

Eiffel SDL multimedia library (ESDL)
DIPLOMA THESIS

Till G. Bay

24th September 2003

Project Period: Monday, 2003-05-26 - Friday, 2003-09-26

Student: Till G. Bay (tillbay@student.ethz.ch)

Student-No: 98-803-000

Supervising Assistant: Karine Arnout

Supervising Professor: Bertrand Meyer

Contents

I	Description	5
1	Project description	5
1.1	Overview	5
1.2	Scope of the work	6
1.3	Intended results	7
II	Project Steps	8
2	Evaluation	8
2.1	Evaluation of SDL Wrappers	8
2.1.1	Evaluation of EWG	8
2.1.2	Evaluation of jegl	8
2.1.3	Conclusion	9
2.2	Evaluation of existing multimedia libraries for Eiffel	9
2.2.1	Evaluation of EiffelFURY	9
2.2.2	Evaluation of Eiffel.SDL	10
2.2.3	Conclusion	10
3	Usability discussion	11
3.1	Demo Creation	11
3.2	Discussion of the found usability flaws	12
4	Redesign	13
5	Applications	15
6	Extensions	16
III	State and future of ESDL	17
IV	Project Organization	22
7	Background material	22
7.1	Reading list	22
8	Project management	22
8.1	Objectives and priorities	22
8.1.1	Design objectives	22
8.1.2	Objectives for the intended results	23
8.2	Criteria for success	23
8.3	Method of work	24
8.4	Quality management	24
8.4.1	Documentation	24
8.4.2	Validation steps	24

9	Plan with milestones	25
9.1	Project steps	25
9.2	Deadline	26
V	Appendices	27
A	Map Widget	27
A.1	Requirements	27
A.1.1	Traffic project	27
A.1.2	Map Widget	28
A.2	Design	29
A.2.1	Displaying a city and its traffic infrastructure	29
A.2.2	Displaying a simplified representation of the city's geographical features	29
A.2.3	Displaying transportation links of various kinds	29
A.2.4	Displaying monuments and other typical landmarks	30
A.2.5	Displaying traffic participants moving around in the city	30
A.2.6	Zooming and panning of the visualization of the city	30
A.2.7	Implications resulting from the requirements	31
A.2.8	EV_CANVAS	32
A.2.9	DRAWABLE_OBJECT	35
A.2.10	ANIMATABLE	36
A.3	Tutorial and example application	38
A.3.1	Step 1, creating a window	39
A.3.2	Step 2, placing the canvas and setting the world coordinate system	40
A.3.3	Step 3, adding polygons, lines, icons, spots and circles	42
A.3.4	Step 4, adding zooming and panning	44
A.3.5	Step 5, adding animations	45
A.3.6	Step 6, changing the canvas dimension	46
B	HELLO_WORLD_SDL	47
C	HELLO_WORLD_ESDL	50
D	Other applications	52
D.1	Coordinate application	52
D.2	Fractal application	53
E	Acronyms	54
F	Versions of libraries used	55
G	Additional resources	56
G.1	Map Widget tutorial files	56

List of Tables

1	Design objectives and their priorities	22
2	Priorities of intended results	23

List of Figures

1	Generation of the wrapper code	10
2	Use of EWG's abstraction layer in ESDL	11
3	Screenshot of SDL Hello World	12
4	Pixel drawing	14
5	Surface moving on another surface	15
6	Lovely flying in Zermatt	16
7	Bounding boxes	30
8	Clipping regions	32
9	EV_CANVAS	33
10	EV_CANVAS's coordinate systems	34
11	Map Widget example application	38
12	The window after step 1	39
13	The window after step 2	40
14	The application with all graphical objects	42
15	The application with zooming support	44
16	Distorted world coordinates	46
17	Coordinate application	52
18	Fractal application	53

List of Code-listings

1	Intersection of two rectangles	31
2	Declaration of <i>visible_area</i> and <i>canvas</i>	40
3	Creation of the canvas	41
4	Set the world coordinate system	41
5	Create a polygon	42
6	Declare some graphical objects	43

Part I

Description

1 Project description

1.1 Overview

Support for building multimedia applications is mandatory for libraries accompanying modern programming languages. Therefore many programming languages provide not only standard libraries that allow a programmer to perform simple tasks like I/O, memory access or string handling, but they also ship with libraries that provide support for video, audio, CD-ROM, multi threading, networking, access to OpenGL [17] and access to input devices such as mice, keyboards and joysticks. Naturally a multimedia library is bigger than one of the mentioned standard libraries, since it tries to harbor all APIs for the various applications in one comprehensive package (more on this in section 8.1 on page 22).

The history of OpenGL [17] shows that the success and the widespread acceptance of a multimedia library depend on the portability of the applications using it. Eiffel [20, 14] is lacking a platform independent multimedia library. This diploma thesis aims to fill this gap by building an object oriented multimedia library in Eiffel based on SDL [18] which itself constitutes a platform independent multimedia library written in C [10].

Here is a summary of the multimedia library options available for Eiffel today and corresponding drawbacks:

- EiffelFURY [5]: a 2D and multimedia library
 - only for Windows
- EiffelOpenGL [3]: an Eiffel wrapper for OpenGL, GLU [7] and GLUT [8]
 - not multimedia, neither sound nor video support
- jegl [11]: a set of wrapper classes around SDL written in SmallEiffel - a previous version of SmartEiffel [19]
 - is incomplete and not maintained anymore
 - since it relies on a previous version of SmartEiffel there may also be compatibility problems, therefore it is also quite obsolete
- Eiffel_SDL [22]: multimedia library for Eiffel
 - under development

This listing shows that there have been and that there are attempts to provide a multimedia library for Eiffel. Unfortunately it also points out the limitations of the available multimedia libraries.

1.2 Scope of the work

The ultimate goal of the diploma thesis is the creation of a cross platform multimedia library for Eiffel. There are two key forms of multimedia library usability - interface and organizational¹. The interface usability dimension comprises the following four aspects:

- **Learn-ability** indicates how fast a programmer can start using the library productively.
- **Memorability** indicates how easily a programmer can continue using the library after not using it for some period of time.
- **Efficiency** indicates the ability to use the library with a high level of productivity.
- **Errors** produced using the library should not be catastrophic, the number of user errors producible should be low, they should be documented and the system should be able to easily recover from them. The library would benefit from a concise logging facility for errors.

