t reference_count

  Number of iterators using this dataset.
- AbstractDataSet ** dataset

7.37.1 Detailed Description

Memory map structure, including matching datasets.

7.37.2 Member Data Documentation

7.37.2.1 AbstractDataSet** ccdvl::memorymanager::SequentialMemoryManager::MappedMemory::dataset

Array of dataset instances stored in the managed memory area.

The documentation for this struct was generated from the following file:

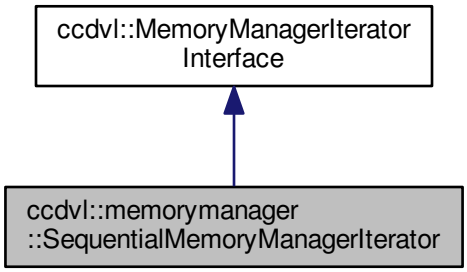
- include/sequential_memory_manager/sequential_memory_manager.h

7.38 ccdvl::memorymanager::SequentialMemoryManagerIterator Class Reference

An iterator used to iterate a SequentialMemoryManager.

`#include <sequential_memory_manager_iterator.h>`

Inheritance diagram for ccdvl::memorymanager::SequentialMemoryManagerIterator:



Collaboration diagram for ccdvl::memorymanager::SequentialMemoryManagerIterator:

Public Types

- typedef SequentialMemoryManager::MemoryAllocationListIterator MemoryAllocationListIterator

  A shorter iterator type of memory managers in class scope.

Public Member Functions

- SequentialMemoryManagerIterator (SequentialMemoryManager ∗mit, MemoryAllocationListIterator dit)

  Construct a new SequentialMemoryManagerIterator.
- virtual ∼SequentialMemoryManagerIterator ()

  Destroys the object and frees any allocated resources.
- virtual MemoryManagerIteratorInterface ∗ Clone () const

  Creates a copy of this iterator.
- virtual void Next ()

  Increment this iterator.
- virtual bool Equals (const MemoryManagerIteratorInterface ∗rhs) const

  Test if this iterator is the same as the one provided.
- virtual const AbstractDataSet ∗ Get () const

  Get current dataset.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (SequentialMemoryManagerIterator)

Private Attributes

- SequentialMemoryManager * sequential_manager_

  Reference to the stub memory manager iterated.
- MemoryAllocationListIterator bucket_iterator_

  Bucket iterator.
- int8_t at_

  Current dimension.

7.38.1 Detailed Description

An iterator used to iterate a SequentialMemoryManager.

Uses reference counting to ensure access to data.

7.38.2 Constructor & Destructor Documentation

7.38.2.1 ccdvl::memorymanager::SequentialMemoryManagerIterator::SequentialMemoryManagerIterator (SequentialMemoryManager * *mit*, MemoryAllocationListIterator *dit*)

Construct a new SequentialMemoryManagerIterator.

Parameters

in	*mit*	Sequential memory manager to iterate.
in	*dit*	Data allocation iterator to start at.

7.38.3 Member Function Documentation

7.38.3.1 MemoryManagerIteratorInterface * ccdvl::memorymanager::SequentialMemoryManagerIterator::Clone ( ) const [virtual]

Creates a copy of this iterator.

Returns

Pointer to new instance.

Implements ccdvl::MemoryManagerIteratorInterface.

7.38.3.2 bool ccdvl::memorymanager::SequentialMemoryManagerIterator::Equals (const MemoryManagerIteratorInterface * *rhs*) const [virtual]

Test if this iterator is the same as the one provided.

Parameters

rhs	[in] The iterator to compare with.

Returns

True if this iterator is equal to the one provided.

Implements ccdvl::MemoryManagerIteratorInterface.

7.38.3.3 const AbstractDataSet * ccdvl::memorymanager::SequentialMemoryManagerIterator::Get ( ) const [virtual]

Get current dataset.

Returns

The current dataset.

Implements ccdvl::MemoryManagerIteratorInterface.

The documentation for this class was generated from the following files:

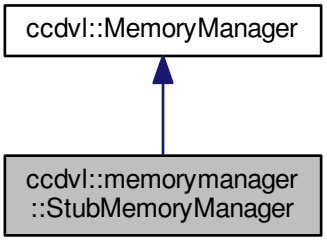
- include/sequential_memory_manager/sequential_memory_manager_iterator.h
- src/sequential_memory_manager/sequential_memory_manager_iterator.cc

7.39 ccdvl::memorymanager::StubMemoryManager Class Reference

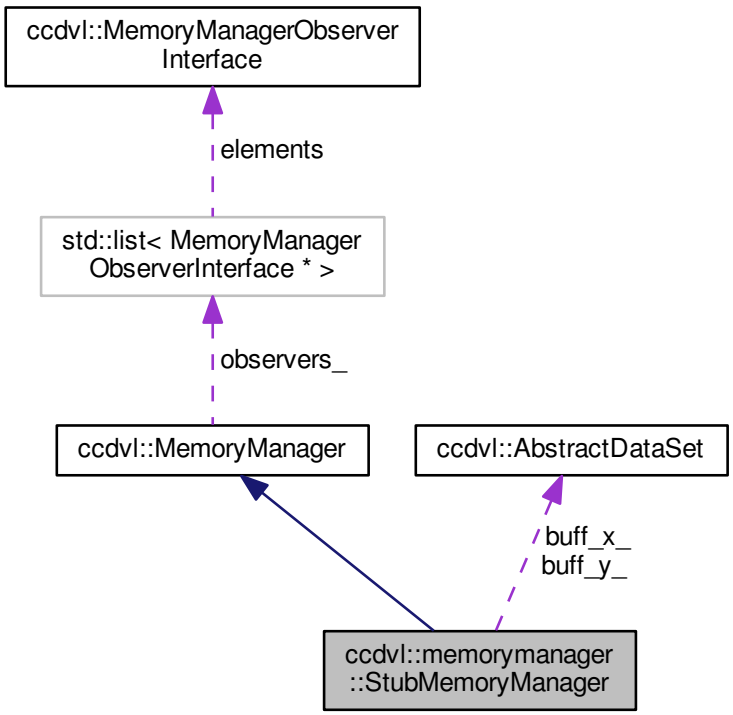
A MemoryManager which do not support add or clear.

`#include <stub_memory_manager.h>`

Inheritance diagram for ccdvl::memorymanager::StubMemoryManager:



Collaboration diagram for ccdvl::memorymanager::StubMemoryManager:



Public Member Functions

- virtual MemoryManagerError AddData (const AbstractDataSet **data)

  Adds data.
- virtual MemoryManagerError Clear ()

  Remove all managed data.
- virtual iterator begin ()

  Get an iterator for the full dataset managed.
- virtual iterator end ()

  Get an iterator pointing past the last element of the dataset.
- virtual iterator GetRange (const std::vector< GraphDouble > &start, const std::vector< GraphDouble > &stop)

  Get an iterator for a managed data subset.
- StubMemoryManager ()

  Initializes a new StubMemoryManager instance.
- virtual ~StubMemoryManager ()

  Destroys the object and frees any allocated resources.
- void SwitchTo (size_t n)

  Switch dataset held.

Public Attributes

- AbstractDataSet * buff_x_

  Active dataset x.
- AbstractDataSet * buff_y_

  Active dataset y.
- size_t current_

  Index of active dataset.

Additional Inherited Members

7.39.1 Detailed Description

A MemoryManager which do not support add or clear.

Generates a static dataset for rederering tests.

7.39.2 Member Function Documentation

7.39.2.1 MemoryManager::MemoryManagerError ccdvl::memorymanager::StubMemoryManager::AddData (const AbstractDataSet ** *data*) [virtual]

Adds data.

