

**Research Paper**

**Units Library Research**

**Inspiro BV**

|  |  |
| --- | --- |
| **Customer** | Inspiro BV  Mr B M Teldersstraat 4 6842 CT Arnhem |
| **Project name** | C++ Units Library |
| **Project number** | 0569 |
| **Document type** | Research Paper |
| **Document number** | INSP0420 |
| **Document version** | [2.0] |
| **Release date** | 03/04/2025 |
| **Author(s)** | Lars van Duijnhoven, Inspiro BV |
| **Attachment(s)** | Research Table |

TABLE OF CONTENTS

[1 Introduction 5](#_Toc194493914)

[1.1 General information 5](#_Toc194493915)

[1.1.1 Background information 5](#_Toc194493916)

[1.1.2 Research goal 5](#_Toc194493917)

[1.2 Reference documents 5](#_Toc194493918)

[1.3 Definitions 5](#_Toc194493919)

[1.4 Abbreviations 6](#_Toc194493920)

[2 Research methodology 6](#_Toc194493921)

[2.1 Problem Analysis 6](#_Toc194493922)

[2.2 Literature Study 7](#_Toc194493923)

[2.3 Available Product Analysis 7](#_Toc194493924)

[2.4 Data Analytics 7](#_Toc194493925)

[3 Criteria and rationale 8](#_Toc194493926)

[3.1 Plan for research 8](#_Toc194493927)

[3.2 Criteria 9](#_Toc194493928)

[3.2.1 Knockout criteria 9](#_Toc194493929)

[3.2.2 In-depth criteria 9](#_Toc194493930)

[3.3 Background information about unit library problems 12](#_Toc194493931)

[3.3.1 Difference between binary and decimal memory types 12](#_Toc194493932)

[3.3.2 Affine- and vector spaces 13](#_Toc194493933)

[3.3.3 Decibel and other logarithmic units 14](#_Toc194493934)

[3.3.4 Differentiating between Hz, Bq and Bd 15](#_Toc194493935)

[4 Results 16](#_Toc194493936)

[4.1 Highest Common Denominator C++ version 16](#_Toc194493937)

[4.2 Knockout table/ long list 16](#_Toc194493938)

[4.3 Medium list 17](#_Toc194493939)

[4.4 Analysis used units within Inspiro 17](#_Toc194493940)

[4.4.1 Previously experienced problems 18](#_Toc194493941)

[4.4.2 Would a library have helped? 19](#_Toc194493942)

[4.4.3 Concerns with future use of a library 19](#_Toc194493943)

[4.4.4 Other idea’s or comments 21](#_Toc194493944)

[4.4.5 Used units 21](#_Toc194493945)

[4.5 Overflow protection 22](#_Toc194493946)

[5 Conclusion 23](#_Toc194493947)

[6 Sources 24](#_Toc194493948)

[Appendix 1: Research table spreadsheet 25](#_Toc194493949)

list of tables

[Table 1: Document release history 3](#_Toc194568740)

[Table 2: Open issues 4](#_Toc194568741)

[Table 3: Reference documents 5](#_Toc194568742)

[Table 4: Definitions 6](#_Toc194568743)

[Table 5: Abbreviations 6](#_Toc194568744)

[Table 6: Medium list criteria 12](#_Toc194568745)

[Table 7: Shortlist criteria 12](#_Toc194568746)

[Table 8: Binary and decimal memory comparison 13](#_Toc194568747)

[Table 9: Highest common denominator table without sources 16](#_Toc194568748)

[Table 10: Top 5 libraries from the medium list 17](#_Toc194568749)

list of Figures

[Figure 1: Automatic conversions example 10](#_Toc194568793)

[Figure 2: Explanation of affine- and vector spaces 14](#_Toc194568794)

[Figure 3: Screenshot of the long list including all libraries that continued to the medium list 17](#_Toc194568795)

[Figure 4: Screenshot used units results 21](#_Toc194568796)

DOCUMENT RELEASE HISTORY

Since this is a different way of releasing things, a different way of version numbering has to be thought of.  
Instead of Semantic Versioning, it has been decided to create a custom version numbering for this specific research. This shall be explained below:

Version 1.0 still has major 1 and minor 0, but the patch identifier does not exist anymore.  
Every time the progress will be discussed with the client, a new major version needs to be added.  
A PDF also needs to be made of this new major version, which must be added in the releases folder.  
The same goes for the research table, this has version numbering in the same way, only a different folder.

Any time new chapters are added and finished, the minor version needs to be updated.  
Multiple functionalities can be added in one minor version, but all functionalities have to be named in the description.

|  |  |  |
| --- | --- | --- |
| Version | Release date | Description |
| 1.0 | 27-03-2025 | First official release with chapters 0, 2 and 3 added and finished. |
| 1.1 | 01-04-2025 | Added highest common denominator C++ version, longlist, medium list and analysis results and conclusions to chapter 4 Results. This does need to still be reviewed. |
| 2.0 | 02-04-2025 | Results from chapter 4 Results have been reviewed. The feedback given in these reviews has been processed and 3 appendixes have been added, these being Appendix 1:, Appendix 2: and Appendix 3:. Furthermore, chapter 3 has been partially reviewed as well. |
|  |  |  |

Table 1: Document release history

OPEN ISSUES

The following issues are still open and need to be resolved:

|  |  |  |
| --- | --- | --- |
| No. | Date | Description |
| 1 | 27-03-2025 | Scaling overflow protection in results needs to be better documented. As of right now it is just a reminder with a few links. |
| 2 | 27-03-2025 | A conclusion must be drawn from the gathered results. |
| 3 | 27-03-2025 | An advice must be given after creating the conclusion. |
| 4 | 27-03-2025 | Sources needs to be expanded with a few new sources |
| 5 | 31-03-2025 | The project description has to be added in English, since it is currently available in Dutch in a different document. |
| 6 | 31-03-2025 | A reflection on the way the analysis was executed has to be added, perhaps in a different  “reflection” chapter before or after the conclusion. |
| 7 | 01-04-2025 | Al background information from chapter 3.3 has to be reviewed. |
| 8 | 02-04-2025 | The additional licenses in chapter 4.2 have to be updated. As of 02-04-2025 we are waiting on answers from a colleague. This is about the LLC license, BSD3-Clause license and the CPOL license |
| 9 | 03-04-2025 | Built-in units criterion in chapter 3.2.2 have to be modified to contain the new list of units that got created after the analysis. |

Table 2: Open issues

# Introduction

This chapter focuses on the background information, the project goal and explaining the terms and definitions found in this document.

