Project Proposal for Google Summer of Code 2019 with the Eclipse Foundation

Eclipse SWT Chart - Extending the export options

Mentor - Philip Wenig
Student - Sanatt Abrol

Contact Details
Address : 53, Resham Ghar Colony, Jammu, India, 180001
E-Mail : sanatt.abrol.in@gmail.com, sanattabrol@gmail.com
Phone : +917889875852, +917292887928
GitHub : github.com/mavrk

Contents

1. Project Proposal
    a. General aspects
    b. Goals
        i. Exporting to PDF
        ii. Exporting to SVG
        iii. Exporting to EPS
        iv. Update clipboard with LaTeX table
    c. Bonus Tasks
2. Timeline
3. Personal Information

Project Proposal

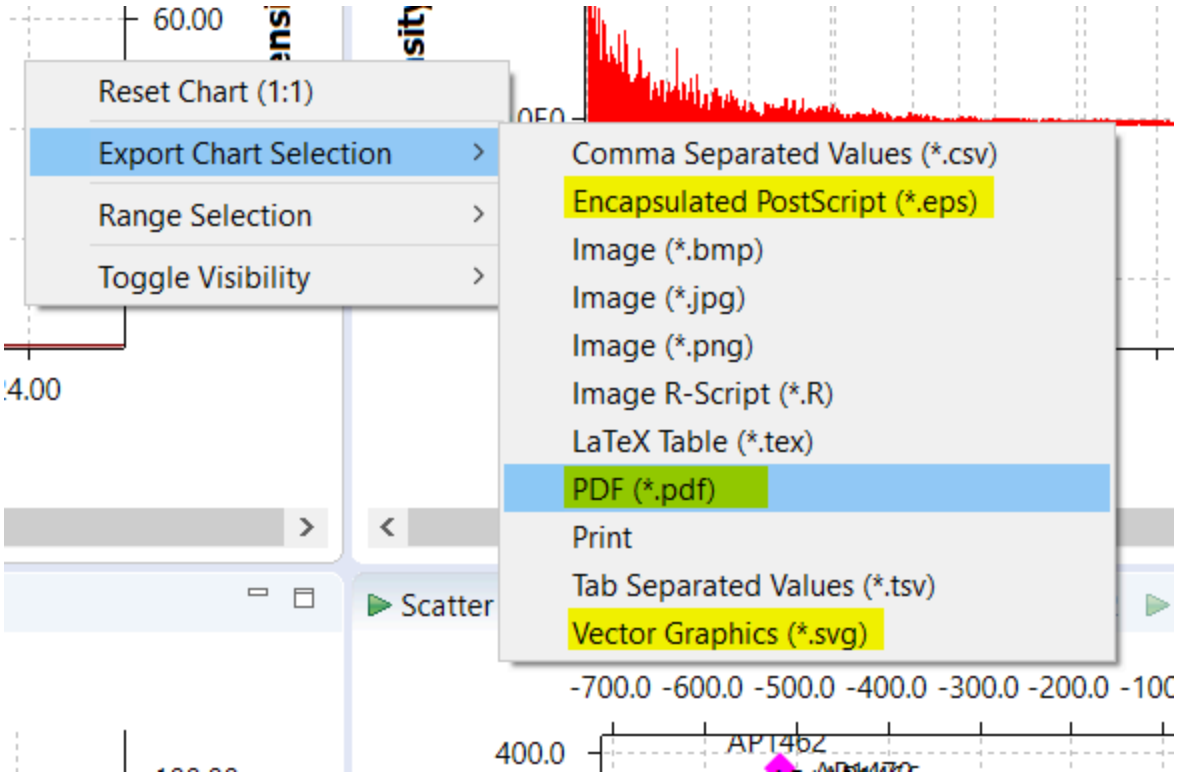
1.a. General Aspects

Eclipse SWTChart allows to create different types of charts. The API is well designed and allows to create **Line**, **Bar** and **Scatter charts** easily. In addition to that, charts can be created even more easily with the **SWTChart extensions**. It uses the **convention over configuration pattern** and offers many additional improvements to scale axes of different type automatically or to select specific data ranges.

This project aims to extend the export options already available under the SWTChart extensions menu. Currently available export options are PNG, JPG, BMP, CSV, LaTeX, R Script.

New export options to be added are
.PDF (Portable Document Format)
.SVG (Scalable Vector Graphics)
.EPS (Encapsulated PostScript)

These new export options would be available to the user as illustrated below:



Another goal of the project is to put a pre-configured LaTeX table into the clipboard.