This method will copy provided input.

Parameters

in	*data*	The data to manage, should be an array and its length must match dimensions_.

Returns

A MemoryManagerError code, kEOk iff the operation was successful.

Todo Change parameter to std::vector<const AbstractDataSet*>.

Note

This method is an empty stub.

Implements ccdvl::MemoryManager.

7.39.2.2 StubMemoryManager::iterator ccdvl::memorymanager::StubMemoryManager::begin ( ) [virtual]

Get an iterator for the full dataset managed.

Returns

An iterator for all data managed.

Implements ccdvl::MemoryManager.

7.39.2.3 MemoryManager::MemoryManagerError ccdvl::memorymanager::StubMemoryManager::Clear ( ) [virtual]

Remove all managed data.

Memory will also be freed.

Returns

A MemoryManagerError code, kEOk iff the operation was successful.

Note

This method is an empty stub.

Implements ccdvl::MemoryManager.

7.39.2.4 StubMemoryManager::iterator ccdvl::memorymanager::StubMemoryManager::end ( ) [virtual]

Get an iterator pointing past the last element of the dataset.

Returns

An iterator which is pointing past the last element.

Implements ccdvl::MemoryManager.

7.39.2.5 StubMemoryManager::iterator ccdvl::memorymanager::StubMemoryManager::GetRange (const std::vector< GraphDouble > & *start*, const std::vector< GraphDouble > & *stop*) [virtual]

Get an iterator for a managed data subset.

Any memory manager fully supporting this must set search_support_ to true.

This will imply some restrictions on how data sets are ordered and it has not been decided how to address this.

Parameters

in	*start*	The start point of the bounding box for elements to iterate, its length must be equal to the number of dimensions.
in	*stop*	The stop point of the bounding box for elements to iterate, its length must be equal to the number of dimensions.

Returns

An iterator which iterates all elements within the provided bounding box.

This method return the same iterator as begin.

Implements ccdvl::MemoryManager.

7.39.2.6 void ccdvl::memorymanager::StubMemoryManager::SwitchTo (size_t *n*)

Switch dataset held.

Parameters

n	The dataset index.

The documentation for this class was generated from the following files:

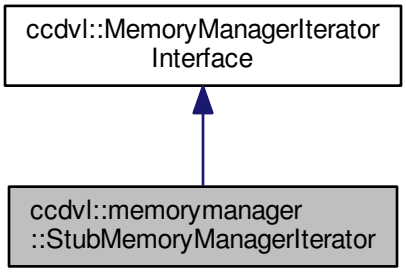
- include/stub_memory_manager/stub_memory_manager.h
- src/stub_memory_manager/stub_memory_manager.cc

7.40 ccdvl::memorymanager::StubMemoryManagerIterator Class Reference

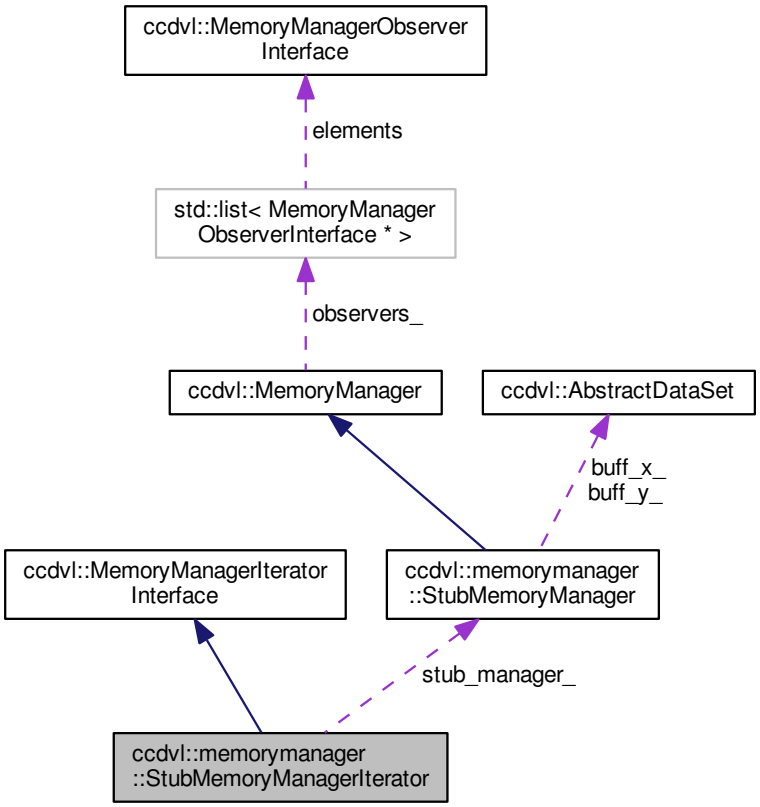
An iterator used it iterate data in a StubMemoryManager.

`#include <stub_memory_manager_iterator.h>`

Inheritance diagram for ccdvl::memorymanager::StubMemoryManagerIterator:



Collaboration diagram for ccdvl::memorymanager::StubMemoryManagerIterator:



Public Member Functions

- StubMemoryManagerIterator ()

  Initializes an invalid iterator.
- StubMemoryManagerIterator (StubMemoryManager ∗mit)

  Construct a new iterator.
- virtual ∼StubMemoryManagerIterator ()

  Destroys the object and frees any allocated resources.
- virtual MemoryManagerIteratorInterface ∗ Clone () const

  Creates a copy of this iterator.
- virtual void Next ()

  Increment this iterator.
- virtual bool Equals (const MemoryManagerIteratorInterface ∗rhs) const

  Test if this iterator is the same as the one provided.
- virtual const AbstractDataSet ∗ Get () const

  Get current dataset.
- void End ()

  Sets the iterator to point past the last element.

Private Attributes

- StubMemoryManager * stub_manager_

  Reference to the stub memory manager iterated.
- size_t at_

  Current dataset index.

7.40.1 Detailed Description

An iterator used it iterate data in a StubMemoryManager.

7.40.2 Constructor & Destructor Documentation

7.40.2.1 ccdvl::memorymanager::StubMemoryManagerIterator::StubMemoryManagerIterator (StubMemoryManager * *mit*) [explicit]

Construct a new iterator.

Parameters

in	*mit*	The Stub memory manager to iterate.

7.40.3 Member Function Documentation

7.40.3.1 MemoryManagerIteratorInterface * ccdvl::memorymanager::StubMemoryManagerIterator::Clone ( ) const [virtual]

Creates a copy of this iterator.

Returns

Pointer to new instance.

Implements ccdvl::MemoryManagerIteratorInterface.

7.40.3.2 bool ccdvl::memorymanager::StubMemoryManagerIterator::Equals (const MemoryManagerIteratorInterface * *rhs*) const [virtual]

Test if this iterator is the same as the one provided.

Parameters

rhs	[in] The iterator to compare with.

Returns

True if this iterator is equal to the one provided.

Implements ccdvl::MemoryManagerIteratorInterface.

7.40.3.3 const AbstractDataSet * ccdvl::memorymanager::StubMemoryManagerIterator::Get ( ) const [virtual]

Get current dataset.

Returns

The current dataset.

Implements ccdvl::MemoryManagerIteratorInterface.