On the other hand the organizational dimension can be interpreted as the indicator of how effectively the library can be integrated into working practices of people and organizations using it. The three factors describing organizational usability are:

- **Portability** measures the ability of using the library productively in various environments transparently. In the case of a library for Eiffel, portability among compilers is a second portability requirement which will not be addressed in the scope of this thesis. I will focus on ISE EiffelStudio's compiler. Depending on the tools that I will use it may as well be that the built library is portable among compilers.
- **Compatibility** with other systems or libraries that interact with the library.
- **Extendibility** indicates how easily a library can be extended to perform new functionality.
- **Installation** indicates how easily a library can be installed and used.

Even though both forms of library usability are equally important the diploma thesis will focus on the first form and provide solutions for parts of the second form where it is easily achievable. In other words the interface usability is the main goal of the diploma thesis and the audience in mind are in the first place programmers using the library to write multimedia applications, and not entire organisations embedding the library into their working practices. Organisational usability is in my opinion the second step on the way to a good library and it would require extensions that surpass the scope of this diploma thesis. To illustrate these thoughts one can for example take SDL's audio capabilities: To ensure organizational usability, more precisely to grant the second aspect of it - compatibility - it would be necessary to extend ESDL so it could handle different audio formats - e.g. the ever so popular ISO 11172-3 (mp3).

¹The criteria mentioned here are partly inspired by the judging criteria mentioned on <http://www.eiffel-nice.org/eiffelstruggle/2003/judging.html> [4].

1.3 Intended results

The intended results of the diploma thesis are:

- A cross platform multimedia library for Eiffel

The multimedia library enabling programmers to build multimedia applications with Eiffel on any platform. The multimedia library (hereafter referred to as ESDL) aims at ultimately replacing all other alternatives available for Eiffel today. Therefore it has to be designed very carefully and its building process needs to include adjusting and reviewing.

- Ports of demos available on www.libsdl.org

The graphical demonstration programs (hereafter referred to as demos) illustrate the possibilities of the multimedia library. There are numerous demos available for SDL. Since ESDL is meant to wrap SDL in an object oriented way, it makes sense to port some interesting demos to ESDL to provide future library users with a starting point for their development and to illustrate the possibilities of ESDL.

Part II

Project Steps

2 Evaluation

2.1 Evaluation of SDL Wrappers

The goal is to be able to determine the fitness of the existing wrappers in order to decide whether they will be used in the implementation of ESDL, or whether the wrapping of the underlying C library will be done manually. The term wrappers is somewhat ambiguous since in the case of EWG [12] one cannot consider it a wrapper but rather a wrapping tool. The reason why I put *jegl* [11] and EWG into the same category is that EWG provides a wrapper of SDL among the examples illustrating the use of the actual wrapping tool. Since both components provide an Eiffel wrapper for SDL I decided to evaluate them together - even though EWG on its own is not a wrapper but a wrapper generator.

2.1.1 Evaluation of EWG

EWG (Eiffel Wrapper Generator) is a tool that generates Eiffel wrapper classes for C libraries. It can be used to create libraries that bridge the gap between Eiffel and C. It aims to work for arbitrary ANSI C and with all common Eiffel compilers². EWG is ideal for creating cross platform, cross Eiffel compiler and cross C compiler C library wrappers.

EWG creates wrappers for structures, enumerations, functions and callbacks. So far there is no support for unions and macros. To evaluate EWG I chose to use the omnipresent hello-world example. First I wrote a C library (`helloworld.lib`) providing a function called `print_message()`. Since EWG strongly relies on the *geant* and *gexace* tools included in the Gobo delivery [2] I chose to use them as well. The *geant* tool was used to set up a build environment for `helloworld.lib`. That way the preprocessing of the C header files, the compilation of the C files, the creation of the C glue code and the wrapping of the C library could be automated. The *gexace* tool was used to create an ACE (more on this in [14], Appendix D) file specifically for EiffelStudio. This ACE file was then used to import the wrapped library into an EiffelStudio project.

This process was used to get to know the inner workings of EWG. The wrapped library was then used from within EiffelStudio to produce a Hello World program that used a C function to output a message.

2.1.2 Evaluation of *jegl*

Jegl is a set of Eiffel classes wrapping SDL. Additionally high-level features such as sprites (movable 2D graphics - mostly used in computer games and 2D animations), bit mapped fonts and simple pixel access were planned but never implemented.

Those high-level features would make *jegl* interesting for me as it is exactly the extensions in the object oriented wrapper of SDL that could facilitate the

²ISE Eiffel, SmartEiffel, GNU Eiffel and Visual Eiffel

development of a multimedia application. The downside of `jegl` is that it is not maintained anymore and that even the download of the library from sourceforge is not possible anymore - so I had to contact the developer to get the latest release of `jegl`, dating from July 20, 2000. Furthermore `jegl` is an incomplete wrapper for SDL - only the graphics capabilities of `jegl` have been wrapped. As I chose to target ESDL to the ISE Eiffel compiler it would also require a port of `jegl` since `jegl` was built with the `Smalleiffel` compiler. As a last disadvantage I have to mention that `jegl` is built on top of SDL 1.1.3 whereas ESDL will be built on the latest version of SDL - version 1.2.5.

2.1.3 Conclusion

The ease of use of the wrapping tools provided by EWG will enable future maintainers of ESDL to easily upgrade the version of SDL that ESDL uses without breaking anything. This attribute combined with all the disadvantages of `jegl` lead me to the conclusion that it is better to start developing on top of the wrapped version of SDL contained as an example in EWG rather than completing `jegl` in the scope of this diploma thesis.

2.2 Evaluation of existing multimedia libraries for Eiffel

The evaluation of existing multimedia libraries is used to gather ideas for extending SDL so it can be used easily by multimedia application developers. As I mentioned in the previous paragraph the high-level features that were planned for `jegl` would be one of those ideas that could be realised in the scope of this diploma thesis. Nevertheless there are a few multimedia libraries used by Eiffel developers today, so to have a complete view of the available features I will further examine EiffelFURY [5] (a multimedia library used by Eiffel developers on Windows) and Eiffel_SDL [22] a similar library to the one I am implementing. It has recently been published on sourceforge.net.

2.2.1 Evaluation of EiffelFURY

EiffelFURY is a multimedia library for Windows. It provides support for Window's MCI (Media Control Interface) devices, and adds support for 2D primitives GDI (Graphical Display Interface) enabling the programmer to create simple multimedia or graphical applications for the 32 bit Windows platforms. EiffelFURY makes use of the `winmm` and `gdi32` dynamically linked libraries found on all 32 bit Windows platforms. Thus EiffelFURY can be considered an extended wrapper for the multimedia libraries found on the different Windows platforms.

The library is mainly used to create games, but it has never been documented. A thorough analysis of the techniques used in EiffelFURY requires both the knowledge of the multimedia libraries existing on Windows and time to experiment with the EiffelFURY's tutorial examples.