## General information

### Background information

In the current Inspiro projects their programmers create their own types, such as “length”.  
This in itself is not the problem.

The problem arises when this is required for more projects, so more time has to be spent on this. Instead of continuously spending time on this, Inspiro BV has decided to invest more time into this to get a library that they can use for all future projects. This will also improve type-safety so accidents like [these](https://en.wikipedia.org/wiki/Mars_Climate_Orbiter#Cause_of_failure) do not happen.  
This is where the project itself started.

The research paper that you are currently reading helps create a first step in this project by defining where the starting point will be (using an existing library or creating one).

### Research goal

The main goal of this research paper is to give substantiated advice to Inspiro BV about what open-source library to use in future projects.  
This advice can also be to not implement a library from someone else, but to make their own.

Another goal of this research paper is to get a comparison of which libraries are available and what they have to offer.  
If Inspiro BV decides to not adopt the advice given in this paper, they might want to compare the libraries themselves.  
This comparison provides the tools necessary to come to your own conclusion.

## Reference documents

The following documents are used as input for this document:

|  |  |  |  |
| --- | --- | --- | --- |
| Ref | Document number | Version | Description |
| [REF1](https://inspirobv.sharepoint.com/:w:/r/sites/QualityManagementSystemQMS/Gedeelde%20documenten/General/Inspiro%20-%20Templates/INSP000-EN%20Generic%20template%20v2.1.dotx?d=w7e246060964740faa9c82329506f2ea2&csf=1&web=1&e=ARUp48) | Generic Template EN | 2.1 | The English version of the generic template on which this document is based. |
| [REF2](https://inspirobv.sharepoint.com/:x:/r/sites/Inspiro-StageenAfstuderen-LarsvanDuijnhoven/Gedeelde%20documenten/Lars%20van%20Duijnhoven%20-%20Cpp%20Units%20Library/Oplevering/Releases%20-%20Research%20Table/V1.0_Research_Table.xlsx?d=wcbc8259e0a9640529641ca2ed518c478&csf=1&web=1&e=5N7MCP&nav=MTVfezUyOTNGQzkxLTAyRkUtNDE4NS1CMTVFLTZBQUI5OTFFQjIwM30) | Research\_Table.xlsx | 1.0 | The research table (excel sheet) for comparing the libraries. |
| [refc] |  | … | …….. |

Table 3: Reference documents

## Definitions

The following definitions are used in this document:

|  |  |
| --- | --- |
| Term | Description |
| Typesafety | Typesafety ensures that values are only used in ways consistent with their defined types. For example, you cannot make an integer value a string, because typesafety blocks it from being set. |
| Compiletime | Compiletime refers to the moment when a program is being compiled.  Important to note is that this happens **before** running the program. Preferably, checks are shown here, because they are shown earlier. |
| Runtime | Runtime refers to the moment when a program is running. This can only happen after after compiling. |
| Dimension | A way to measure physical values. Examples are distance, temperature and voltage. |
| Unit | Most of the time a physics unit, such as meter, Celsius or Volt. A way to express a dimension. |
| Dimension checking | The checking if a unit belongs to the correct dimension, since a distance in Volts cannot work. |
| Highest common denominator | The highest shared supported value. For example with C++ version you can have 11, 14 and 17. 11 is the highest common denominator then, since 14 and 17 also support 11, but 11 does not support 14 nor 17. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Table 4: Definitions

## Abbreviations

The following abbreviations are used in this document:

|  |  |
| --- | --- |
| Abbreviation | Description |
| SI | International System of Units (Système International d'Unités) |
| GCC | GNU Compiler Collection, C++ build system |
| CPPCheck | Name of a C/C++ static analysis tool |
| BSL-1.0 | Boost Software License 1.0 |
| WTFPL | What The Fuck Public License |
|  |  |

Table 5: Abbreviations

# Research methodology

This chapter covers the research methods which are used in this research and, more specifically, what they are used for.  
All methods found below were taken from the [ICT Research Methods](https://www.ictresearchmethods.nl/) site.

## [Problem Analysis](https://www.ictresearchmethods.nl/field/problem-analysis/)

In order to get a suitable library out of the research for Inspiro, the core problem must be understood.  
What current problem does Inspiro have, why is this a problem, who has this problem, what is the origin of the problem and why are libraries seen as a possible solution for this problem?

This method and these questions were used to get the information for chapter 1.1 and 1.2 from this document.

## [Literature Study](https://www.ictresearchmethods.nl/library/literature-study/)

This method is mostly used to get an understanding of the field that the research is being held in, for which I am going to be using it too.

The libraries must be compared on some criteria, but what are these criteria going to be and how can they be compared? That is the question that gets answered by this method. For example, if it is unclear how to check if a library is made for compile time- or run time checks, then the difference will first have to be researched. It is only after understanding the difference that this criterion can be checked in the research.

In short, this method has been used to get a better understanding of what a C++ units library is, what the differences between libraries are and how to grade these differences in a points grading system.

## [Available Product Analysis](https://www.ictresearchmethods.nl/library/available-product-analysis/)

This method is for researching if your problem has not already been solved by someone else.

In this research it has been used to research which libraries are currently available on the market and if there is one that can be the solution to Inspiro’s problem. Since the chances of finding the perfect solution are astronomically low, all found libraries are added to a table so they can later be compared.  
The thought behind this is, if the perfect one cannot be found, find the one closest to being perfect and change it to your needs.

## [Data Analytics](https://www.ictresearchmethods.nl/lab/data-analytics/)

This method describes the creation of data to then analyse this data and get conclusions out of this.  
For example, advertising companies analyse data to show more relevant advertisements.

In this research data analytics will be used to create data (in the form of points for the grading system) based on the libraries gathered with the available product analysis.  
This data/ These points are then used to draw a conclusion about which libraries are most suitable as solution for Inspiro.

# Criteria and rationale

The criteria and rationale for the research will be explained in this chapter.  
Furthermore it will also include the plan for how this research is going to take these criteria into account.

## Plan for research

In order to get a structured research paper I have split my work into 5 steps.   
These steps can be found below.