The documentation for this class was generated from the following files:

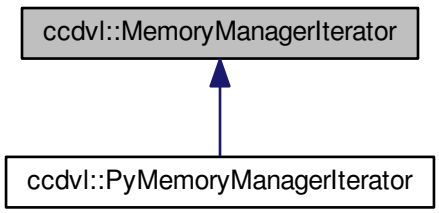
- include/stub_memory_manager/stub_memory_manager_iterator.h
- src/stub_memory_manager/stub_memory_manager_iterator.cc

7.41 ccdvl::MemoryManagerIterator Class Reference

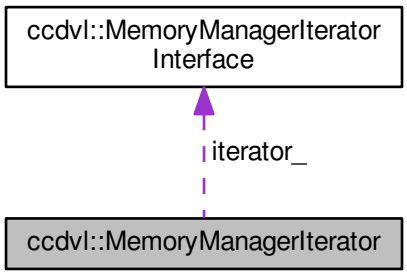
Wrapper class used to create std::iterator like objects.

`#include <memory_manager_iterator.h>`

Inheritance diagram for ccdvl::MemoryManagerIterator:



Collaboration diagram for ccdvl::MemoryManagerIterator:



Public Member Functions

- MemoryManagerIterator ()

  Constructs a new invalid memory manager iterator.

- MemoryManagerIterator (const MemoryManagerIterator &mit)

  Basic copy constructor.
- MemoryManagerIterator (MemoryManagerIteratorInterface *mit)

  Wraps an existing memory manager iterator interface.
- ~MemoryManagerIterator ()

  Destroys the object and frees any allocated resources.
- MemoryManagerIterator & operator= (const MemoryManagerIterator &mit)

  Basic assignment operator.
- MemoryManagerIterator & operator++ ()

  Basic post-increment operator.
- MemoryManagerIterator operator++ (int32_t)

  Basic pre-increment operator.
- bool operator== (const MemoryManagerIterator &rhs)

  Equality test.
- bool operator!= (const MemoryManagerIterator &rhs)

  Inequality test.
- const AbstractDataSet * operator* ()

  Dereference operator.

Private Attributes

- MemoryManagerIteratorInterface * iterator_

  The wrapped memory manager iterator interface.

7.41.1 Detailed Description

Wrapper class used to create std::iterator like objects.

Wraps a MemoryManagerIteratorInterface class to simplify usage and make them more like regular C++ iterators.

See also

MemoryManagerIteratorInterface.

Warning

Beware that memory managers may use reference counters and can therefore deallocate datasets as needed when no iterators are pointing to them. This is both to prevent dangling iterators between threads and to improve memory management, so all iterators to used datasets must persist.

7.41.2 Constructor & Destructor Documentation

7.41.2.1 ccdvl::MemoryManagerIterator::MemoryManagerIterator ( )

Constructs a new invalid memory manager iterator.

Intended to be assigned a real position.

7.41.2.2 ccdvl::MemoryManagerIterator::MemoryManagerIterator (const MemoryManagerIterator & *mit*)

Basic copy constructor.

Creates a new copy of the provided memory manager iterator.

Parameters

in	*mit*	The instance to copy.

7.41.2.3 ccdvl::MemoryManagerIterator::MemoryManagerIterator (MemoryManagerIteratorInterface * *mit*)

Wraps an existing memory manager iterator interface.

Creates a newly wrapped instance of the provided memory manager iterator interface.

Parameters

in	*mit*	The instance to wrap.

See also

MemoryManagerIteratorInterface.

7.41.3 Member Function Documentation

7.41.3.1 bool ccdvl::MemoryManagerIterator::operator!= (const MemoryManagerIterator & *rhs*)

Inequality test.

Test if two iterators references the same data location.

Returns

False iff both iterators point to the same data location.

7.41.3.2 const AbstractDataSet * ccdvl::MemoryManagerIterator::operator* ( )

Dereference operator.

Returns

The dataset at the current location.

7.41.3.3 MemoryManagerIterator & ccdvl::MemoryManagerIterator::operator++ ( )

Basic post-increment operator.

Increments the iterator to the next location.

Returns

The incremented iterator (this).

7.41.3.4 MemoryManagerIterator ccdvl::MemoryManagerIterator::operator++ (int32_t)

Basic pre-increment operator.

Increments the iterator to the next location.

Returns

The current (before it is incremented) iterator.

7.41.3.5 MemoryManagerIterator & ccdvl::MemoryManagerIterator::operator= (const MemoryManagerIterator & *mit*)

Basic assignment operator.

Reassigns the internal state from the provided iterator.

Parameters

in	*mit*	The iterator to copy from.

7.41.3.6 bool ccdvl::MemoryManagerIterator::operator== (const MemoryManagerIterator & *rhs*)

Equality test.

Test if two iterators references the same data location.

Returns

True iff both iterators point to the same data location.

The documentation for this class was generated from the following files:

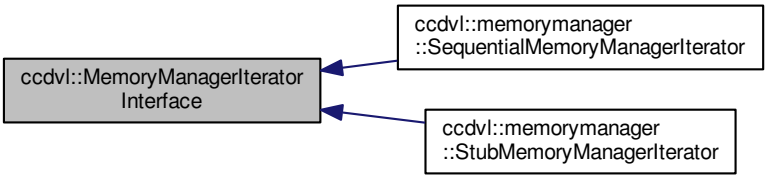
- include/memory_manager_iterator.h
- src/memory_manager_iterator.cc

7.42 ccdvl::MemoryManagerIteratorInterface Class Reference

A simple iterator interface for iterating various memory managers.

`#include <memory_manager_iterator.h>`

Inheritance diagram for ccdvl::MemoryManagerIteratorInterface:



Public Member Functions

- virtual ~MemoryManagerIteratorInterface ()

  Destroys the object and frees any allocated resources.
- virtual MemoryManagerIteratorInterface * Clone () const =0

  Creates a copy of this iterator.
- virtual void Next ()=0

  Increment this iterator.
- virtual bool Equals (const MemoryManagerIteratorInterface *rhs) const =0

  Test if this iterator is the same as the one provided.
- virtual const AbstractDataSet * Get () const =0

  Get current dataset.

7.42.1 Detailed Description

A simple iterator interface for iterating various memory managers.

This class is not meant to be used directly, rather it is supposed to be extended by the specific memory manager iterator and wrapped by the MemoryManagerIterator helper class.

Note

If this iterator is destroyed, the dataset returned by the Get() method may no longer be valid. This allows reference counters to be used for allocating and deallocating datasets as needed.

See also

MemoryManagerIterator

7.42.2 Member Function Documentation

7.42.2.1 virtual MemoryManagerIteratorInterface* ccdvl::MemoryManagerIteratorInterface::Clone ( ) const [pure virtual]

Creates a copy of this iterator.

Returns

Pointer to new instance.

Implemented in ccdvl::memorymanager::SequentialMemoryManagerIterator, and ccdvl::memorymanager::StubMemoryManagerIterator.

7.42.2.2 virtual bool ccdvl::MemoryManagerIteratorInterface::Equals (const MemoryManagerIteratorInterface * *rhs*) const [pure virtual]

Test if this iterator is the same as the one provided.

Parameters

rhs	[in] The iterator to compare with.

Returns

True if this iterator is equal to the one provided.

Implemented in ccdvl::memorymanager::SequentialMemoryManagerIterator, and ccdvl::memorymanager::StubMemoryManagerIterator.

7.42.2.3 virtual const AbstractDataSet* ccdvl::MemoryManagerIteratorInterface::Get ( ) const [pure virtual]

Get current dataset.

Returns

The current dataset.

Implemented in ccdvl::memorymanager::SequentialMemoryManagerIterator, and ccdvl::memorymanager::StubMemoryManagerIterator.

The documentation for this class was generated from the following file:

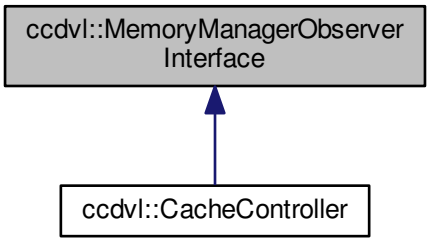
- include/memory_manager_iterator.h

7.43 ccdvl::MemoryManagerObserverInterface Class Reference

Observer for MemoryManager dataset update events.