To judge whether ESDL would benefit from the techniques used and the API provided by EiffelFURY is difficult. Apparently EiffelFURY provides more support for widget creation and handling than SDL does. Whether it supports pixel based drawing or whether the APIs for sound support were comparable remained unclear due to the lack of documentation.

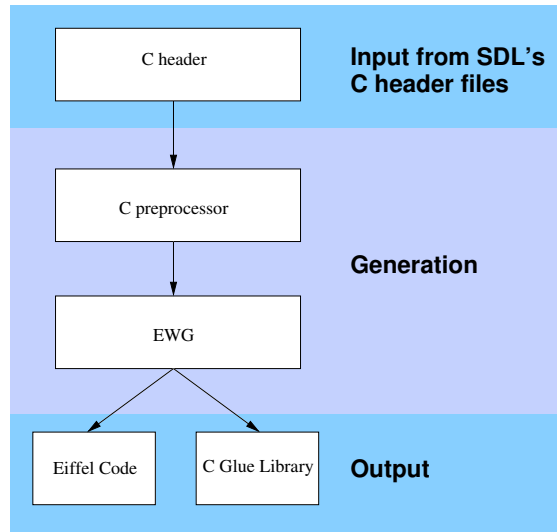


Figure 1: Generation of the wrapper code

2.2.2 Evaluation of Eiffel_SDL

Eiffel_SDL is developed by Steve Thompson as part of the Colorado Eiffel User Group [23]. The library is wrapping SDL just like ESDL. One difference - it does not use a wrapper generator like EWG but the calls are wrapped by hand.

2.2.3 Conclusion

The evaluation of EiffelFURY remains incomplete because of the missing documentation, nevertheless it is apparent that EiffelFURY is built onto a platform specific multimedia system. This is the reason why many concepts from EiffelFURY seem familiar to Windows programmers. For a platform independent multimedia library like ESDL this proximity to one platform should be avoided. Therefore EiffelFURY's design won't influence the design of ESDL substantially.

To the contrary Eiffel_SDL can be considered a sibling project of ESDL.

After having evaluated these multimedia libraries for Eiffel I come to the conclusion that SDL will be the library that I will use to create a platform independent multimedia library for Eiffel, hence the title of this diploma thesis: ESDL. I will wrap SDL using the Eiffel Wrapper Generator (EWG) and then build ESDL onto the abstraction layer created by EWG. This process is illustrated by figure 1 and figure 2 on the following page. In figure 1 the two light blue parts of the image stand for the input and output of the wrapping process. In the middle part of the figure the tools involved creating the output are shown and the arrows indicate the sequential steps to achieve a wrapped C library. In figure 2 however the arrows signify that ESDL is using both the Eiffel Code and the C Glue Library that has been generated by EWG.

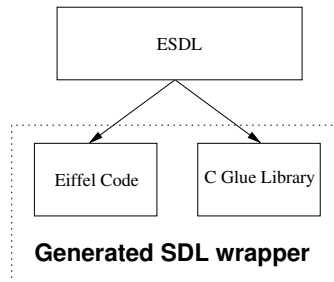


Figure 2: Use of EWG's abstraction layer in ESDL

3 Usability discussion

Before I start with the usability discussion of the wrapper code generated by EWG, let me explain why this diploma thesis does not only contain the documentation of the ESDL project but also the documentation of the Map Widget (see Appendix A on page 27). During the time I started working on ESDL the Chair of Software Engineering at ETH Zurich began implementing a traffic simulation application that will be used in the Introductory Programming course starting late October 2003. For this traffic simulation a widget capable of displaying city maps was needed. Since I was working on a graphics related project, I was also assigned the job of implementing that widget, the Map Widget.

Because other developers were involved in the traffic simulation project, the Map Widget had higher priority than ESDL. To allow the developers of the traffic simulation to advance in their project, I had to start implementing the Map Widget immediately. Of course this changed the envisioned schedule of the ESDL project considerably.

This change of plans is the reason that I did not implement all of SDL's subsystems, but only started implementing the video subsystem of ESDL. Nevertheless, I did not change the table of contents of this document because I believe that the implementation of the video subsystem left me with enough material so that the original layout of the document can be retained. As an example I have not created a demo for all of SDL's subsystems but only for the video subsystem. Let me now talk about that demo.

3.1 Demo Creation

I chose to create a very simple demo of SDL's video subsystem. An application that uses ESDL to display an image seemed appropriate to demonstrate the capabilities of the video subsystem. Because this application shows the basic workings of SDL, I decided to call it SDL Hello World (a screenshot ³ of the application is shown in figure 3 on the following page). It consists of only one class called HELLO_WORLD_SDL (for the following discussion please consider HELLO_WORLD_SDL's source code listed in appendix B on page 47). SDL Hello World is built with the SDL library wrapper generated by EWG. That

³The image shows the three famous swiss mountains Eiger, Mönch und Jungfrau as seen from my hometown Burgdorf.



Figure 3: Screenshot of SDL Hello World

way I was able to find the deficiencies of the abstraction layer of SDL generated by EWG.

3.2 Discussion of the found usability flaws

In the following discussion I am referring to SDL Hello World's source code that you can find in appendix B on page 47.

Three important design objectives I had for this project are: Learn-ability, memorability and efficiency. If you look at `HELLO_WORLD_SDL` class, you see that even for such a simple application we need to inherit from six classes. I think that the three mentioned objectives are hard to satisfy if one has to know all these classes in order to choose the right ones to inherit from. Eliminating the need to inherit from any class or at least reducing the number of classes to inherit from became one of the tasks for the redesign.

The next thing that stands out looking at the class is that the logging or reporting of errors that happen during runtime has to be done manually. For example, integer return codes coming from the underlying C functions must be checked to enable reporting success or failure of the function. In other cases even pointer values must be examined to determine what happened. This is clearly non intuitive and annoying for the developer using ESDL. A central logging facility should be created or at least the logging should be automated and hidden from the developer. Furthermore the error message strings should be externalized in a way that logging and error reporting would be possible in other languages.

Pointers are not only used for logging purposes, but one also needs pointers to create normal objects. For example in the statement `create_display.make_unshared(p)` a pointer p is used to create the main video surface that the user will see. I think that this behaviour should be avoided. One of the reasons is memory management. If I hide all pointer manipulation from the user of the library, I will be able to correct all pointer related memory management problems inside the library. Another reason is that it is absolutely non-intuitive for the user to use pointers to create objects.