First some extra clarification:

“Rating” a criteria means the library will get a score from 0 to 10 depending on how good it handles the criteria, with 10 being the highest score.  
These scores are added together in the end1.

Now onto the steps:

1. **Create a list:** Make an extensive list of open-source libraries that are currently available online. The minimum is 20. All libraries shall have a name and link, so readability and reproducibility are improved and the libraries can be identified easily.
2. **Find C++ highest common denominator:** In order to find out which C++ versions will result in a knockout in the next step, the highest common denominator must be found. Here the highest C++ version shall be found that is supported by the STM32, ESP32 and the Nordic nRF. It is only after finding this that we can move on to the next step.
3. **Knockout round (longlist):**  Check the libraries from step 1 for the most important criteria, such as C++ version and software license. These criteria are all knockout criteria, since we cannot use it if it was made for a newer C++ version. The remaining libraries go through to the next step.
4. **In-depth round (mediumlist):** Rate the libraries on the remaining criteria, such as unit test coverage, total stack usage and external dependencies. These ratings are eventually added together to get one general rating. All libraries that are rated here go through to the next step.
5. **Summary (shortlist) & conclusion:** Create a top 5 of highest scoring libraries, after which the differences shall be researched and documented in a conclusion. Here the advice shall also be created and added for the best current C++ unit library for Inspiro.

1 Not all criteria have the same weight, but this cannot be processed by the current way of researching (using the score system). This can lead to the top 5 from step 5 consisting of the absolute highest scoring libraries, but not the top 5 highest scoring libraries with Inspiro’s preferences in mind.  
In short, the statistically best library can differ from the actual best library.

In this case these values are only helpful for getting an idea of the better and worse libraries, so in the broader picture. Once the top 5 have been selected, they can also be used to get a more in-depth view of where the top 5 differ from each other.

A different solution is to add a column where preferences can be added. This column is to add weight to certain criteria using Excel math. For example, when you double the points in the column for expandability, that criteria will have twice as much weight to it. This means expandable libraries are preferred above less expandable libraries.

In this research paper I will only use the weight solution, since these weights can be adapted to Inspiro’s preferences.  
The weights that were used for this research paper can be found below chapter 3.2.2.

## Criteria

### Knockout criteria

* *Software license*The following licenses are usable for Inspiro BV: BSL-1.0, MIT, WTFPL, Apache 2.0 as of 05-03-2025  
  Any other license will result in a knockout.
* *C++ version*  
  The C++ version must be supported within Inspiro, if not yet supported it should at least be possible. The result of the highest common denominator research will determine which versions can’t be used and are a knockout.
* *When does dimensional analysis occur?*  
  This must occur compile time, if this happens runtime it will result in a knockout.
* *Heap and stack usage (will only be checked in the shortlist/ top 5 libraries)*  
  This criteria is to see how efficient the libraries provide the functionalities and for checking if no dynamic memory allocation is used. If it is found that the library uses heap, then this will result in a knockout. If it does not use heap, then the stack usage of one unit will be tested. This must be the same unit among all libraries and for this research the unit “length” was used.

### In-depth criteria

* *Expandability*  
  This criteria measures how difficult it is to expand the library yourself with new units.  
  Macro’s should be avoided, since normal C++ code (templates and aliases) is more reliable and better in line with Inspiro’s code guidelines. It has been explicitly named here, because it plays too big of a role here to be put under the “C++ coding guidelines” criteria.
* *Configurability*This criteria measures the options for configuring the different units, so for example changing from a double to an integer. Here the difficulty is measured, which means a library with a tutorial will score more points than without.
* *Quality of documentation*  
  This criteria purely focuses on the **quality** of the written documentation. Is the documentation easy to understand? Is the documentation well written and does it contain examples?
* *Amount of documentation*  
  This criteria purely focuses on the amount of documentation written. Is there just one example and no other documentation?  
  Is the documentation complete? Does the library have functionalities that are not explained?  
  Are there tutorials that explain how something works?
* Has conversions  
  This criteria is for checking if conversions are implemented and how difficult they are to use. Automatic conversions will receive a higher rating than manual conversions. That means this criteria will be rated on how difficult it is to use conversions, if they are even implemented. Below is what an automatic conversion would look like, which would be 10 points:  
  A black background with white text

  AI-generated content may be incorrect.

Figure 1: Automatic conversions example

* *External dependencies*This criteria is for checking how many external dependencies a library has. External dependencies are not preferred, since this can cause the library to not work after an external update. For each external dependency it has, 3 points will be subtracted from an original score of 10.
* *CPPCheck result*  
  This criteria is here to check the code quality of the library.  
  There should be no errors and preferably no warnings. No errors or warnings means 10 points, just warnings mean only 5 points and **all** errors (with or without warnings) mean 0 points.
* *Unittest coverage*  
  This criteria is here because unittests are helpful for checking individual components, so individual units such as length and mass. This means an error in a specific unit can be found more quickly.  
  This will be rated in 5 degrees.
  + 0 = no unittests, not useful for examples.
  + 2,5 = some unittests, most likely not useful.
  + 5 = average amount of unittests, can be useful.
  + 7,5 = better than average amount of unittests, is pretty useful.
  + 10 = a lot of unittests, most likely a very high unittest %, is very useful as examples.
* *CMake integration difficulty*This criteria is here to check how difficult it is to integrate the library into CMake. This shall be rated according to the time it takes to integrate this library in example code.
* *C++ coding guidelines*This criteria is to get a view of how much of the library code matches the current Inspiro code guidelines. Version 1.10 of the code guidelines has been used for this research.
* *Diverging unit support*This criteria will rate the supportability of diverging units, such as temperature, and how well they handle the errors that might be caused by this difference. See the chapter 3.3 for explanations about different diverging units. If the library does not provide the functionality to support these diverging units, then it shall receive a 0. For each **supported** unit found in chapter 3.3, it shall receive 2,5 points.
* *Readability of library example code*This criteria will rate the readability and ease of use. The goal of this criteria is to find code that is easy to read and easy to use, so 10 namespaces might not be the best.   
  Bad readability can still result in a knockout, even if the library has already made it to the shortlist. It will only be checked for the shortlist, since that is when actual code shall be written.
* *Built-in units*  
  This criteria is to get an overview of which library already has which units build-in. Below is a list of currently important units. For each unit a library does not have, 1 point will be subtracted from 10.
  + Temperature - SI base unit: Kelvin, K
  + Time - SI base unit: Second, s
  + Frequency - SI base unit: Hertz, Hz
  + Length - SI base unit: Meter, m
  + Angle - SI base unit: Radian, rad
  + Force - SI base unit: Newton, N
  + Mass - SI base unit: Kilogram, kg
  + Voltage - SI base unit: Volt, V
  + Electrical current - SI base unit: Ampere, A
  + Power - SI base unit: Watt, W
  + Resistance - SI base unit: Ohm, Ω
  + Capacitance - SI base unit: Farad, F
  + Inductance - SI base unit: Henry, H
  + Pressure - SI base unit: Pascal, Pa
  + Luminous intensity - SI base unit: Candela, cd
  + Volume - SI base unit: Cubic meter, m3
  + Area/ surface - SI base unit: Square meter, m2
  + Computer memory - SI base unit: Byte (units have to be binary)