`#include <memory_manager_observer_interface.h>`

Inheritance diagram for ccdvl::MemoryManagerObserverInterface:



Public Member Functions

- virtual ~MemoryManagerObserverInterface ()

  Destroys the object and frees any allocated resources.
- virtual void MemoryManagerUpdate (MemoryManager &callee, const AbstractDataSet **new_data)=0

  Update callback when more data is added.
- virtual void MemoryManagerCleared (MemoryManager &callee)=0

  Update callback when a MemoryManager is cleared.

7.43.1 Detailed Description

Observer for MemoryManager dataset update events.

7.43.2 Member Function Documentation

7.43.2.1 virtual void ccdvl::MemoryManagerObserverInterface::MemoryManagerCleared (MemoryManager & *callee*) [pure virtual]

Update callback when a MemoryManager is cleared.

Parameters

in	*callee*	MemoryManager that caused this update.

Implemented in ccdvl::CacheController.

7.43.2.2 virtual void ccdvl::MemoryManagerObserverInterface::MemoryManagerUpdate (MemoryManager & *callee*, const AbstractDataSet ** *new_data*) [pure virtual]

Update callback when more data is added.

Parameters

in	*callee*	MemoryManager that caused this update.
in	*new_data*	The newly added dataset.

Implemented in ccdvl::CacheController.

The documentation for this class was generated from the following file:

- include/memory_manager_observer_interface.h

7.44 ccdvl::MessageQueue< T > Class Template Reference

Simple message passing queue.

`#include <message_queue.h>`

Public Member Functions

- MessageQueue ()
  Initializes a new instance.
- virtual ~MessageQueue ()
  Destroys the object and frees any allocated resources.
- void SendMessage (T &message)
  Enqueue a message to the message queue.
- T & WaitForMessage ()
  Wait for a message.
- void WaitForAllMessages (std::list< T > &messages)
  Wait for atleast one message.
- void GetAllMessages (std::list< T > &messages)
  Get all messages.
- bool RemoveMessage (T &message)
  Remove a queued message.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (MessageQueue)

Static Private Member Functions

- static void CancellationHandler (void *mutex)

  Cancellation handler.

Private Attributes

- std::list< T > message_queue_

  List of queued messages.
- pthread_mutex_t message_queue_mutex_

  Message queue mutex.
- pthread_cond_t new_message_cond_

  New message condition.

7.44.1 Detailed Description

template<typename T>class ccdvl::MessageQueue< T >

Simple message passing queue.

Template Parameters

T	Message type.

7.44.2 Member Function Documentation

7.44.2.1 template<typename T> static void ccdvl::MessageQueue< T >::CancellationHandler (void * *mutex*) [inline], [static], [private]

Cancellation handler.

Unlocks mutex.

Parameters

in	*mutex*	Mutex to realese on cancel.

7.44.2.2 template<typename T> void ccdvl::MessageQueue< T >::GetAllMessages (std::list< T > & *messages*) [inline]

Get all messages.

Get all messages currently in queue. This operation will not wait for any messages and return immediately, however it must wait for exclusive queue access.

Parameters

messages	[in, out] The list to hold dequeued messages.

7.44.2.3 template<typename T> bool ccdvl::MessageQueue< T >::RemoveMessage (T & *message*) [inline]

Remove a queued message.

Remove a message if it exists.

Parameters

message	The message to remove.

Returns

True iff a message was removed.

7.44.2.4 template<typename T> void ccdvl::MessageQueue< T >::SendMessage (T & *message*) [inline]

Enqueue a message to the message queue.

Parameters

message	The message to enqueue.

7.44.2.5 template<typename T> void ccdvl::MessageQueue< T >::WaitForAllMessages (std::list< T > & *messages*) [inline]

Wait for atleast one message.

Blocks until there is atleast one message queued, then dequeue all message and add them the provided list. This method is also a cancellation point.

Parameters

messages	[in, out] The list to hold dequeued messages.

7.44.2.6 template<typename T> T& ccdvl::MessageQueue< T >::WaitForMessage ( ) [inline]

Wait for a message.

Blocks until a message is queued, then dequeue and return it. This method is also a cancellation point.

Returns

An enqueued message.

The documentation for this class was generated from the following file:

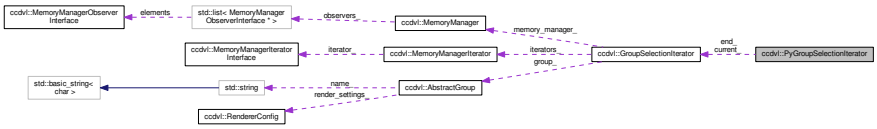
- include/synchronization/message_queue.h

7.45 ccdvl::PyGroupSelectionIterator Class Reference

Iterator used to wrap GroupSelectionIterator and provide Python like iterator methods.

`#include <pyccdvl_group_selection_iterator.h>`

Collaboration diagram for ccdvl::PyGroupSelectionIterator:



Public Member Functions

- PyGroupSelectionIterator (const GroupSelectionIterator &start, const GroupSelectionIterator &end)

  Constructs a new Python like iterator from a starting and stop iterator.
- bool HasNext ()

  Checks if there are elements left.
- std::vector< GraphDouble > Next ()

  Return the current point and increments the iterator.

Private Attributes

- GroupSelectionIterator * current_

  Current iterator.
- GroupSelectionIterator * end_

  Stop iterator.

7.45.1 Detailed Description

Iterator used to wrap GroupSelectionIterator and provide Python like iterator methods.

Combines both start and end iterator to create a safe iterator for sip and Python.

See also

GroupSelectionIterator

7.45.2 Constructor & Destructor Documentation

7.45.2.1 ccdvl::PyGroupSelectionIterator::PyGroupSelectionIterator (const GroupSelectionIterator & *start*, const GroupSelectionIterator & *end*)

Constructs a new Python like iterator from a starting and stop iterator.

Since C++ iterators end method return a iterator pointing past the last element, only the interval [start, end) is considered.

Parameters

in	*start*	Starting iterator.
in	*end*	Stop iterator.

7.45.3 Member Function Documentation

7.45.3.1 bool ccdvl::PyGroupSelectionIterator::HasNext ( )

Checks if there are elements left.

Returns

True iff there is atleast one point left to iterate using the Next method.

7.45.3.2 std::vector< GraphDouble > ccdvl::PyGroupSelectionIterator::Next ( )

Return the current point and increments the iterator.

Returns

The current point.

7.45.4 Member Data Documentation

7.45.4.1 GroupSelectionIterator* ccdvl::PyGroupSelectionIterator::end_ [private]

Stop iterator.

Used to track when the iterator has reached its end.

The documentation for this class was generated from the following files:

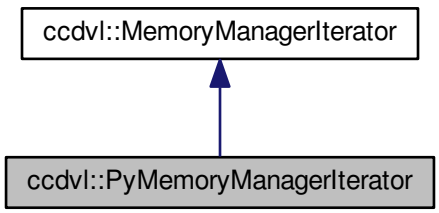
- python/src/pyccdvl_group_selection_iterator.h
- python/src/pyccdvl_group_selection_iterator.cc

7.46 ccdvl::PyMemoryManagerIterator Class Reference

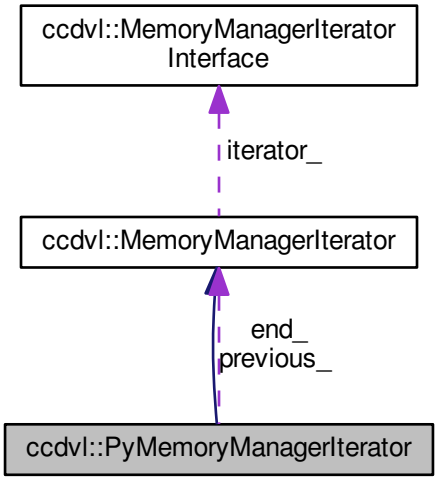
Iterator used to wrap MemoryManagerIterator and provide Python like iterator methods.