The next goal of the redesign is that I wanted to be able to load image files without taking care of the image format. The `SDL_image` library is not part of the SDL library itself, but it is SDL's most important child project and

will eventually be merged with SDL. EWG was used again to wrap SDL and `SDL_image` together. That way I was able to provide a format independent way of loading images.

Perhaps the most important characteristic of SDL is that it does not have its own event loop. In fact it provides all functions and events necessary to implement an event loop, but leaves the design of the event loop to the developer. That way it is possible to create different event loops tailored to the needs one has. For example one could even think of integrating the handling of SDL events inside the event loop of another application framework. Nevertheless I thought a standard event loop should be provided. In `HELLO_WORLD_SDL` you see the absence of any event handling. Instead of entering an event loop, the feature *`sdl_delay_external`* is called with an `INTEGER` delay.

Coming to the bottom of the `HELLO_WORLD_SDL` class you can see two constant declarations:

`Sdl_init_video: INTEGER is 32`, and *`Sdl_swsurface: INTEGER is 0`*. These two constants are used to initialize the video subsystem. EWG does not yet generate macro constants that appear in the C header files. It would be very annoying for a developer using ESDL to go and find out these constants. Therefore all necessary constants should also be provided in a class of ESDL.

Even though this has been a very small and simple example application using ESDL or maybe I should call it EWG SDL it pinpoints already many problems that can be corrected by adding another abstraction layer. Altogether programming applications with EWG SDL leaves the impression of programming in C with Eiffel syntax. Clearly correcting this should be the principal goal of the redesign.

4 Redesign

As mentioned earlier the implementation of ESDL is work in progress. The implementation has advanced far enough to implement an ESDL Hello World application that will be compared to the SDL Hello World that was presented in the previous section. The source code for ESDL Hello World can be found in the class `HELLO_WORLD_ESDL` shown in appendix C on page 50. All usability problems discussed in the previous section have been addressed. I will not make any references to the source code of `HELLO_WORLD_ESDL` directly, but only describe how I have addressed the usability problems. The reader is of course invited to compare the source code of the two classes.

Since SDL consists of eight subsystems and one global system harbouring all of those subsystems and handling their initialisation, I also decided to create a class that would be responsible for initializing the separate subsystems: The class `ESDL`. At the moment only the video subsystem of SDL can be initialized using this class, but as the development of the library continues, the idea is to have all subsystems initialized, enabled and disabled using this class. Not only subsystem handling, but also application handling is provided by this class, for example shut down of the entire ESDL system. Since SDL can also be used without an event loop I did not include the event loop inside the `ESDL` class, but made two additional classes that could be used for event handling: `ESDL_EVENT_LOOP` and `ESDL_EVENT_LOOP_BASE`. While the later is only a helper class for developers that would like to implement their

own event loop, the class `ESDL_EVENT_LOOP` can provide two different event loops. `ESDL_EVENT_LOOP` can be created with two different creation features: `make_wait` and `make_poll`. The difference between the two event loops is that a polling event loop provides the possibility to register callbacks that will be executed outside the event loop. The downside of this functionality is that the actual waiting for events is implemented as a busy wait. In other words all the pending events in the event queue are dispatched and then control goes outside the event loop, the registered callback is executed and then control is passed inside the event loop again.

On the other hand the waiting event loop implements a non busy wait for events but does not allow registering a callback that will be executed outside the event loop. In `HELLO_WORLD_ESDL` I used a non busy event loop because I didn't need to do anything outside the event loop. Generally one should use the waiting event loop, but if you know what you are doing you can of course use the polling event loop.

The mechanism to register callbacks that will be executed after a certain event has happened is provided by the `EVENT_TYPE` [1] class. That way the callbacks can be registered as agents for every kind of event.

The central component of the video subsystem of SDL is the video surface. In ESDL this is the class `ESDL_SURFACE`. `ESDL_SURFACE` harbours all video surface related features. For a detailed description of the features that are implemented see the state of implementation of the video subsystem in part III on page 17.

As you can see in ESDL Hello World the developer does not have to take care of error logging or pointer handling anymore, because the logging facility is merely hidden from library users even though it is not yet centralized as it should be. Constants for ESDL's subsystems and the error messages for the logging facility are all inside a class called `ESDL_CONSTANTS`. This class should be factorized in the future.



Figure 4: Pixel drawing

5 Applications

Besides the two Hello World applications I created three more applications demonstrating ESDL's capabilities. The first one is a demonstration of ESDL's pixel drawing functionality. All I do is iterate through the coordinates of an ESDL surface and color each pixel with an RGB value related to its coordinate. The result is shown in figure 4 on the page before.

The second application demonstrates two things: Fast, double buffered blitting of a surface onto another surface and the polling event loop. The polling event loop is used to query the arrow buttons on the keyboard outside the event loop in order to move the blue square surface around on the background. The possibility to hide the mouse cursor and to toggle between full screen and windowed mode is also shown in this example. The corresponding screenshot can be seen in figure 5.

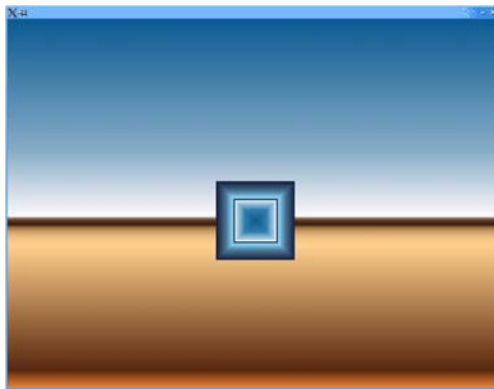


Figure 5: Surface moving on another surface

The last demo is similar to the second one except that the second surface that is blitted on the background has additionally a colorkey that allows to show non rectangular objects. Screenshot⁴ is figure 6 on the next page.

Even though I have not yet created the final presentation of my diploma thesis I am planning to use ESDL to build the presentation as an ESDL application thus becoming self hosting.

⁴The mountain in the background is the world famous Matterhorn in Zermatt, Switzerland and the cow's name is "Lovely" (courtesy of SMP - Swiss Milk Producers).



Figure 6: Lovely flying in Zermatt

6 Extensions

To talk about extensions of a library that is work in progress is difficult. The most important thing for such a library would be that it is finished. Therefore I will describe the state of ESDL in the next part of this document. Nevertheless I have ideas for future extensions of ESDL that I would like to mention here for completeness.