It is important to note that computer memory is inherently binary, since this is how computers interpret data. This binary and decimal difference is the difference between a KB and a KiB.  
An explanation can be found in chapter 3.3.1.

Below is a table of what criteria had which weights assigned when doing this research, so the results can be replicated. A weight of 1 is normal, a weight of 2 means twice the original value.

|  |  |
| --- | --- |
| **Medium list criteria** | **Weight** |
| Expandability | 2 |
| Configurability | 2 |
| Quality of documentation | 2 |
| Amount of documentation | 2 |
| Has conversions | 2 |
| External dependencies | 1 |
| Unittest coverage | 3 |

Table 6: Medium list criteria

|  |  |
| --- | --- |
| **Shortlist criteria** | **Weight** |
| CPPCheck result | 2 |
| CMake Integration Difficulty | 1 |
| Diverging unit support | 2 |
| Readability of example | 1 |
| Build-in units | 1 |
| C++ Coding Guidelines | 2 |
| Heap and stack usage | 1 |

Table 7: Shortlist criteria

These weights were gathered in an initial offer by me, since I had researched a few libraries already and knew what the differences were. This initial offer was shown to the client, after which some changes were made. For example, the unittest coverage was changed to 3 (and some other changes as well).

While doing the research, the analysis for used units within Inspiro showed that readability was the biggest concern, but this was already taken into account. This is because the client had previously stated that bad readability would still be a knockout. Therefore no changes were made to these weights after the analysis results had been processed.

## Background information about unit library problems

### Difference between binary and decimal memory types

A lot of people confuse the binary memory system and the decimal mathematical system, or have not heard of the difference before. This chapter was created after getting a hint from a colleague to be aware of this.

Computer memory is in the binary system, so usage of the decimal system in programming can cause memory issues, since it would be speaking in 2 different languages. This is why the decimal system is coloured red and binary is coloured green (source: <https://physics.nist.gov/cuu/Units/binary.html>).

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **Name** | **Symbol** | **Size** |
| 2^1 | Bit | b or bit | 1 bit |
| 2^2 | Nibble | - | 4 bits |
| 2^3 | Byte | B | 8 bits |
| 10^3 | Kilobyte | KB | 1000 B |
| 2^10 | Kibibyte | KiB | 1024 B |
| 10^6 | Megabyte | MB | 1 000 000 B |
| 2^20 | Mebibyte | MiB | 1 048 576 B |
| 10^9 | Gigabyte | GB | 1 000 000 000 B |
| 2^30 | Gibibyte | GiB | 1 073 741 824 B |
| 10^12 | Terabyte | TB | 1 000 000 000 000 B |
| 2^40 | Tebibyte | TiB | 1 099 511 627 776 B |
| 10^15 | Petabyte | PB | 1 000 000 000 000 000 B |
| 2^50 | Pebibyte | PiB | 1 125 899 906 842 624 B |
| 10^18 | Exabyte | EB | 1 000 000 000 000 000 000 B |
| 2^60 | Exbibyte | EiB | 1 152 921 504 606 846 976 B |

Table 8: Binary and decimal memory comparison

As is visible in the table above, this small difference can lead to a different memory size than expected, which can cause unwanted memory issues.  
For example, since computer memory is always in the binary system, this means 1024 bytes is equal to a kibibyte (KiB). If you expect the 1000 bytes from a kilobyte (KB) and you only have 1010 bytes of space remaining, this can cause issues with memory overflow.

### Affine- and vector spaces

Sources for this information: <https://en.wikipedia.org/wiki/Affine_space>,  
<https://aurora-opensource.github.io/au/main/discussion/concepts/quantity_point/#temperatures-are-error-prone> and <http://videocortex.io/2018/Affine-Space-Types/>

**Explanation:**

In many physical systems, quantities can belong to either affine spaces or vector spaces.   
A vector space can be easiest explained as the difference between 2 points. This means it supports operations like addition and scalar multiplication.  
   
In contrast, an affine space is a specific point, which means that it cannot be added to another affine space.

Now onto an example:

The temperature outside is 15 degrees Celsius, meanwhile inside it is 20 degrees Celsius.  
These are two absolute points, so both are affine spaces. This means we can subtract them to get the difference. This gives us a difference of 20-15 = 5 degrees Celsius without a location/ point.

This difference is a vector space, which means we can add, multiply and do all sorts of different mathematical operations to it. We can multiply 5 degrees Celsius by the scalar 2 to get 10 degrees Celsius as difference and this would still have meaning.

This is not the case for the affine space.  
If we try and add the outside temperature to the inside temperature, you get a temperature that has no meaning. So 15 degrees outside + 20 degrees inside = 35 degrees without a location, so this gives us an useless number of degrees. This is only a scalar and has no meaning anymore as a temperature.

**Why it is a problem:**

A C++ unit library that treats all quantities as if they belong to a vector space might allow operations that make no physical sense (for example: adding two absolute temperatures, like in the example above).

Not all libraries have the functionality to distinguish between these different spaces, which reduces the typesafety of the library.