`#include <pyccdvl_memory_manager_iterator.h>`

Inheritance diagram for ccdvl::PyMemoryManagerIterator:



Collaboration diagram for ccdvl::PyMemoryManagerIterator:



Public Member Functions

- PyMemoryManagerIterator (const MemoryManagerIterator &start, const MemoryManagerIterator &end)

  Constructs a new Python like iterator from a starting and stop iterator.
- bool HasNext ()

  Checks if there are elements left.
- const AbstractDataSet * Next ()

  Return the current dataset and increments the iterator.

Private Attributes

- MemoryManagerIterator * previous_

  Previous iterator.
- MemoryManagerIterator * end_

  Stop iterator.

7.46.1 Detailed Description

Iterator used to wrap MemoryManagerIterator and provide Python like iterator methods.

Combines both start and end iterator to create a safe iterator for sip and Python.

See also

MemoryManagerIterator

7.46.2 Constructor & Destructor Documentation

7.46.2.1 ccdvl::PyMemoryManagerIterator::PyMemoryManagerIterator (const MemoryManagerIterator & *start,* const MemoryManagerIterator & *end*)

Constructs a new Python like iterator from a starting and stop iterator.

Since C++ iterators end method return a iterator pointing past the last element, only the interval [start, end) is considered.

Parameters

in	*start*	Starting iterator.
in	*end*	Stop iterator.

7.46.3 Member Function Documentation

7.46.3.1 bool ccdvl::PyMemoryManagerIterator::HasNext ( )

Checks if there are elements left.

Returns

True iff there is atleast one dataset left to iterate using the Next method.

7.46.3.2 const AbstractDataSet * ccdvl::PyMemoryManagerIterator::Next ( )

Return the current dataset and increments the iterator.

Returns

The current dataset.

7.46.4 Member Data Documentation

7.46.4.1 MemoryManagerIterator* ccdvl::PyMemoryManagerIterator::end_ [private]

Stop iterator.

Used to track when the iterator has reached its end.

7.46.4.2 MemoryManagerIterator* ccdvl::PyMemoryManagerIterator::previous_ [private]

Previous iterator.

There must exists a copy of the iterator or the data set aquired trough it is not guaranteed to presist. See MemoryManagerIterator.

The documentation for this class was generated from the following files:

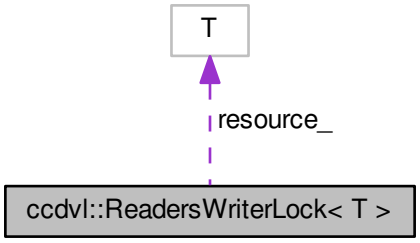
- python/src/pyccdvl_memory_manager_iterator.h
- python/src/pyccdvl_memory_manager_iterator.cc

7.47 ccdvl::ReadersWriterLock< T > Class Template Reference

Synchronization wrapper class which allows multiple readers.

`#include <readers_writer_lock.h>`

Collaboration diagram for ccdvl::ReadersWriterLock< T >:



Public Member Functions

- ReadersWriterLock (T resource)

  Initializes a new resource protected with a readers, writer lock.
- virtual ~ReadersWriterLock ()

  Destroys the object and frees any allocated resources.
- const T & AtomicReadLock ()

  Obtain a read-only lock for wrapped resource.
- void AtomicReadUnlock ()

  Releases read-only access previously obtained by AtomicReadLock().
- T & AtomicWriteLock ()

  Obtain exclusive write for wrapped resource.
- void AtomicWriteUnlock ()

  Releases exclusive access to wrapped resource previously obtained with AtomicWriteLock().

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (ReadersWriterLock)

Static Private Member Functions

- static void CancellationHandler (void *mutex)

  Cancellation handler.

Private Attributes

- T resource_

  Protected resource.
- int32_t reader_count_

  Number of readers waiting for, or accessing resource.
- int32_t writer_count_

  Number of writers waiting for, or accessing resource.
- pthread_mutex_t resource_mutex_

Resource mutex.

- pthread_cond_t readers_waiting_cond_

  Readers waits on this condition.

- pthread_cond_t writers_waiting_cond_

  Writers wait on this condition.

7.47.1 Detailed Description

template<class T>class ccdvl::ReadersWriterLock< T >

Synchronization wrapper class which allows multiple readers.

This class wraps a resource and allows multithreaded read access while only blocking on write. Additionally write has priority over read operations.

Template Parameters

T	The object type to wrap.

7.47.2 Constructor & Destructor Documentation

7.47.2.1 template<class T > ccdvl::ReadersWriterLock< T >::ReadersWriterLock (T *resource*) [inline], [explicit]

Initializes a new resource protected with a readers, writer lock.

Parameters

resource	The resource to wrap.

7.47.3 Member Function Documentation

7.47.3.1 template<class T > const T& ccdvl::ReadersWriterLock< T >::AtomicReadLock ( ) [inline]

Obtain a read-only lock for wrapped resource.

This guarantee that no writer uses the wrapped resource. Blocks until all writers are done. This method is also a cancellation point.

Returns

The wrapped resource.

7.47.3.2 template<class T > T& ccdvl::ReadersWriterLock< T >::AtomicWriteLock ( ) [inline]

Obtain exclusive write for wrapped resource.

This guarantee that no other writer and no readers uses the wrapped resource. Blocks until exclusive access is obtained. This method is also a cancellation point.

Returns

The wrapped resource.

7.47.3.3 template<class T > static void ccdvl::ReadersWriterLock< T >::CancellationHandler (void * *mutex*) [inline], [static], [private]

Cancellation handler.

Unlocks mutex.

Parameters

in	*mutex*	Mutex to realese on cancel.

The documentation for this class was generated from the following file:

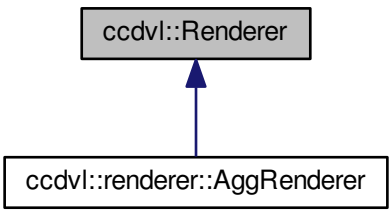
- include/synchronization/readers_writer_lock.h

7.48 ccdvl::Renderer Class Reference

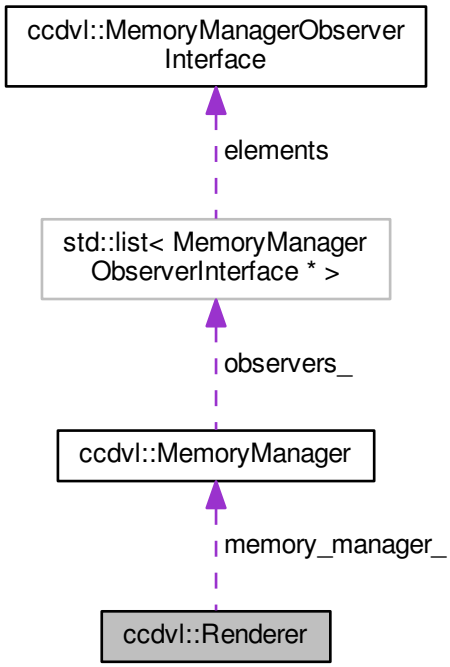
Abstract class for renderers used to render a graph to image slices.