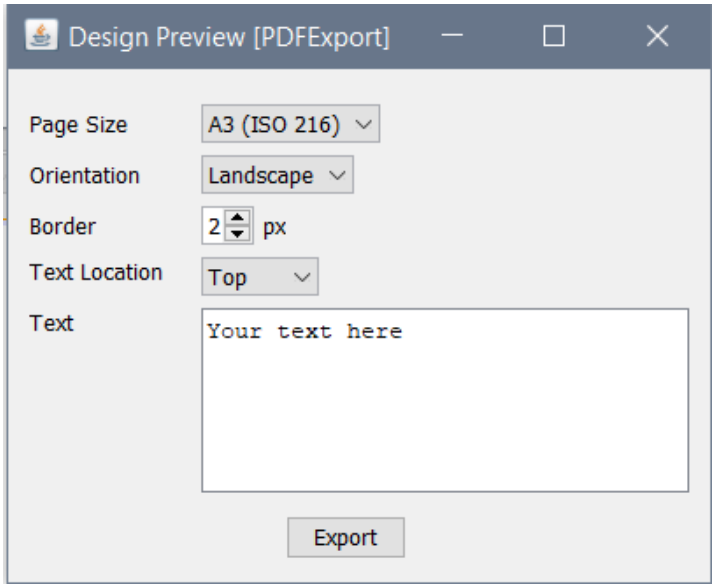
1.b. Goals

1.b.i. Exporting to PDF

PDF is one of the most popular document formats. Adding a PDF export option to the swt-chart menu is the first major goal of the project. PDF files can't be created using the built in capabilities of Java language and hence we need to use some form of library for our purpose. Apache PDFBox is an open-source library with a lot of features. Having an open-source license ensures compatibility with the licenses used by the Eclipse foundation.

The user will be provided with a range of options to manage the pdf export like alignment, borders, text, page size, orientation, etc

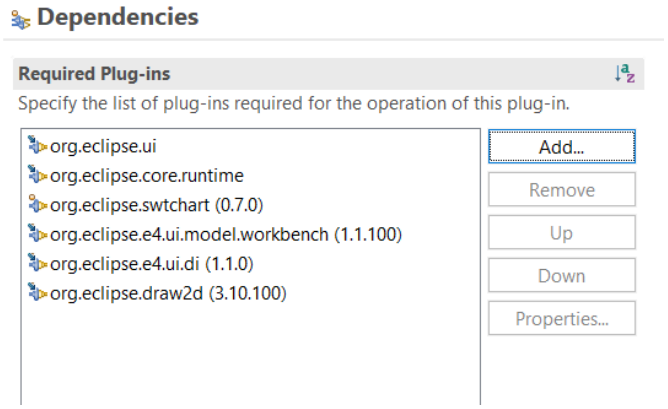
The *PDFExportHandler* will invoke a *PDFSettingsDialog* to enable the users to customise their export.



After selecting desired options, the user is presented with a FileChooser to select the name and path of the exported PDF file.

The first task is to update the Eclipse Orbit repository, so that it includes the latest version of the Apache PDFbox library (2.0.14 at the time of writing this proposal).

The dependency *org.apache.pdfbox* can then be added to our project manually or using the Eclipse wizard.



The Eclipse Chemclipse project has a pdfbox utility which enables us to use this library in a more convenient manner. `org.eclipse.chemclipse.pdfbox.extensions` is the package

Please note this code snippet is only for illustration

```
public class PDFExportHandler extends AbstractSeriesExportHandler
implements ISeriesExportConverter {

	private static final String FILE_EXTENSION = "*.pdf";
	public static final String NAME = "PDF (" + FILE_EXTENSION + ")";
	private static final String TITLE = "Save As PDF";

	@Override
	public String getName() {

		return NAME;
	}

	@Override
	public void execute(Shell shell, ScrollableChart scrollableChart) {

		FileDialog fileDialog = new FileDialog(shell, SWT.SAVE);
		fileDialog.setOverwrite(true);
		fileDialog.setText(NAME);
		fileDialog.setFilterExtensions(new String[]{"*.jpeg",
"*.jpg"});
		//
```

```
            String fileName = fileDialog.open();
            if(fileName != null) {
                  /*
                   * Select the format.
                   */
                  ImageSupplier imageSupplier = new ImageSupplier();
                  ImageData imageData =
imageSupplier.getImageData(scrollableChart.getBaseChart());
                  File file = new File(fileName);
                  PDFdocument document = new PDFDocument.load(file);
                  PDFPage page = document.getPage(0);
                  PDFImageXObject imageObject =
PageUtil.getPDFImageXObjectfromImageData(imageData);
                  PDPageContentStream contentStream = new
PDPageContentStream(document, page);
                  contentstream.drawImage(imageObject, HEIGHT, WIDTH);
                  contentstream.close();
                  document.save(fileName);
                  document.close();
                  MessageDialog.openInformation(shell, TITLE, MESSAGE_OK);
            }
      }
}
```