An example of why this is a problem for such a library can be found below:

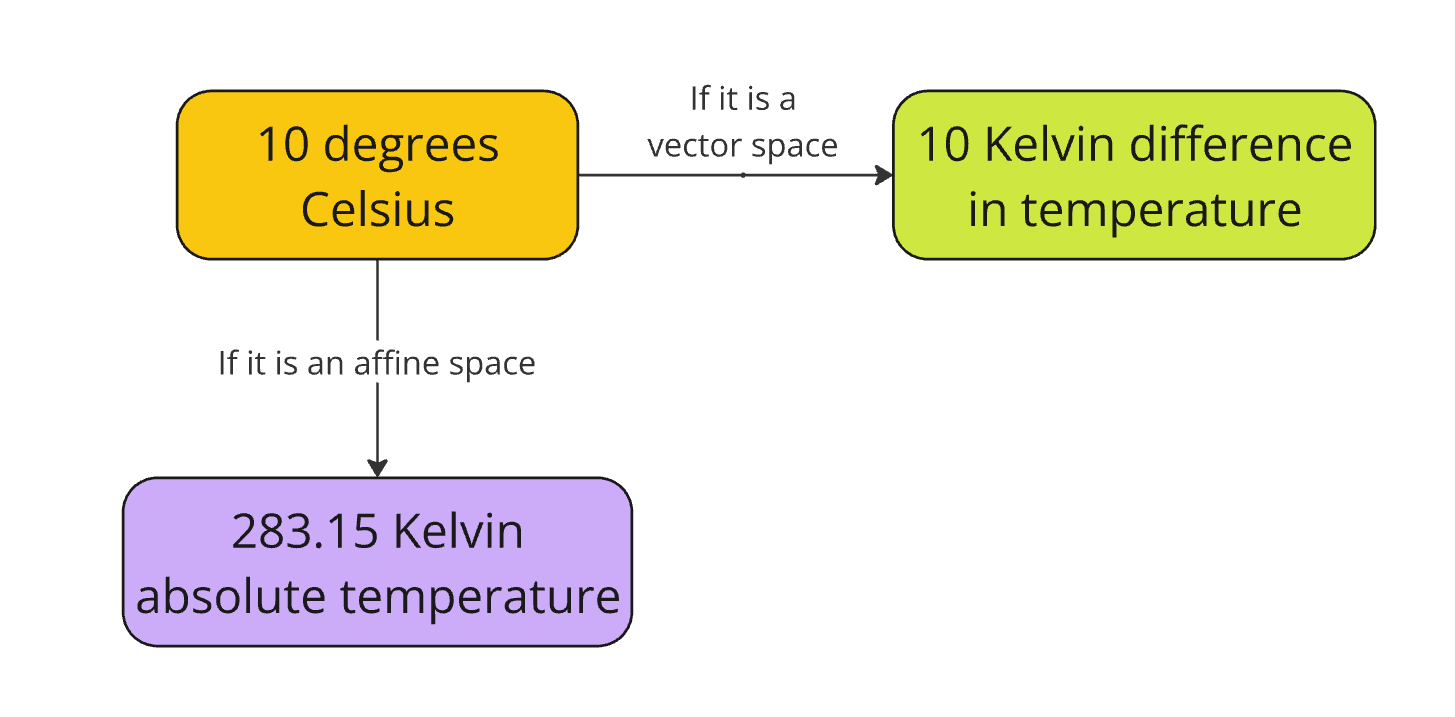


Figure 2: Explanation of affine- and vector spaces

In the above picture, you can see that 10 degrees Celsius alone does not know what to convert to when converting to Kelvin. The space needs to be specified in order to be able to use the correct conversion.  
If a library does not have this, it cannot provide the correct conversion between (in this case) temperature.

**How this can be solved**:

In the Au units library this has been solved by added a quantity point. Normally all units are quantities, but specifically for temperature and other affine spaces this quantity point has been added. This means a room temperature will be a quantity point, while a difference in temperature will be a standard quantity. See this link for more information:   
<https://aurora-opensource.github.io/au/main/discussion/concepts/quantity_point/#temperatures-revisited>

### Decibel and other logarithmic units

**Explanation:**

Decibels and other logarithmic units have conversions and ratio’s with logarithms. This complicates the use of these units in a library, since they don’t use the standard methods.  
This means they do not scale linearly like length or weight. For example, an addition of 10 dB represents a tenfold change in power.

**Why it is a problem:**

As stated above, these types do not scale linearly.  
This requires a special way of handling these types, other than the standard quantity or quantity point we have seen so far.

**How this can be solved**:

In the [nholthaus units library](https://github.com/nholthaus/units/blob/master/include/units.h) this has been solved by adding a new scale. This means there is a normal, linear scaling, but also a decibel scaling.   
This way the Decibels can be handled as expected.

### Differentiating between Hz, Bq and Bd

**Explanation:**

Hertz (Hz), Becquerel (Bq) and Baud (Bd) all have the same dimensional representation (1/second). Hertz is used for frequency (cycles per second), while Becquerel is used for radioactive decay (disintegrations per second) and Baud for data transfer (signal changes per second).

**Why it is a problem:**

Some libraries do not differentiate between these, which means a value in Hz can be written as Bq. Or even worse, when adding 1 Hz to 1 Bq and 1 Bd the result will be 3 Hz. This should not be possible and could cause problems with calculations.

**How this can be solved:**

In the MP-units library a quantity kind has been added, on top of the normal quantity and quantity point. See this link for more information: <https://mpusz.github.io/mp-units/latest/blog/2024/10/21/international-system-of-quantities-isq-part-3---modeling-isq/?h=#dimension-is-not-enough-to-describe-a-quantity>

# Results

This chapter will cover the results of the research into C++ unit libraries.

## Highest Common Denominator C++ version

To know which C++ version would mean a knockout, the highest common denominator must be found.  
Once this is found, we can formulate a substantiated requirement for which C++ versions will become knockouts. Since this will probably be the foundation for future integration into project, it is important that this is renewable with newer toolchain versions.

To find these versions for the STM32, ESP32 and Nordic nRF, the newest toolchain versions had to be found. Not all toolchain websites mentioned the specific C++ versions that were supported. In fact, the ESP-IDF was the only one that specifically named C++23 support. Both the STM32CubeIDE and nRF Connect SDK have no mention of a supported C++ version. This meant it was going to be more difficult to find the C++ version than originally was thought.

In this case, we can compare the compiler versions and find the corresponding C++ version(s). For this research we chose GCC, because this is generally the best-known compiler with the most support.