`#include <renderer.h>`

Inheritance diagram for ccdvl::Renderer:



Collaboration diagram for ccdvl::Renderer:



Public Types

- typedef std::vector< const AbstractDataSet * > DataPoints

  Type representing a set of data points.

Public Member Functions

- Renderer (int8_t dimensions, MemoryManager *memory_manager)

  Initialize constant for subclasses.
- virtual ~Renderer ()

  Destroys the object and frees any allocated resources.
- virtual void ClearDraw (const GraphTileState &state, GraphTile *render_target)=0

  Clear all graph tiles associated with a given state.
- virtual void DrawSet (const GraphTileState &state, const DataPoints &data, GraphTile *render_target)=0

  Draw provided data onto graph tiles associated with a given state.
- virtual void DrawAll (std::list< std::pair< const GraphTileState *, GraphTile * > > *tiles)

  Draw a list of tiles as a batch operation.
- virtual void Abort ()

  Abort current draw operation.

Public Attributes

- const int8_t dimensions_

  Number of dimensions supported.

Protected Attributes

- MemoryManager * memory_manager_

  Memory manager holding the data to render.
- bool abort_

  Quick termination control.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (Renderer)

7.48.1 Detailed Description

Abstract class for renderers used to render a graph to image slices.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 ccdvl::Renderer::Renderer (int8_t *dimensions*, MemoryManager * *memory_manager*)

Initialize constant for subclasses.

Parameters

	dimensions	The number of dimensions. Must be positive and larger then zero.
in	*memory_manager*	Memory manager to read data from.

7.48.3 Member Function Documentation

7.48.3.1 void ccdvl::Renderer::Abort ( ) [virtual]

Abort current draw operation.

When called, drawing threads will be interrupted and return early. ClearDraw() excluded.

See also

abort_

Reimplemented in ccdvl::renderer::AggRenderer.

7.48.3.2 virtual void ccdvl::Renderer::ClearDraw (const GraphTileState & *state*, GraphTile * *render_target*) [pure virtual]

Clear all graph tiles associated with a given state.

Parameters

in	*state*	Graph state for tile.
in,out	*render_target*	Graph tile to clear.

Implemented in ccdvl::renderer::AggRenderer.

7.48.3.3 void ccdvl::Renderer::DrawAll (std::list< std::pair< const GraphTileState *, GraphTile * > > * *tiles*) [virtual]

Draw a list of tiles as a batch operation.

When completed the tiles will be marked as such.

Parameters

in,out	*tiles*	The list of graph tiles and their states to render.

See also

abort_, GraphTile::completed

7.48.3.4 virtual void ccdvl::Renderer::DrawSet (const GraphTileState & *state*, const DataPoints & *data*, GraphTile * *render_target*) [pure virtual]

Draw provided data onto graph tiles associated with a given state.

Parameters

in	*state*	Graph state for tile.
in	*data*	Data to draw.
in,out	*render_target*	Graph tile to update with provided data.

Implemented in ccdvl::renderer::AggRenderer.

7.48.4 Member Data Documentation

7.48.4.1 bool ccdvl::Renderer::abort_ [protected]

Quick termination control.

When set to true, rendering will be aborted causing an early return of DrawAll().

See also

Abort(), DrawAll()

The documentation for this class was generated from the following files:

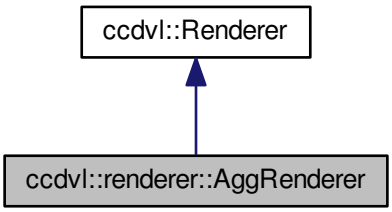
- include/renderer.h
- src/renderer.cc

7.49 ccdvl::renderer::AggRenderer Class Reference

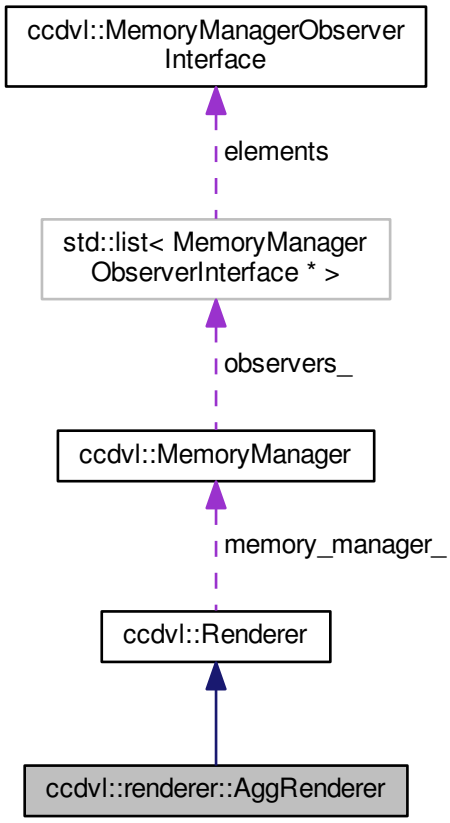
A simple 2D software renderer using Anti-grain Geometry (AGG).

`#include <agg_renderer.h>`

Inheritance diagram for ccdvl::renderer::AggRenderer:



Collaboration diagram for ccdvl::renderer::AggRenderer:



Public Types

- typedef MemoryManager::iterator MemoryManagerIterator

  Memory manager iterator type.

Public Member Functions

- AggRenderer (MemoryManager *memory_manager)

  Constructs an AGG renderer.
- ~AggRenderer ()

  Destroys the object and frees any allocated resources.
- virtual void ClearDraw (const GraphTileState &state, GraphTile *render_target)

  Clear all graph tiles associated with a given state.
- virtual void DrawSet (const GraphTileState &state, const DataPoints &data, GraphTile *render_target)

  Draw provided data onto graph tiles associated with a given state.
- virtual void Abort ()

  Abort current draw operation.

Private Attributes

- bool abort_draw_

  Quick termination control.

Related Functions

(Note that these are not member functions.)

- template<class T >
  void AggRenderPoint (const GraphTileState &state, const RendererConfig &settings, const std::vector< double > &point, const std::vector< double > *next_point, agg::renderer_primitives< agg::renderer_base< T > > *primitive_renderer)

  Render a point from a two dimensional data set.

Additional Inherited Members

7.49.1 Detailed Description

A simple 2D software renderer using Anti-grain Geometry (AGG).

7.49.2 Constructor & Destructor Documentation

7.49.2.1 ccdvl::renderer::AggRenderer::AggRenderer (MemoryManager * *memory_manager*) [explicit]

Constructs an AGG renderer.

Parameters

in	*memory_manager*	Memory manager to read data from.

7.49.3 Member Function Documentation

7.49.3.1 void ccdvl::renderer::AggRenderer::Abort ( ) [virtual]

Abort current draw operation.

When called, drawing threads will be interrupted and return early. ClearDraw() excluded.

See also

abort_

Reimplemented from ccdvl::Renderer.

7.49.3.2 void ccdvl::renderer::AggRenderer::ClearDraw (const GraphTileState & *state*, GraphTile * *render_target*) [virtual]

Clear all graph tiles associated with a given state.

Parameters

in	*state*	Graph state for tile.
in,out	*render_target*	Graph tile to clear.

Implements ccdvl::Renderer.

7.49.3.3 void ccdvl::renderer::AggRenderer::DrawSet (const GraphTileState & *state*, const DataPoints & *data*, GraphTile * *render_target*) [virtual]

Draw provided data onto graph tiles associated with a given state.

Parameters

in	*state*	Graph state for tile.
in	*data*	Data to draw.
in,out	*render_target*	Graph tile to update with provided data.

Implements ccdvl::Renderer.

