

CoordConvert 1.4.6d

Erik Fløisbonn

February 6, 2009

Contents

1	Getting started	2
1.1	Introduction	3
1.1.1	File formats	3
1.1.2	Download	3
2	Release	5
2.1	What's new?	6
2.1.1	File formats	6
2.1.2	User interface	6
2.1.3	Code	6
3	Design	7
3.1	Model	8
3.1.1	CoordConvert	8
3.1.2	Reader	8
3.1.3	Writer	8
3.1.4	GUI	8
4	Formats	9
4.1	txt writer	10
4.1.1	Variables	10
4.1.2	Example	10
4.2	txt reader	11
4.2.1	Unknown delimiter	11
4.2.2	Example	11
4.3	txt reader stop on mismatch	12
5	Customization	13
5.1	How to create a custom reader/writer	14
5.1.1	A reader needs to	14
5.1.2	A writer needs to	14
5.1.3	Input variables	14
5.1.4	Errors	15
5.1.5	Common methods	15
5.1.6	Examples	15

Chapter 1

Getting started

1.1 Introduction

CoordConvert is a program that can convert between files with point data. The program was originally built to convert points from pxy, kof to csv. As new formats has been added, the program has been reorganized, and it now supports user-created readers/writers. As long as the reader and writer agree on the data they exchange, they can exchange whatever data they want and not just points.

1.1.1 File formats

- It can read points from pxy, kof, csv, dxf (highly experimental), and txt (user defined format).
- It can write points to kof (new), csv, dxf (highly experimental), and txt.

1.1.2 Download

Windows binary

You can download CoordConvert for windows here:

- <http://0x6.org/CoordConvert/CoordConvert.exe>

The program included in the installer is NOT the latest version of CoordConvert. To ensure you have the latest version, update the application by going to File-Update the first time you run the program.

Python module

The main python module can be found here:

- <http://0x6.org/CoordConvert/CoordConvert.py>

and the update/wrapper can be found here:

- <http://0x6.org/CoordConvert/Update.py>

You need Update.py to get CoordConvert.py working, as it defines certain variables CoordConvert.py needs.

Dependencies

- PyQt4
- python 2.5

Source

The source of the project can be found here:

- <http://0x6.org/CoordConvert/src/>

To create CoordConvert.py, run make.sh. The source includes several separate python files that are glued together forming CoordConvert.py by using a macro language I created for this project. The language resembles doctests, but the evaluation of an python expression is inserted into the resulting file.

Documentation

The documentation created with pydoc can be found here:

- <http://0x6.org/CoordConvert/CoordConvert.html>

The manual can be found here:

- <http://0x6.org/CoordConvert/manual.pdf>

Chapter 2

Release

2.1 What's new?

2.1.1 File formats

- Support for writing to and reading from .dxf points in AutoCad files. As I don't have access to AutoCad, you might say that it's experimental. It works with QCaD on Ubuntu Linux.
- Writing to and from .txt files. This new format uses a format specified by the user to interpret the file, and a user specified format for writing to a file.
- Writing to .kof files, 05 koordinatblokk.

2.1.2 User interface

- Dynamic input options for formats. The format classes state what input it needs, and the program handles the storage and retrieval of the input. This means no GUI programming needed in the classes defining the file formats.
- Split the GUI into three groups: input, reader and writer.
- Created *this* window to show longer texts as it's alot better to the eye than the standard dialogs.
- The program checks for updates automatically. It's done in it's own thread to avoid locking up the application if not connected to the Internet.
- Added an options editor so that you can add, delete and change value of input variables.
- "Give feedback" window. It sends the message to me as an email.
- Added a skin. Set the variable CoordConvert.skin to Skanska with the options editor.

2.1.3 Code

- Complete overhaul, bugs galore!
- New build system - seperated documentation from the code.
- Added script for easy uploading of releases and sourcode.
- Created unittests and framework for testing of conversion between formats.
- Added scripts for creation of a manual, which is a compilation of the documentation in the form of a pdf document.

Chapter 3

Design

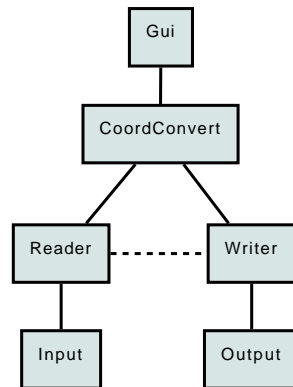


Figure 3.1: Diagram showing the components and the dataflow.

3.1 Model

3.1.1 CoordConvert

This component controls the flow of information between the reader and the writer. It handles situation such as the user stopping the conversion process or the reader/writer encountering errors. When a conversion is done, it generates a report based on the data from the reader and writer.

3.1.2 Reader

The reader knows the format of the input, and has as its job to parse the input and deliver the data to CoordConvert in a format that is recognizable by the writers.

3.1.3 Writer

The reader knows the format of the output, and its job is to convert the data it receives from the reader to its known format.