After getting the GCC versions, the following site was used to get the corresponding C++ version:   
<https://gcc.gnu.org/projects/cxx-status.html>.   
For example, GCC 12 supports all functionalities from C++20, but not everything from C++23. This means that the highest (fully supported) C++ version is C++20, even if it does support a few C++23 functionalities.

All of this is summed up in a table that can be found within Appendix 1: called “Highest common denominator”  
All the used sources are visible there as well.   
To get a short summary, the following table may also be used, but this does not include the sources:

|  |  |  |  |
| --- | --- | --- | --- |
| Microcontroller | Latest toolchain version | GCC version | Highest C++ version |
| ESP32 | [ESP-IDF v5.4](https://github.com/espressif/esp-idf/tree/v5.4) | [gcc-14.2.0](https://docs.espressif.com/projects/esp-idf/en/v5.4/esp32/migration-guides/release-5.x/5.4/gcc.html) | 23 |
| STM32 | [STM32CubeIDE v1.17.0](https://www.st.com/en/development-tools/stm32cubeide.html#get-software) | [arm-none-eabi-gcc-12.3.1](https://developer.arm.com/downloads/-/arm-gnu-toolchain-downloads/12-3-rel1) | 20 |
| Nordic nRF | [nRF Connect SK v2.9.0](https://github.com/nrfconnect/sdk-nrf/tree/v2.9.0) | [gcc-12.2.0](https://github.com/zephyrproject-rtos/sdk-ng/releases/tag/v0.17.0) | 20 |
|  |  | Highest common  denominator version | 20 |

Table 9: Highest common denominator table without sources

In conclusion, the highest common denominator at the time of researching this (March of 2025) is C++20.  
This means that all C++23 and higher libraries are a knockout criterion.

## Knockout table/ long list

After having found the C++ version that would result in a knockout (highest common denominator C++ version), we can move on to the knockout table. This can also be called the long list, since this is where the list of libraries is the longest.

To get a good understanding of the libraries and have at least a decent chunk of the available libraries, the minimum number of libraries was set to 20.  
This minimum amount was easily reached and eventually there were 36 libraries found for this longlist.

This longlist, including all these libraries, can be found in Appendix 1: in the specific table called “Knockout table (long list)”.

I do not believe that Inspiro’s criteria made it impossible to gather good libraries, since most of the better quality libraries were suitable. A handful of libraries even had embedded microcontrollers in mind when being created, which would mean this is not an Inspiro specific requirement. Therefore I suspect there are existing libraries that can be used by Inspiro and creating a custom library is not needed.

An overview of all libraries that went on to the medium list can also be found in Figure 3.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 3: Screenshot of the long list including all libraries that continued to the medium list

## Medium list

From the previously named 36 libraries, only 20 made it to the medium list.

This medium list is made to rank the libraries on the less time-consuming criteria, such as amount of documentation. These criteria have specifically been chosen to not include any criteria that can only be tested by running or building the code, such as CPPCheck. All criteria that must be tested by having running code have been added to the shortlist, since this reduces the time needed for this research. This was discussed with the client.

Furthermore, once all points were given, the weights found in Table 6 were applied. This resulted in an (descending) ordered list, which has been summarised by selecting the top 5 in Table 10.

|  |  |  |
| --- | --- | --- |
| Name of library and link | Total points (70 is max) | Total points with weights applied (130 is max) |
| [Au](https://github.com/aurora-opensource/au) | 70 | 130 |
| [nholthaus/units](https://github.com/nholthaus/units) | 68 | 126 |
| [xo-unit](https://github.com/Rconybea/xo-unit) | 65 | 121 |
| [mp-units](https://github.com/mpusz/mp-units) | 64 | 120 |
| [unlib](https://github.com/gitsbi/unlib) | 63 | 116 |

Table 10: Top 5 libraries from the medium list

To see how these results were gathered, I would suggest looking through them yourself in Appendix 1:. These medium list results can be found in the table “Medium list”.

## Analysis used units within Inspiro

Before we can move on to the shortlist, one of the criteria for this shortlist was the built-in units.  
In order to rate this in the best way for Inspiro, a form was to be created and sent around the workplace.

A few questions were created and put into a Microsoft Forms form. A copy of these questions can be found in Appendix 2 and all answers to these questions can be found in Appendix 3.

To summarize all answers that were given, there were some problems, ideas and concerns or comments about the idea of having a unit library.  
These have been summarized below in chapter 4.4.1 to 4.4.4.  
Lastly, chapter 4.4.5 consists of the results of the filled in units.

All text parts that were underlined are answers to these concerns, problems or comments. This mostly consists of why this has or has not been added in this research or if this problem/ concern could be solved using a unit library.

### Previously experienced problems

* *Prefixes issues (milli, deci, centi etc.)*A common issue raised in the responses was the confusion over prefixes. For example, one program might return a value in milliseconds while the receiving program expects seconds, or the other way around. Similarly, the abbreviations for kibibytes and kilobytes can easily be mixed up, even though they represent different values. These small differences are regularly overlooked, which gives hard to find bugs.Most libraries support prefixes and abbreviations and even have automatic conversions within the same unit, so this problem should not be an issue anymore after correctly implementing such a library.
* *Roundtrip problems*  
  Roundtrip problems occur when the conversion factors between units are so large that they lead to precision loss during these conversions.  
  For example, when converting from nano seconds to days and then back. This would cause the value to be normal in nano seconds, then become extremely small when converting to days and then back to normal in nano seconds. Due to it having become so small after converting to days, this conversion might have rounded off some digits.   
  This loss of precision means that when the value is converted back to nanoseconds, it no longer exactly matches the original value, which can cause issues in applications that require high precision.  
    
  This problem is not something a unit library can solve, since the unit library would have the same issue with converting these values. This is a problem that can be solved by the programmer by not switching back and forth between such large differences.
* *Complex methods of storing a temperature*  
  One of the answers said that temperatures were being stored in an integer or an integer with 0,1 degree accuracy, such as 23,1, being saved as 231. This could be a problem when trying to get a more accurate temperature reading, or when not all developers know about this.   
    
  A unit library will make sure everyone is on the same page about what the value of the temperature means, therefore less issues would arise. It does not provide a way to get a 0,1 degree accuracy with an integer, since this is kind of a workaround and if not strictly necessary, this should be avoided.
* *Unit naming and typing (or lack of)*A lot of previously had problems were because an integer could mean everything. A simple integer of 100 had no semantic meaning, which meant this could be everything. It could be 100 meters, 100 degrees, 100 apples, but the most important difference was with units like meters or feet. This is because these have the same meaning, being for example length, but have quite different values.  
    