One of the reasons for not using SDL or now ESDL as visualization library in Traffic is that we had no EiffelVision2 binding for it. Since EiffelVision2 is wrapping existing GUI toolkits on the supported platforms, I think it would be appropriate to find a way to integrate SDL into these GUI toolkits. Practically this would mean that a GTK SDL widget is implemented for Linux and that a GDI SDL widget would be implemented for Windows and that Vision2 would then provide access to these two widgets. I know that a project implementing SDL as a GTK [9] widget exists but have not found a corresponding project for the GUI toolkit available on Windows. Please see Reference [25] for details about the GTK SDL widget.

Another extension that might be more straightforward is that the OpenGL binding that SDL provides should be honored and combined with the efforts of those porting OpenGL to Eiffel.

There are also the most direct extensions that SDL needs: These include support for drawing graphical primitives such as lines, polygon lines, splines, bezier and other curves, rectangles, triangles, polygons and ellipses and so on. The possibility to draw all possible graphical primitives anti-aliased and to fill all available polygons with phong shading should not be forgotten.

Part III

State and future of ESDL

This part of the document serves the purpose of documenting how much of SDL's capabilities are already implemented in ESDL. For the subsystems of SDL that are already partly implemented in ESDL I am listing all functions available in the subsystem. For the remaining subsystems that I have not touched yet I will only list the name of the subsystem. For all functions that are already implemented in ESDL I am mentioning which feature from which class in ESDL implements the function. For the functions not yet implemented I will not give any recommendations which existing or new ESDL class should implement it because I believe that the design of ESDL will also be adapted during the progress of the development. Generally I believe that not all available functions need to be included in ESDL and even more importantly not all available structures need to surface in ESDL. An example: If the SDL Rectangle (SDL_Rect) is only used inside the classes building ESDL then the wrapper class of SDL_Rect (SDL_RECT_STRUCT_EXTERNAL) can be used inside ESDL and it will not be necessary to implement an ESDL_RECTANGLE. However this is a decision that can only be taken once the whole library has been built, and the developer can determine whether it is necessary for an ESDL_RECTANGLE to be available as API to the outside world.

General system

ESDL

make SDL_Init: Initializes SDL
 SDL_InitSubSystem: Initialize subsystems (at the moment only the video subsystem is initialized)

quit SDL_Quit: Shut down SDL
 SDL_QuitSubSystem: Shut down a subsystem (this should be taken outside of *quit* once initialisation of separate subsystems is supported)

to-do: SDL_WasInit: Check which subsystems are initialized

Video subsystem

ESDL

video_subsystem_ok SDL_VideoModeOK: Check whether if a particular video mode is supported

initialize_video_surface SDL_SetVideoMode: Set up a video mode with the specified width, height and bits-per-pixel

showcursor SDL_ShowCursor: Toggle whether or not the cursor is shown on the screen

ESDL_SURFACE

redraw SDL_Flip: Swap screen buffers
free SDL_FreeSurface: Free (delete) a SDL_Surface
lock SDL_LockSurface: Lock a surface for direct access
unlock SDL_UnlockSurface: Unlock a previously locked surface
set_colorkey SDL_SetColorKey: Set the color key (transparent pixel) in a blit-table surface and RLE acceleration
blit_surface, blit_surface_part SDL_BlitterSurface: This performs a fast blit from the source surface to the destination surface
make_from_pointer SDL_Surface: Graphical Surface Structure

ESDL_RECT

make SDL_Rect: Define a rectangular area

ESDL_COLOR

make SDL_Color: Format independent color description

ESDL_PALETTE

make SDL_Palette: Color palette for 8-bit pixel formats

ESDL_PIXELFORMAT

make SDL_PixelFormat: Store surface format information

REMAINING

to-do: SDL_MapRGB
SDL_UpdateRect
SDL_VideoInfo
SDL_GetVideoSurface
SDL_GetVideoInfo
SDL_VideoDriverName
SDL_ListModes
SDL_UpdateRects
SDL_SetColors
SDL_SetPalette
SDL_SetGamma
SDL_GetGammaRamp
SDL_SetGammaRamp
SDL_MapRGBA
SDL_GetRGB
SDL_GetRGBA
SDL_CreateRGBSurface
SDL_CreateRGBSurfaceFrom

SDL_LoadBMP
SDL_SaveBMP
SDL_SetAlpha
SDL_SetClipRect
SDL_GetClipRect
SDL_ConvertSurface
SDL_FillRect
SDL_DisplayFormat
SDL_DisplayFormatAlpha
SDL_WarpMouse
SDL_CreateCursor
SDL_FreeCursor
SDL_SetCursor
SDL_GetCursor
SDL_GL_LoadLibrary
SDL_GL_GetProcAddress
SDL_GL_GetAttribute
SDL_GL_SetAttribute
SDL_GL_SwapBuffers
SDL_CreateYUVOverlay
SDL_LockYUVOverlay
SDL_UnlockYUVOverlay
SDL_DisplayYUVOverlay
SDL_FreeYUVOverlay
SDL_GLAttr
SDL_Overlay

Window Management subsystem

ESDL_SURFACE

fullscreen SDL_WM_ToggleFullScreen: Toggle full screen mode (only works on Linux, platform check is implemented)

REMAINING

to-do: SDL_WM_SetCaption
 SDL_WM_GetCaption
 SDL_WM_SetIcon
 SDL_WM_IconifyWindow
 SDL_WM_GrabInput

Events subsystem

ESDL_EVENT_LOOP

make_poll SDL_PollEvent: Poll for currently pending events.

make_wait SDL_WaitEvent: Wait indefinitely for the next available event.

ESDL_EVENT

SDL_Event: General event structure

ESDL_ACTIVE_EVENT

make SDL_ActiveEvent: Application visibility event structure

ESDL_KEYBOARD_EVENT

make SDL_KeyboardEvent: Keyboard event structure

ESDL_MOUSEMOTION_EVENT

make SDL_MouseMotionEvent: Mouse motion event structure

ESDL_MOUSEBUTTON_EVENT

make SDL_MouseButtonEvent: Mouse button event structure

ESDL_JOYSTICK_AXIS_EVENT

make SDL_JoyAxisEvent: Joystick axis motion event structure

ESDL_JOYSTICK_BUTTON_EVENT

make SDL_JoyButtonEvent: Joystick button event structure

ESDL_JOYSTICK_HAT_EVENT

make SDL_JoyHatEvent: Joystick hat position change event structure

ESDL_JOYSTICK_BALL_EVENT

make SDL_JoyBallEvent: Joystick trackball motion event structure

ESDL_RESIZE_EVENT

make SDL_ResizeEvent: Window resize event structure

ESDL_WINDOWMANAGER_EVENT

make SDL_SysWMEvent: Platform-dependent window manager event

ESDL_USER_EVENT

make SDL_UserEvent: A user-defined event type

ESDL_QUIT_EVENT

make SDL_QuitEvent: Quit requested event

ESDL_KEY

make SDLKey: Keysymbol definitions

REMAINING

to-do: SDL_Keysym
 SDL_GetKeyState
 SDL_PumpEvents
 SDL_PeepEvents
 SDL_PushEvent
 SDL_SetEventFilter
 SDL_GetEventFilter
 SDL_EventState
 SDL_GetModState
 SDL_SetModState
 SDL_GetKeyName
 SDL_EnableUNICODE
 SDL_EnableKeyRepeat
 SDL_GetMouseState
 SDL_GetRelativeMouseState
 SDL_GetAppState
 SDL_JoystickEventState

Remaining subsystems

Like mentioned earlier these are the subsystems that are not implemented at all yet.