The above code is only for illustration. The method getPDFXObjectfromImageData() is the method which converts an `org.eclipse.swt.graphics.ImageData` object to an `org.apache.pdfbox.pdmodel.graphics.image.PDImageXObject` object.

1.b.ii. Exporting to SVG

SVG files are nothing but XML graphics files. We can generate SVG files for our charts using the Apache XMLGraphics project. This project has an open-source license and is already available under Eclipse Orbit.

The Apache Batik is a sub-project of the Apache XMLGraphics project and has a built in SVGGenerator class https://xmlgraphics.apache.org/batik/using/svg-generator.html

On the Java platform, all rendering goes through the Graphics2D abstract class, which offers methods such as drawRect, fillRect, and drawString. There are specialized implementations of

this abstract class for each type of output, such as a screen or a printer. SVGGraphics2D is a new implementation of that interface that generates SVG content instead of drawing to a screen or a printer.

Generating SVG file from our chart is a three-step process.

Please note this code snippet is only for illustration

1. Create an instance of *org.w3c.dom.Document* that the generator will use to build its XML content, and create an SVG generator using the Document instance.

```
// Get a DOMImplementation.
DOMImplementation domImpl =
    GenericDOMImplementation.getDOMImplementation();

// Create an instance of org.w3c.dom.Document.
String svgNS = "http://www.w3.org/2000/svg";
Document document = domImpl.createDocument(svgNS, "svg", null);

// Create an instance of the SVG Generator.
SVGGraphics2D svgGenerator = new SVGGraphics2D(document);
```

2. Convert our imageData into a Java Graphics2D object. This is a method we need to create which converts an *org.eclipse.swt.graphics.ImageData* object to a native *java.awt.Graphics2D* object. This Graphics2D object can then be used to paint our svg file.

```
// Ask the test to render into the SVG Graphics2D implementation.

Utl.paint(imageData, svgGenerator);
```

3. Stream out the SVG content. The SVG generator can stream its content into any java.io.Writer. In our example, we stream the content to the standard output stream:

```
// Finally, stream out SVG to the standard output using
// UTF-8 encoding.
boolean useCSS = true; // we want to use CSS style attributes
Writer out = new OutputStreamWriter(System.out, "UTF-8");
svgGenerator.stream(out, useCSS);
```

The .svg file generated can then be saved in a location of user's choice.

1.b.iii. Exporting to EPS

EPS is a file extension for a graphics file format used in vector-based images in Adobe Illustrator. EPS stands for Encapsulated PostScript. An EPS file can contain text as well as graphics. It also usually contains a bit map version of the image for simpler viewing rather than the vector instructions to draw the image.

Apache XMLGraphics project has built-in capabilities to create .eps files. An example is :

```
EPSDocumentGraphics2D g2d = new EPSDocumentGraphics2D(false);
g2d.setGraphicContext(new org.apache.xmlgraphics.java2d.GraphicContext());

//Set up the document size
g2d.setupDocument(out, 400, 200); //400pt x 200pt
//out is the OutputStream to write the EPS to

g2d.drawRect(10, 10, 50, 50); //paint a rectangle using normal Java2D calls

g2d.finish(); //Wrap up and finalize the EPS file
```

This code creates an eps file with a rectangle in it. For creating eps files from charts, we have two possible ways:

1. Write methods to convert *org.eclipse.swt.graphics.ImageData* to *java.awt.Graphic2D* and then create an EPSDocumentGraphics2D object
2. Create a .svg file and then convert it to .eps using ApacheFOP (a sub-project of Apache XMLGraphics).

1.b.iv. Update clipboard with pre-built LaTeX table

The extension menu already has a *LaTeXTableExportHandler* which generates LaTeX table from the selected chart. However, instead of saving this table as a .tex file we can copy it to the system clipboard, which is useful in some applications. This can be done using the *org.eclipse.swt.dnd.Clipboard* class.