  This is the main reason why such a unit library could prove very useful and therefore this problem will be solved with a library. Instead of it being a simple integer without any semantic meaning, a library provides a semantic meaning to a type.
* *Changes made to higher level code, which cause lower level code to not work as expected.*  
  These changes to a different component within a project can, and have, lead to some unwanted errors. For example, the higher level code has a function that originally sent a value in seconds, but after a change was made to send this in milliseconds. All higher level code still worked, but once the unchanged lower level code was added, this did not work, because it still expected seconds.

This problem can be solved by using a units library, since automatic conversions would take care of this. The automatic conversion would take place in the lower level code and convert the value from milliseconds to seconds. This is, of course, if both the higher- and the lower level code include this library.

### Would a library have helped?

These are all summaries based on the answers given in the form. This means that the people who have filled in this form think that this would solve their problem.

* *Automatic conversions, prefixes and abbreviations*Having prefixes built in would have definitely helped with problems for mill, deci, centi conversions. Automatic conversions builds upon this solution by enabling it to not only be visible to the programmer, but by converting it automatically to the desired prefix or unit.
* *Readability*  
  A library would have helped with the readability of what a value means. If the type of a variable already provides enough information about what it is supposed to be, then this saves time searching for it in code. For example, you do not have to go to the comment above the declaration of the variable to know what it stands for.
* *Typesafety*  
  Having type checking for these types ensures more errors are found before running the program.  
  It would have helped a few colleagues with their experienced problems.
* *Switching between system of measurements on the fly*  
  Switching between these systems can likely not be solved with just a library, since these different systems have to be designed in from the start. Whether this is done by a library or somewhere else is not of importance.

### Concerns with future use of a library

Firstly, there were some concerns about the usage of such a library.

* *Dynamic memory allocation*  
  There were some concerns that the library would use dynamic memory allocation, which would not be optimal for use within Inspiro.  
    
  Since it was also stated by the client, this has been added as criteria to the shortlist from the start and no changes were made to the research after this concern. It got combined with the criteria that can be found below.
* *RAM or flash usage*On the topic of memory usage, the RAM and flash usage was also stated in an answer. Since Inspiro works with small microcontrollers that do not have nearly infinite storage, this was also a consideration.  
    
  While researching, the criteria “stack usage” was added to the shortlist to compare the ways in which the library handles or creates the units. After this concern, this criteria has been renamed and expanded to also include RAM and flash usage.
* *License*  
  There was another concern about the license that the library uses, since some licenses force the whole project to be open-source. Since Inspiro sells its software, if this were to become open source, they would lose a lot of revenue. Therefore only licenses that allow for Inspiro to sell this software can be allowed.  
    
  The client stated this as a criteria from the start too, so this has also been taken into account and no changes will be made after this concern.
* *Bad readability/ difficult to understand or read syntax*One of the, if not the, most mentioned concern: readability. There were concerns about declaring your own types, which can clutter workspaces or give barely readable compile errors.   
  Furthermore the template way of implementing such a library, such as the examples given in the form, is too long to be read easily.   
    
  This already was a criteria that was discussed and added to the shortlist, but these answers gave a good explanation of what- and why it matters.   
  5 examples will be written for the shortlist, so the readability can easily be compared between these.  
  Lastly, bad readability is a knockout, even if a library made it to the shortlist.
* *Overhead (or other problems) caused by usage of floating point values*  
  Because Inspiro uses small(er) microcontrollers, this was a popular problem.  
  Smaller microcontrollers tend to not have hardware floating point instructions, so performance could take a hit if a library only uses floating point values.  
  This also applies to heterogenous computing with one core containing a double-precision floating point unit and the other core having no floating point unit at all.  
    
  This was not a criteria from the start, but it did get added before the libraries were even found. This was because not just the possibility is not always there to use floating points, but it reduces the accuracy of values as well.   
  It was added in the medium list as criteria “configurability” and will rate the freedom that is given to choose your own type. For example, an uint8\_t instead of a float or double. This means no changes were made after receiving this concern, since it was already added in the research.
* *Printability of unit types*  
  The last problem that was thought of was the printability of unit types. If these new types from the library are not printable, then this would cause issues for some programs.   
    
  Very few libraries are not able to print their raw- and type values, such as 5 and 5 cm. Some can only print the raw value, but that still means it is printable. Therefore this has not been added as a criteria and no changes were made after this concern.

### Other idea’s or comments

Secondly, there were some other idea’s or comments about the usage of such a library.

* *User defined literals*  
  It got mentioned that someone had experimented with user defined literals before and that this might be an easier solution to his previously experienced problem.  
    
  Even though user defined literals are very good at what they are, they cannot be broadly used. This is because C++ also uses them and therefore not all abbreviations are available anymore.  
  Furthermore, since these are applied to literals (variables known at compile time, such as constants), all runtime variables cannot make use of this.   
  (Source: <https://www.youtube.com/watch?v=3XSVCmWQklI> from 14:45, this includes other reasons as well.)  
  This makes it not very useful to be used across a whole company with many different use cases.
* *Floating point & SI-units without prefactors*  
  Another solution that was given was to use floating point values and to use SI-units without prefactors (like centi, deci or milli) where applicable. It was already stated that this is not always an option due to performance and predictability issues with floating points.  
    
  As stated by the author of this answer, floating point values are not always an option. This solution can work, but is not the best there is and feels more like a workaround than an actual solution. A library would provide a more suitable and stable solution.

### Used units

Since there are multiple questions asked in this form, the answers can seem overwhelming. For this reason it was necessary to extract the named units and put these into a different table. This makes it easier to create an overview of specifically the used units and as a bonus it allows us to view the most mentioned units.

This table can be found here: [C++ units library research units.xlsx](https://inspirobv-my.sharepoint.com/:x:/r/personal/lars_van_duijnhoven_inspiro_nl/Documents/C++%20units%20library%20research.xlsx?d=w47b3a3e771ee4949948da7dd029fb128&csf=1&web=1&e=lQWEsN&nav=MTVfe0ZGOUQ4NkY2LTI3RDItNDQxRS1BQTBFLTdCMTZFRDI3NkE5Q30), but an image of all units that were mentioned more than once has also been added below.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4: Screenshot used units results

After discussing these results with the client, it was decided that **all** the named units should be checked for.  
All units that were mentioned more than once shall be the more important units, while all units that were only mentioned once are less important. For the shortlist criterion called “built-in units” these units shall be compared.