- Joystick subsystem
- Audio subsystem
- CD-ROM subsystem
- Multi-threaded Programming subsystem
- Time subsystem

Part IV

Project Organization

7 Background material

7.1 Reading list

Bertrand Meyer, “Object-oriented Software Construction”, 2nd Edition, Prentice Hall, 1997 [15].

8 Project management

8.1 Objectives and priorities

There are, as we have seen in the project description part, two categories of objectives, the ones from section 1.2 on page 6 and the concrete ones from section 1.3 on page 7. Therefore I will provide two separate priority listings here. Where necessary I am going to give explanations for my choice of priorities. The priorities range from one to three, where one is the highest and three the lowest. Objectives of priority three are beyond the scope of this diploma thesis.

8.1.1 Design objectives

Objective	Priority
Learn-ability	1
Memorability	1
Efficiency	1
Errors	1
Portability	3 ^a
Compatibility	1
Extendibility	1
Installation	2

^aPortability is inherent because we will be using SDL

Table 1: Design objectives and their priorities

8.1.2 Objectives for the intended results

Objective	Priority
API design	1 ^a
API completeness	2
Demos	1
API Documentation	1
Thesis Documentation	1
API Extensions	3

^aThis can also be regarded as the sum of the priorities from table 1 on the preceding page

Table 2: Priorities of intended results

8.2 Criteria for success

While the ultimate goal of the project is the creation of a complete cross-platform multimedia library for Eiffel, the criteria for success is the quality of the software and the documentation handed in at the end. The result may be a partial implementation of the above objectives without implying any penalty on the success of the project. In this respect the quality of the software is measured according to the following points:

- use of Design by Contract
 - pre- and postconditions
 - class invariants
 - loop variants and invariants
- careful design
 - design patterns
 - extendibility
 - reuse-ability
 - careful abstraction
- core principles of OOSC2 [15]
 - command/query separation
 - simple interfaces
 - uniform access
 - information hiding
- style guidelines (from OOSC2 [15], and from Gobo[2])
- correct and robust code

- readability of the source code

The quality of the documentation is measured according to the following points:

- completeness
- understandability
- usefulness
- accessibility and structure

8.3 Method of work

The technologies involved are:

- Gobo [2] tools such as *geant* and *gexace*
- EWG [12], used to wrap SDL [18]
- Eiffel [20, 14]

ESDL is developed according to the development plan in section 9 on the next page. The library was developed on Linux and tested on Linux and Windows to be able to detect possible portability problems. The documentation is written using the LyX [13] text-setting software and bibliographic references are handled by Bib-T_EX [21].

8.4 Quality management

The quality will be ensured by (in descending order of importance) :

- Close contact with the supervisor, recurring discussions about the design of ESDL
- The documentation, see subsection 8.4.1

8.4.1 Documentation

The documentation was written throughout the whole project. Each of the project steps is reflected as a chapter of this thesis report. ESDL itself is documented in the source code. The whole documentation will be included in the thesis report.

The thesis report was exposed to constant review by the supervisor.

8.4.2 Validation steps

The validation steps are guidelines for the recurrent events that allowed me to validate my progress along the project plan. If it was not possible to have a meeting with my supervisor on a weekly basis, there was a meeting at least at the end of each project step.

- Weekly status report to supervising assistant, either via e-mail or personally
- Intermediate presentations of the work during group meetings
- Final presentation

9 Plan with milestones

9.1 Project steps

In the following I will present a short description and a grouping of the project steps:

- Evaluation

- Evaluate existing code wrappers for Eiffel (EWG [12], jeql [11])

The goal was to be able to determine the fitness of the existing wrapping tools in order to decide whether they would be used in the implementation of ESDL, or whether the wrapping of the underlying C library would be done manually.

- Evaluate existing multimedia libraries (EiffelFURY [5], Eiffel.SDL [22])

The existing multimedia libraries need to be examined to decide whether the design of ESDL should be influenced by their design.

- Usability discussion

- Creation of a demo with the SDL library generated by the C wrapping tools for each subsection of the library

This allowed us to spot eventual flaws in the API.

- Discussion of the found usability flaws

This allowed us to determine what should be done better

- Redesign

- Design of the object oriented library
- Implementation of the object oriented library
- Porting of the demos to the object oriented version of the library

- Usability discussion II

- Second review of the usability of ESDL
- Second adjustment cycle of the usability flaws of ESDL
- Port of the demos to this final version of ESDL

- Application

- Creation of an example application using various capabilities of ESDL

Possibly create the final presentation of the diploma thesis as a multimedia application based on ESDL (self-hosting)

- Extension

- Extend ESDL to support existing multimedia standards⁵

9.2 Deadline

Project start: Monday, 2003-05-26

Project end: Friday, 2003-09-26

Total work time: 17 weeks = 85 days

⁵The multimedia standards to be implemented are subject to discussion: A possible candidate would be the scalable vector graphics [24].

Part V

Appendices

A Map Widget

This appendix describes the design and implementation of a widget used for the introductory programming course given by Prof. Dr. B. Meyer. The course will start in the winter semester 2003/2004. The description of the widget will be split into three parts: The requirements for such a widget will be listed in the first part. At the same time the necessary background information about the context in which the widget will be used is given. In the second part, the design of the widget and the choice of involved libraries is documented, and the third part consists of a tutorial on how to use the widget in another application. The tutorial will be illustrated by code examples from a demo application, see details in [G.1](#) on page 56.

A.1 Requirements

A.1.1 Traffic project

Before exploring the requirements for the Map Widget I think it is necessary to explain the context in which the widget will be used. The Introductory Programming course will follow the teaching method known as Inverted Curriculum [16]. Programming courses thought with this method provide students with an application or a library that is extensible. As the students' programming knowledge evolves during the course they are able to use more and more functionality of the library. Thus a student finds himself in a real world environment of a programmer right from the start and is taught the skills a programmer needs to have. The Introductory Programming course will use a traffic simulation application - therefore the development of this application is called Traffic project.