3.1.4 GUI

This component handles the user interface and decides which reader and writer to use depending on the user input. The GUI is a separate component to make the program work without having to use a GUI, mostly used in testing cases and batch programming.

Chapter 4

Formats

4.1 txt writer

The program supports several file formats which in all cases define x, y and z coordinates of each point. Since some formats also include other information like pointcode, certain formats also define format specific variables. This is a list of the variables and functions one can use with the .txt writer class, which enables the user to specify a custom writer format.

4.1.1 Variables

- $\$(pointcode)$ = pointcode from a .kof file.
- $\$(x)$, $\$(y)$, $\$(z)$ = x, y, z coordinates for each point
- $\$(tab)$, $\$(space)$, $\$(newline)$, $\$(comma)$ = Tab, space, newline, comma
- Functions: $\$+$, $\$/$, $\$-$, $\$*$ on the form $\$(arg1,arg2,...) = i arg1 + arg2 + \dots$

Note: Valid arguments to functions are numbers and variables, as they are evaluated the first pass. Calls to functions are invalid arguments.

4.1.2 Example

- To write "x,y,z" for all points, use the format " $\$(x), \$(y), \$(z)$ ".
- To write "x/2, y-1, z+1" for all points, use the format " $\$/(\$(x),2), \$(y),1, \$(z),1$ ".

4.2 txt reader

This reader enables the user to read files with a custom userdefined format. The reader tries to match the format to each line of the infile, by matching and setting the variable values as it goes from left to right. The reader learns as it moves along, so if you are looking for and matching $\$(x)$, the next $\$(x)$ will only match it's already set value.

4.2.1 Unknown delimiter

If you do not know the delimiter, or if it varies from line to line, you can match each delimiter to it's own variable. The reader then looks for numbers, and uses anything inbetween as delimiter. For example:

- If you have a file with "x,y,z" and "x-y_z", you can use the format " $\$(x)\$(s1)\$(y)\$(s2)\$(z)$ ". This will match both lines and it will set s1,s2 accordingly. If you had used $\$(s)$ instead of s1,s2, the second line would not match as "-" does not equal "_".

4.2.2 Example

- If you have a file with points on the format "x,y;z", you can extract the values by using the format " $\$(x),\$(y);\$(z)$ ".

4.3 txt reader stop on mismatch

If you check "stop on mismatch" the program will stop when it encounters a line that does not match the format given. If it is not checked, the program will ignore all lines that do not match.

Chapter 5

Customization

5.1 How to create a custom reader/writer

This program is written in Python, and you can create your own reader/writer by placing a python source (.py) file in the custom format directory. The file must contain a class definition, that inherits "reader" and/or "writer".

5.1.1 A reader needs to

- Inherit reader
- Overload the next(self) method, which is called by the program until it receives an exception.
- Call return_point(dict) in next() with a dictionary with x,y,z and other format specific data defined as strings.

The infile can be referenced with self.file. As with all file objects, self.file.next() gives the next line of the file. You can read line by line, or byte for byte with fread.

5.1.2 A writer needs to

- Inherit writer
- Overload the write(self, dict) method, where dict contains variables from the reader. These are x, y, z and other format specific data like pointcode. This method is called for each point found by the reader.
- Call write_point(string) in next() to write point to the outfile.

5.1.3 Input variables

If the format needs input from the user, you can define a set of input variables. You do this by overloading the writer_variables(self) for a writer and reader_variables(self) for a reader, which should return a list of dicts. Each dict must contain:

- "name": name of the variable (string)
- "type": type of variable, either "string", checkbox with string called "cstring", or "checkbox". (string)
- "value": default value, True/False for checkboxes, text for strings. (bool) or (string)
- "label": optional label (string)
- "description": optional description. (string)
- "priority": optional priority, used to set the position of the widget. 99 is default and lowest position. (int)

In the reader/writer you can refer to the input variables with self.get_var("name") for strings, and self.is_checked("name") for checkboxes. The values of each input variable is stored between program runs, so the default value is not forced. If you want to change the value of an input variable, use the options editor.

5.1.4 Errors

If you reader or writer encounters a error, you can stop the conversion process by calling `self.stop("message")`. If you want to show a warning you can call `self.warning("warning")`. But please do not call `warning()` in `next()` or `write()`, as this will show a warning for each point found. The beste place for a warning is in `before()` or `after()`.

5.1.5 Common methods

- `before(self)` is called before the conversion begins, and `after(self)` is called after. These can be used to add headers or tails.
- `writer_report(self)` and `reader_report(self)` are methods you can overload to give reports of the conversion. These should return html formated text, and will be shown when the user wants to look at a detailed report of the conversion.
- `self.stop("error")` and `self.warning("warning")` are methods you can call to stop the conversion process or give a warning to the user.

5.1.6 Examples

If you want examples of readers/writers please look at `CoordConvert.py` (located at your "documents and settings" folder i windows) and at the classes `csv` and `text`. `csv` is small class and should be a good starting point - it reads comma seperated files. `txt` is a larger class, and it uses more of the methods described above including input variables.