We can add an option to copy to clipboard when LaTeXTableExportHandler invokes ExportSettingsDialog. Instead of writing to the PrintWriter, we can convert it into a String and paste to clipboard using

```
Clipboard clipboard = new Clipboard(display);
        String textData = this.getTableAsString();
        TextTransfer textTransfer = TextTransfer.getInstance();
        Transfer[] transfers = new Transfer[]{textTransfer};
        Object[] data = new Object[]{textData};
        clipboard.setContents(data, transfers);
        clipboard.dispose();
```

getTableAsString() method will return a String of the LaTeX table generated.

1.c. Bonus Tasks

If time permits, I'd like to work on some other tasks like

1. Support Mirror mode (https://github.com/eclipse/swtchart/issues/23).
2. Export to .png file format (with an option to add transparency to our image).
3. Create default, reusable charts.

2. Project Timeline

May 6th - May 27th	**Community Bonding** Get to know more about the community and the project. Set-up the developer environment. Develop a good understanding of libraries (Apache PDFBox and XMLGraphics project) by working on examples.
May 27th	**Coding Period Starts**
May 27th - June 24th	Work on adding the .pdf export option and create a PDFSettingsDialog to offer different export options. Also, add tests to ensure the export tool works as intended. Start working on the .svg export option.
June 24th - June 28th	**First Evaluation**
June 28th - July 22nd	Finish working on .svg and .eps export options. Also, add tests to ensure proper coverage of my work and that the tool works as intended. Complete the copy to clipboard option for copying pre-built LaTeX tables to system clipboard.
July 22nd - July 26th	**Second Evaluation**
July 26th - August 10th	Finish any portion left of the primary objectives and then start working on bonus tasks.
August 10th - August 19th	Start documenting code, create a final blog post and ensure there are no bugs or blockages in the code.
August 19th - August 26th	**Final Evaluation**

Disclosures :

1. I am attending a conference in San Francisco, USA from June 10th to June 17th and won't be able to work during this whole week.
2. From May 6th to July first week, I am completely free and I can work upto 8 hours daily.
3. I will be joining Microsoft as a full time employee sometime in July and it'll take me time to adjust and maintain the same working balance again. However, I can ensure you that this project will remain a priority to me and I will complete it under any circumstance.

3. Personal Information

Who am I? What am I studying?

I am a final year student pursuing Bachelors in Engineering in Computer Science from Birla Institute of Technology, Mesra, India. I love playing football, swimming, hiking and watching crime thrillers. I am a huge Manchester United fan and I can speak English, Hindi and Punjabi. I have elementary knowledge of German as well.

One of my goals in life is to travel every continent on Earth :)

My technical experience?

I have been coding since high school mostly on Java, JavaScript and related frameworks. My work includes free software, hackathon projects and contribution to open-source. I have previously been a GSoC student (2017) and a GSoC mentor (2018) for OpenMRS where I worked on a project to provide secure access to patient data to users and services.

Last semester, I created a document search engine for the college network as my final-year project.

I am currently working on a project which enables peer-to-peer live video streaming, something which will work together will traditional CDNs and greatly improve the streaming experience.

I have experience with Java and GUI frameworks like Swing and JavaFX. To know more about what kind of projects I have worked on, please see my GitHub profile https://github.com/mavrk.

Apart from Software Development, I have a keen interest in hacking, security and networks.

Why Eclipse? And why this project?

I was honored to be selected as a GSoC student in 2017 and it proved to be a huge learning experience for me. As a young software developer, I think there is no better platform to learn and grow than open-source. Eclipse ships my favorite IDE and is one of the largest open-source organisations around. As a student, it was an easy choice for me to try and work for Eclipse foundation.

Last year, I took ‘Computer graphics and multimedia’ as a course and it really got me into graphics. Eclipse SWTCharts is a great tool for visualising knowledge in the form of bar charts, line charts, etc. This project has a tech stack I am familiar with and quality of deliverables is also very good. If selected, I am sure it will be an amazing project to work on and learn from.

My expectations and future plans

Learn. That's all I really want to do. I am going to be joining Microsoft as a Software Developer and this is probably my last opportunity to work on a GSoC project. Whether or not I am selected, I wish to contribute to the Eclipse foundation and this project even after GSoC and help in maintaining the project by constantly adding new ways to improve and reviewing pull requests.