## Overflow protection

Overflow protection with compile time conversion factors, see explanation from Mateusz Pusz:

A screenshot of a black and white text

AI-generated content may be incorrect.

Almost no libraries have decibels except for one, which is the nholsthaus library that was linked to in chapter 3.3.3. Sources: <https://www.youtube.com/watch?v=o0ck5eqpOLc> and   
<https://auroraopensource.github.io/au/0.3.2/alternatives/#library-features>.  
  
Unit libraries in general aren’t complete, but are a work in progress. Sadly for now this means not everything (not all units) is supported.

# Conclusion

In this chapter we look at the results and come to a conclusion, in combination with giving an advice.

This can only be filled in once the research is done, therefore it is empty for now.

# Sources

This chapter includes all the sources used for the research.

Bonestroo, W., Meesters, M., Niels, R., Schagen, J., Henneke, L., & van Turnhout, K. (2018). ICT Research Methods. Amsterdam: HBO-i.

# Research table spreadsheet

This spreadsheet was used for creating the results that can be found is this research paper.  
It has multiple tables that can be switched and referenced to. It also has a versioning, with the currently used one being version 1.0.

It can be found in the same mail as where this research paper can be found. It has been added as an additional attachment and should have the name “V1.0\_Research Table”. The extension should be xlsx, with this being the Excel extension.  
If this is not the case, please message the author of the mail with this problem.

# Form analysis used units within Inspiro

The form that was send to 17 colleagues to hear their responses for which problems they were having with units, if a library might have helped, etc.

A copy of the complete form can be found below.

**C++ units library research form**

The goal of the research for a C++ units library is to have a standard units library for Inspiro which will be included in the CSDD-examples, and thus into future projects.

A **very** in depth explanation by an author of such a library can be found here:  
<https://www.youtube.com/watch?v=pPSdmrmMdjY>.  
  
Some examples of code found in different libraries (this means they use different syntaxes) can be found below as the first question.  
This was due to hitting a word limit in the description for this form...  
  
This form was made in order to create an understanding of which units of measurement are currently used within Inspiro BV.  
Knowing which units are used means the library can be better adapted to Inspiro's needs.  
  
Your responses will be compiled into a table that will be included and referenced in the research paper.   
Please note that this table (including your responses) will be visible to everyone within Inspiro.  
  
I'd encourage you to answer the questions to the best of your knowledge, since your answers will affect the usability of the library.  
If, for example, you use the quantities "length" and "electrical current" often in your work, but you forget to add the "length" unit in your answer, there is a chance that length won't be included in the library.   
  
Lastly, all responses must be provided in English.   
  
If there are any questions regarding the library research or regarding this form, please contact me at lars.van.duijnhoven@inspiro.nl.  
Furthermore, thank you for taking the time to fill out this form! 

**Examples of library code:**

*Example of general code*-  Quantity<Meters, uint8\_t> distance = meters(5);  // Creates a distance of 5 meters   
    std::cout << "Distance in meters: " << (distance + (centi(meters))(100)).as<uint8\_t>(meters) << std::endl; // Prints a value of 6 m  
  
*Example of automatic conversions* -  static\_assert(180\* km / (2 \* h) == 25 \* m / s); // Succeeds because of automatic conversions  
  
*Example of temperature, since this has to include affine spaces. See this link for more information: https://en.wikipedia.org/wiki/Dimensional\_analysis#Affine\_quantities.* -  quantity\_point temp = point<deg\_c>(20.); // The dot makes it a float  
     std::println("Temperature: {} ({})", temp.quantity\_from\_zero(), temp.in(deg\_F).quantity\_from\_zero()); // Outputs "Temperature: 20 ℃ (68 ℉)"  
  
*Example of typesafety* -  Quantity<Seconds, uint64\_t> timeSinceStart = seconds(0);// Initialize time variable      
    timeSinceStart += seconds(2)+ minutes(3)+ hours(1);// Outputs 3782 seconds    timeSinceStart = seconds(2)+ meters(3);// Gives a compiler error  
 *Example of declaring your own types*  
 -  using MetersPerSecondSquared = decltype(Meters{} / squared(Seconds{})); // Declare your own type for usage below  
    Quantity<MetersPerSecondSquared, uint8\_t> acceleration = (meters / squared(second))(2); // Use your own compiletime created type!  
**No answer is required for this question**

|  |
| --- |
|  |

Are there any problems you have had with unit conversions or unit mismatches? If so, please provide examples.

Example of an astronomically big problem:  
<https://en.wikipedia.org/wiki/Mars_Climate_Orbiter#Cause_of_failure>.  
Another example: It was believed that the integer "timeSinceStart" was in seconds, but after having the program crash multiple times it appeared to be in ms.

|  |
| --- |
|  |

Do you believe that a library would have helped solving (or maybe solved entirely) the previously named problems? Please provide an explanation with your answer.

|  |
| --- |
|  |

Are there any problems you foresee that might be had when using a library for the units?

Example: It's uncertain if code **with** the library types could conflict with code **without** the library types due to some form of mismatch between the types.

|  |
| --- |
|  |

What quantities and units do you currently use in your work?

Example: "Temperature" is a quantity and "Kelvin" is a unit that belongs to the quantity "Temperature".  
Combinations such as meter per second also count, it does not have to be a single unit.   
  
The following links can be used as guidance for your answers:   
<https://en.wikipedia.org/wiki/International_System_of_Units>  
and   
<https://en.wikipedia.org/wiki/List_of_physical_quantities>.

|  |
| --- |
|  |

# Answers analysis unsed units within Inspiro

This spreadsheet was used for storing and examining the results that were filled in from the form.  
It has multiple tables that can be switched and referenced to, with “answers” being the raw answers and “units” consisting of an overview of the different units that were mentioned within these raw answers.

The spreadsheet can be found in the same mail as where this research paper can be found. It has been added as an additional attachment and has the following name: “C++ units library research answers”.  
If this is not the case, please message the author of the mail with this problem.