7.49.4 Friends And Related Function Documentation

7.49.4.1 template<class T > void AggRenderPoint (const GraphTileState & *state*, const RendererConfig & *settings*, const std::vector< double > & *point*, const std::vector< double > * *next_point*, agg::renderer_primitives< agg::renderer_base< T > > * *primitive_renderer*) [related]

Render a point from a two dimensional data set.

Template Parameters

T	AGG pixel format class which must represent the used pixel format.

Parameters

	state	The graph tile to render.
	settings	Rendering settings for graph tile.
	point	The data point to render.
in	*next_point*	The next data point to render, NULL means that this it the last point.
in,out	*primitive_renderer*	The AGG initialized buffer and renderer.

7.49.5 Member Data Documentation

7.49.5.1 bool ccdvl::renderer::AggRenderer::abort_draw_ [private]

Quick termination control.

When set to true, rendering will be aborted causing an early return of DrawSet().

The documentation for this class was generated from the following files:

- include/agg_backend/agg_renderer.h
- src/agg_backend/agg_renderer.cc

7.50 ccdvl::RendererConfig Class Reference

A class that holds group specific rendering configuration.

`#include <renderer_config.h>`

Public Types

- enum PointShape { kInvalid = -1, kNone, kCricle, kSquare }

  Enumeration of possible datapoint representations in a graph.

Public Member Functions

- RendererConfig ()

  Create a new instance; with a default configuration.
- RendererConfig (const RendererConfig &instance)

  Copy constructor.
- virtual ~RendererConfig ()

  Destroys the object and frees any allocated resources.
- bool LessThan (const RendererConfig &rhs) const

  Less than comparison from renderer configurations.

Public Attributes

- PointShape point_shape_

  Shape to draw for each point.
- int16_t point_size_

  Size of datapoint shape.
- int16_t line_width_

  Width of lines between points.
- double deviation_

  Numeric deviation used to render error bars.
- uint8_t point_color_ [4]

  Color of data point shapes.
- uint8_t line_color_ [4]

  Color of lines between points.
- uint8_t deviation_color_ [4]

  Color of error bars.

7.50.1 Detailed Description

A class that holds group specific rendering configuration.

7.50.2 Member Enumeration Documentation

7.50.2.1 enum ccdvl::RendererConfig::PointShape

Enumeration of possible datapoint representations in a graph.

Enumerator:

kInvalid Invalid shape.

kNone Nothing.

kCricle A cricle.

kSquare A square.

7.50.3 Member Function Documentation

7.50.3.1 bool ccdvl::RendererConfig::LessThan (const RendererConfig & *rhs*) const

Less than comparison from renderer configurations.

Parameters

in	*rhs*	Right hand side configuration to compare with.

Returns

true Iff this configuration is less than the provided one.

7.50.4 Member Data Documentation

7.50.4.1 double ccdvl::RendererConfig::deviation_

Numeric deviation used to render error bars.

Values equal to or less then zero are invalid, this will cause any implemented renderer to skip rendering error bars.

7.50.4.2 uint8_t ccdvl::RendererConfig::deviation_color_[4]

Color of error bars.

Colors must be stored as RGBA.

Note

Deviation from code standard, unsigned. Qt's QColor accepts signed values while AGG expects unsigned 8bit integers.
Keeping color as unsigned 8bit integers avoids using regular integers and extra typecasting.

7.50.4.3 uint8_t ccdvl::RendererConfig::line_color_[4]

Color of lines between points.

Colors must be stored as RGBA.

Note

Deviation from code standard, unsigned. Qt's QColor accepts signed values while AGG expects unsigned 8bit integers.
Keeping color as unsigned 8bit integers avoids using regular integers and extra typecasting.

7.50.4.4 int16_t ccdvl::RendererConfig::line_width_

Width of lines between points.

Values equal to or less then zero are invalid, this will cause any implemented renderer to skip rendering of lines.

7.50.4.5 uint8_t ccdvl::RendererConfig::point_color_[4]

Color of data point shapes.

Colors must be stored as RGBA.

Note

Deviation from code standard, unsigned. Qt's QColor accepts signed values while AGG expects unsigned 8bit integers.
Keeping color as unsigned 8bit integers avoids using regular integers and extra typecasting.

7.50.4.6 PointShape ccdvl::RendererConfig::point_shape_

Shape to draw for each point.

Invalid values should be treated by any implemented renderer as kNone.

7.50.4.7 int16_t ccdvl::RendererConfig::point_size_

Size of datapoint shape.

Values equal to or less then zero are invalid, this will cause any implemented renderer to skip rendering of points.

The documentation for this class was generated from the following files:

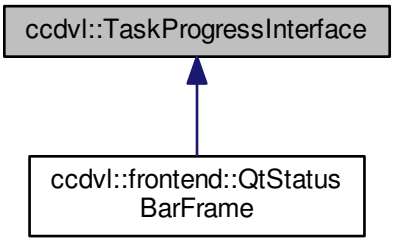
- include/renderer_config.h
- src/renderer_config.cc

7.51 ccdvl::TaskProgressInterface Class Reference

A simple interface that provides a function for sending information to a progress bar or similar type of progress display.

`#include <task_progress_interface.h>`

Inheritance diagram for ccdvl::TaskProgressInterface:



Public Member Functions

- virtual ~TaskProgressInterface ()

  Interfaces have destructors with empty bodies.
- virtual void SetTaskProgress (const char *text, int value, int max)=0

  Updates the progress information for a task.

7.51.1 Detailed Description

A simple interface that provides a function for sending information to a progress bar or similar type of progress display.

7.51.2 Member Function Documentation

7.51.2.1 virtual void ccdvl::TaskProgressInterface::SetTaskProgress (const char * *text*, int *value*, int *max*) [pure virtual]

Updates the progress information for a task.

Parameters

in	*text*	The name or description of the task.
in	*value*	The current value of the progress.
in	*max*	The maximum value of the progress.

Implemented in ccdvl::frontend::QtStatusBarFrame.

The documentation for this class was generated from the following file:

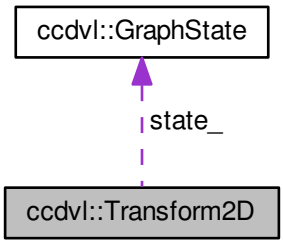
- include/task_progress_interface.h

7.52 ccdvl::Transform2D Class Reference

Simple point transformation class for two dimensional data.

`#include <transform_2d.h>`

Collaboration diagram for ccdvl::Transform2D:



Public Types

- enum Outcome { kTransformSuccess, kTransformFaild }

  Enumerations of return codes.

Public Member Functions

- Transform2D (const GraphTileState &state)

  Initializes a new transformation.
- Transform2D (const GraphSceneState &state)

  Initializes a new transformation.
- Outcome ToGraphSpace (std::vector< double > *vector)

  Transform a scene coordinate to graph space.
- Outcome FromGraphSpace (std::vector< double > *vector)

  Transform a graph coordinate to scene space.
- Outcome CalculateBottomLeft (std::vector< double > *vector)

  Calculates the new bottom left position for the given center point in the graph state.
- bool IsDoubleInfinity (double n)

  Test if a floating point value is infinite.
- bool IsDoubleNaN (double n)

  Test if a floating point value is a not-a-number.

Private Types

- enum FloatingpointClassification { kFiniteValue, kInfinite, kMinusInfinite, kNotANumber }

  Floating-point classification.

Private Member Functions

- FloatingpointClassification ClassifyNumber (double number) const

  Classify a given number.
- void Precompute ()

  Precompute parts of the transformation to reduce the number of required floating-point operations.

Private Attributes

- bool scene_transform_

  Transform is a scene specific transform.
- const GraphState state_

  The state to transform in.
- double zoom_scale_transform_ [2]

  Precomputed transformation constants.