The specification of the traffic simulation application (hereafter referred to as Traffic) can be summarised by the following points:

1. Traffic can display a city and its traffic infrastructure.
2. The city is visualized by a simplified representation of its geographical features - this means that big rivers and the perimeter of the city are shown.
3. The example city chosen for the initial implementation of Traffic is Paris. However the application should not include any specific information about the city, it should read this information from files.
4. A file describing a city should contain the business model of the city, this model consists of the following items:
 - (a) Transportation links of various kinds (streets, metro lines and bus lines)
 - (b) Monuments and other typical landmarks of the city
 - (c) Traffic participants moving around in the city

5. The visualization of the city should be able to satisfy the following:
 - (a) Zooming in and out of the city map
 - (b) Panning of the map
 - (c) Display of the traffic links
 - (d) Selective display of primitives on the city map (e.g. different levels of detail that allow to show only the metro lines or only the bus lines)
 - (e) Display of landmarks pictures as icons on the map
 - (f) Display of moving vehicles in the city
 - (g) Animation of calculation
6. Inside Traffic the city is represented by a model that allows to
 - (a) specify the number of passengers for a certain traffic line
 - (b) specify a capacity of a certain traffic line
7. Based on this model computations can be performed (e.g. shortest path)
8. Traffic has an API that provides an interface for the students to use Traffic's functionalities (e.g. to calculate a shortest path), to enter simulation parameters (e.g. enter the capacity of a metro line) and to use the display and animation capabilities.

This specification shows that visualizing the model of the city is an integral part of the project. The specification lists many points that require a customised widget to facilitate the display of all the information a city has. If we consider Traffic as an application partitioned into a Model, a View and a Controller (MVC, see [6]), this widget is responsible for the View part of the architecture of Traffic.

A.1.2 Map Widget

After having explored the requirements of Traffic itself I will now list the requirements of the Map Widget which will explain the design of the Map Widget. The Map Widget has to provide support for:

1. Displaying a city and its traffic infrastructure.
2. Displaying a simplified representation of the city's geographical features.
3. Displaying transportation links of various kinds (streets, metro lines and bus lines).
4. Displaying monuments and other typical landmarks.
5. Displaying traffic participants moving around in the city.
6. Zooming in and out of the visualization of the city.
7. Panning of the visualization of the city.

In the next subsection I will address each one of those requirements and describe how I implemented the functionality into the Map Widget. These are the requirements of the client of the Map Widget only. There are of course implications of those requirements that need to be considered in the design as well. I will therefore treat the requirements first and the implications afterwards.

A.2 Design

The GUI (Graphical User Interface) of Traffic is built with EiffelVision2 in order to provide the students with a platform independent application. Therefore EiffelVision2 was the graphical library retained to implement the Map Widget as well. The advantages of this approach in contrast to using ESDL's graphical capabilities are:

- Nothing needs to be changed in the event handling to make Traffic and the Map Widget play together.
- Traffic can be a single windowed application.
- Development of the Map Widget is integrated into the graphical libraries of EiffelStudio's base libraries.

The only disadvantage is that ESDL would provide anti-aliased graphics in contrast to EiffelVision2. This would result in a more appealing rendering of the city map.

A.2.1 Displaying a city and its traffic infrastructure

Generally the traffic infrastructure of a city can be represented graphically as a network of interconnected, colored lines. This is illustrated by the various public transportation maps one finds in big cities. These lines do not need to be bent - they can very well just be angular in places where they change direction. The choice I made to be able to represent a city's traffic infrastructure has fallen on polygonal lines.

A.2.2 Displaying a simplified representation of the city's geographical features

As explained above (see 2 on page 27) the representation of a city's geographical features will be schematic. The geometrical object needed to perform such a task are polygons. The Map Widget has to be able to display polygons that have a color. There is no need for setting an edge thickness, an edge color and a fill color, one color for the edge and the filling is enough and the edge thickness can be left a default value.

A.2.3 Displaying transportation links of various kinds

To distinguish between different kinds of transportation links I chose to provide support for changing the color and the line thickness of a transportation link. Naturally there would be other possibilities too: One could allow dashed lines in contrast to normal ones, for example. I chose the mentioned attributes because I think it would be better to represent different kinds of transportation links by layering their representation on the map. In other words the network of metro lines can be switched on and off, as well as the the network of all bus lines. The rationale behind this is the sheer quantity of those two kinds of transportation links. If I had all metro lines as normal lines and all the bus lines as dashed ones shown at the same time, the navigation would become very tedious for a user. The same goes for the case where I would paint all metro lines green and

all bus lines red. The problem here would be that nobody would be able to say whether a line segment belongs to the metro line number four or to the metro line number seven.

A.2.4 Displaying monuments and other typical landmarks

The straightforward approach here is to allow positioning of small bitmap images on the Map Widget, icons that represent the landmarks. It would be desirable if those icons supported transparency in order not to overlap with too much city information from the background.

A.2.5 Displaying traffic participants moving around in the city

Generally all graphical objects that can be represented on the Map Widget should also be animatable. A state of the art mechanism for animating these objects would be a key frame based animation. The advantages of this animation are that the designer of the animation does not need to worry about the translation speed of the animated objects. I will explain the technical details of this animation technique further down in this subsection.

A.2.6 Zooming and panning of the visualization of the city

Zooming and panning are both manipulation on the coordinates of the graphical objects the city is built of. However zooming and panning do not affect the size and location of the Map Widget inside the Traffic application. This implies that the Map Widget will harbour two coordinate systems to perform these tasks. First the screen coordinate system that is represented by integer values and has the dimension of the widget itself on the screen, and second the coordinate system, that I like to call the city coordinate system, or world coordinate system, that is represented by real values and allows the client to set its dimension. The Map Widget will provide an interface to relate between these two coordinate systems to its clients.