7.52.1 Detailed Description

Simple point transformation class for two dimensional data.

Assumes that the Y axis is flipped for scene, but not for tiles.

Note

While Outcome indicate success or failure, currently overflow is not handled correctly. It does catch (-)infinite and NaN values but the FPU can also overflow without returning any of those.

7.52.2 Member Enumeration Documentation

7.52.2.1 enum ccdvl::Transform2D::FloatingpointClassification [private]

Floating-point classification.

Enumerator:

kFiniteValue Value is finite and valid.

kInfinite Value is infinity.

kMinusInfinite Value is negative infinity.

kNotANumber Value is invalid.

7.52.2.2 enum ccdvl::Transform2D::Outcome

Enumerations of return codes.

Enumerator:

kTransformSuccess Transformation successful.

kTransformFaild Transformation failed and the result is now invalid.

7.52.3 Constructor & Destructor Documentation

7.52.3.1 ccdvl::Transform2D::Transform2D (const GraphTileState & *state*) [explicit]

Initializes a new transformation.

Parameters

state	The state to transform in.

7.52.3.2 ccdvl::Transform2D::Transform2D (const GraphSceneState & *state*) [explicit]

Initializes a new transformation.

Parameters

state	The state to transform in.

7.52.4 Member Function Documentation

7.52.4.1 Transform2D::Outcome ccdvl::Transform2D::CalculateBottomLeft (std::vector< double > * *vector*)

Calculates the new bottom left position for the given center point in the graph state.

The internal graph state is updated with the new bottom left coordinate for the provided center coordinate. Beware that failure invalidates the internal graph state for this transformation instance.

In order to find the new bottom left base coordinate from a provided center point and the current view settings require a small trick, first the center point is considered as the base coordinate; then the real base coordinated is computed by converting the relative scene coordinate (-width/2, height∗1.5), which corresponds to the actual bottom left base coordinate.

Parameters

in,out	*vector*	A vector containing the center point coordinates for the graph state.

Returns

The outcome of the computation; the result is stored directly to the given *vector*.

7.52.4.2 Transform2D::FloatingpointClassification ccdvl::Transform2D::ClassifyNumber (double *number*) const [private]

Classify a given number.

See http://www.johndcook.com/IEEE_exceptions_in_cpp.html and FloatingpointClassification. However a few tests show that a double can overflow like an integer if the operation adds too little to double max. This will require activation of FPU signals.

Parameters

number	The number to classify.

Returns

Classification.

7.52.4.3 Transform2D::Outcome ccdvl::Transform2D::FromGraphSpace (std::vector< double > * *vector*)

Transform a graph coordinate to scene space.

Parameters

in,out	*vector*	The coordinate to transform.

Returns

The outcome of the computation; the result is stored directly to the given *vector*.

7.52.4.4 bool ccdvl::Transform2D::IsDoubleInfinity (double *n*)

Test if a floating point value is infinite.

Note

Avoid use, if possible.

7.52.4.5 bool ccdvl::Transform2D::IsDoubleNaN (double *n*)

Test if a floating point value is a not-a-number.

Note

Avoid use, if possible.

7.52.4.6 Transform2D::Outcome ccdvl::Transform2D::ToGraphSpace (std::vector< double > * *vector*)

Transform a scene coordinate to graph space.

Parameters

in,out	*vector*	The coordinate to transform.

Returns

The outcome of the computation; the result is stored directly to the given *vector*.

The documentation for this class was generated from the following files:

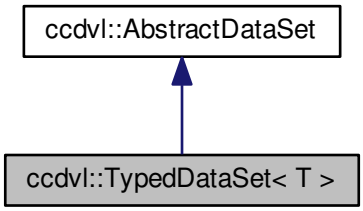
- include/transform_2d.h
- src/transform_2d.cc

7.53 ccdvl::TypedDataSet< T > Class Template Reference

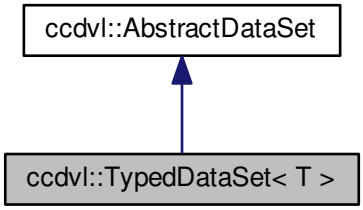
AbstractDataSet container.

`#include <typed_data_set.h>`

Inheritance diagram for ccdvl::TypedDataSet< T >:



Collaboration diagram for ccdvl::TypedDataSet< T >:



Public Member Functions

- TypedDataSet (size_t size)

  Constructs a self allocated data instance.
- TypedDataSet (T *base, size_t size)

  Constructs a pre-allocated data instance.
- TypedDataSet (T **base, ptrdiff_t offset, size_t size)

  Constructs a relocatable, pre-allocated data instance.
- virtual void Accept (DataSetVisitorInterface *v)

  Visitor accept method.
- virtual double GetValue (size_t index) const

  Fetches a value as a double floating point.
- virtual size_t GetTypeSize () const

  Get the type size in bytes.
- T * GetBuffer () const

  Fetches current data buffer.

Private Member Functions

- **DISSALLOW_COPY_AND_ASSIGN** (TypedDataSet)

Private Attributes

- T * allocation_base_

  Memory pointer to data allocated by this class.
- T ** base_

  Pointer to shared allocation pointer.
- ptrdiff_t offset_

  Byte offset to data in allocation.

Additional Inherited Members

7.53.1 Detailed Description

template<typename T>class ccdvl::TypedDataSet< T >

AbstractDataSet container.

Template Parameters

T	Numeric data type to hold in a dataset.

7.53.2 Constructor & Destructor Documentation

7.53.2.1 template<typename T> ccdvl::TypedDataSet< T >::TypedDataSet (size_t *size*) [inline], [explicit]

Constructs a self allocated data instance.

Parameters

size	Number of data elements.

7.53.2.2 template<typename T> ccdvl::TypedDataSet< T >::TypedDataSet (T * *base*, size_t *size*) [inline]

Constructs a pre-allocated data instance.

This constructor is intended to be used internally by MemoryManagerInterface classes. It will take ownership of the allocated memory.

Parameters

in	*base*	Data base pointer to size number of elements.
	size	Number of data elements.

7.53.2.3 template<typename T> ccdvl::TypedDataSet< T >::TypedDataSet (T ** *base*, ptrdiff_t *offset*, size_t *size*) [inline]

Constructs a relocatable, pre-allocated data instance.

This constructor is intended to be used internally by MemoryManager classes.

Parameters

in	*base*	Relocation base pointer.
in	*offset*	Byte offset from relocation base to data.
	size	Number of data elements.

7.53.3 Member Function Documentation

7.53.3.1 template<typename T> virtual void ccdvl::TypedDataSet< T >::Accept (DataSetVisitorInterface * *v*) [inline], [virtual]

Visitor accept method.

Parameters

in	*v*	Visitor to use.

Implements ccdvl::AbstractDataSet.

7.53.3.2 template<typename T> T* ccdvl::TypedDataSet< T >::GetBuffer ( ) const [inline]

Fetches current data buffer.

Returns

Content data buffer.

Note

This buffer is not always valid, as it could be invalidated iff the instance was constructed using the relocatable constructor.

7.53.3.3 template<typename T> virtual size_t ccdvl::TypedDataSet< T >::GetTypeSize ( ) const [inline], [virtual]

Get the type size in bytes.

Returns

Number of bytes.

Implements ccdvl::AbstractDataSet.

7.53.3.4 template<typename T> virtual double ccdvl::TypedDataSet< T >::GetValue (size_t *index*) const [inline], [virtual]

Fetches a value as a double floating point.

Parameters

index	Index of the value to get.

Returns

The value at the provided index as a double floating point.

Implements ccdvl::AbstractDataSet.

The documentation for this class was generated from the following file:

- include/data_set/typed_data_set.h

Index