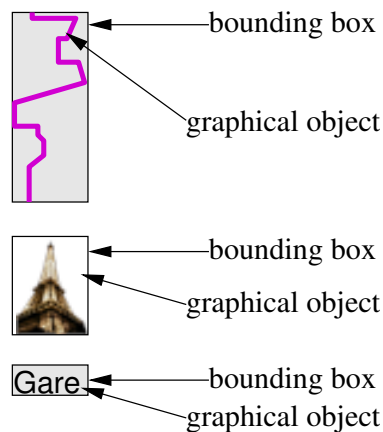


Figure 7: Bounding boxes

A.2.7 Implications resulting from the requirements

An aspect that is not mentioned in the above requirements is the performance of the Map Widget. As always in graphical application the responsiveness of the application is a central usability factor. One concept to achieve better performance is the introduction of bounding boxes for all graphical objects that will be displayed in the Map Widget. Using the bounding boxes of the objects to display, the widget can determine which objects need to be drawn at all and which ones are outside of the widget's perimeter and can be omitted in the processing necessary for the display. That results in a speed up. How the bounding boxes are built is illustrated in figure 7 on the page before. The figure only shows three examples for bounding boxes, but for other graphical objects they would be constructed exactly alike.

```
intersects (other: like Current): BOOLEAN is
  -- Does 'Current' and 'other' overlap?
require
  other_not_void: other /= void
do
  Result := not (
    (right_bound < other.left_bound)
    or (left_bound > other.right_bound)
    or (upper_bound < other.lower_bound)
    or (lower_bound > other.upper_bound))
end
```

Listing 1: Intersection of two rectangles

A simple algorithm shown in listing 1 and illustrated in figure 8 on the next page allows to determine whether an object should be drawn at all.

The second aspect concerning the performance of the widget was the display of bitmaps. Microsoft Windows can only represent bitmap files up to a file size of 48 MB. This operating system limitation is propagated to EiffelVision2 since the library uses the API from Windows to handle bitmap files. This limitation caused problems when zooming in and out. The bitmap file that represents the background of the city was of course much smaller than 48 MB, but when zooming into the city the bitmap was stretched to a bigger size, thus also increasing the file size. The answer to this problem once again was clipping. The approach I took was to cache the original bitmap file (that is smaller than 48 MB) and to calculate then which subsection of that picture needed to be scaled to the size of the Map Widget.

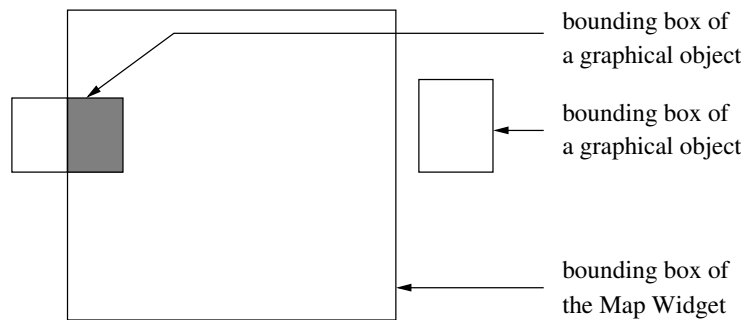


Figure 8: Clipping regions

Another aspect that needed to be considered is connected to the display of the transportation links on the Map Widget. The transportation links are displayed as lines that have a color and a width. Since the Map Widget is able to zoom the representation of the city in and out, the transportation links looked very odd when zooming completely out so that the whole city was shown inside the Map Widget. The width of the lines depicting the transportation links remained constant and this resulted in a representation of the city that looked like a very colorful plate of spaghetti. To overcome this flaw it was evident the the widths of those lines needed to be adapted when zooming.

A.2.8 EV_CANVAS

After talking about the intermediate problems that occurred during development, I would now like to go into the design of the main components of the Map Widget.

EV_CANVAS is the main class of the Map Widget. It inherits from EV_PIXMAP. The drawing of all the graphical primitives is thus implemented using the features from EV_PIXMAP. In figure 9 on the following page, ellipses represent classes - single red arrows represent inheritance relations and a '+' means the class is effected (implemented).

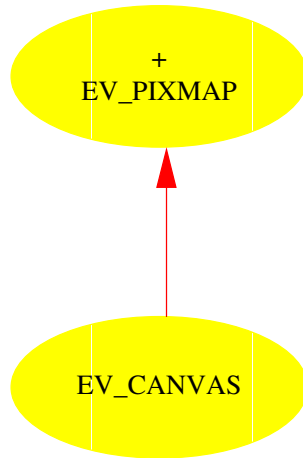


Figure 9: EV_CANVAS

As mentioned earlier, `EV_CANVAS` has two coordinate systems: The screen coordinate system and the world coordinate system. The implementation handles both in a straightforward way. The screen coordinate system results from the creation of `EV_CANVAS`. Because `EV_CANVAS` inherits from `EV_PIXMAP`, the screen coordinate system is equal to the coordinate system provided by `EV_PIXMAP`. The world coordinate system on the other hand is set by default to the range `[0..1]` for both the X- and Y-axis. The user of `EV_CANVAS` should set the world coordinate system to his liking before adding graphical objects to it. (More on how exactly this has to be done is explained in subsection [A.3](#) on page [38](#).) To facilitate things for the user of `EV_CANVAS` I used a different Y-axis orientation for the world coordinate system. There is no ambiguity introduced by this design decision because the user never has to deal with the screen coordinate system. The two coordinate systems and their respective axis orientation are shown in figure [10](#) on the following page.

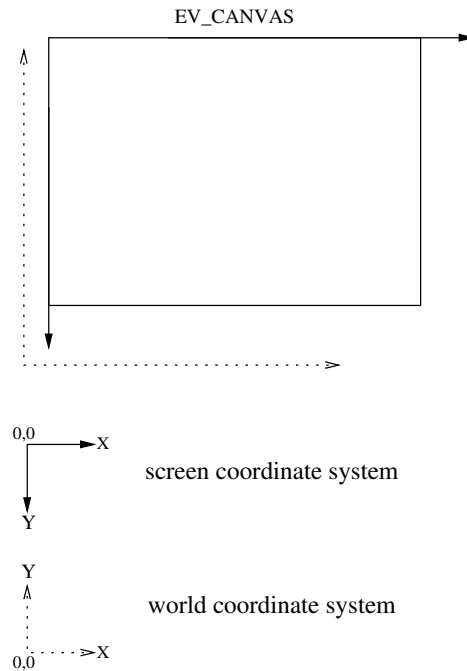


Figure 10: EV_CANVAS's coordinate systems

Furthermore EV_CANVAS maintains a list of the graphical objects the user wants to display on the Map Widget. Three features allow a fine grained control of which graphical objects should be drawn:

- draw_item
Used to draw one specific graphical object.
- draw_all_items
Used to draw all graphical objects in a list.
- refresh
Used to redraw all items in EV_CANVAS's list of graphical objects.

Like mentioned above the real drawing actions are only performed if the graphical object is inside the Map Widget's perimeter.

EV_REAL.COORDINATE

Having mentioned the two coordinate systems again, it is time to introduce the class EV_REAL.COORDINATE. The class represents, as its name already suggests, a two dimensional coordinate in a *real* value based coordinate system. Besides the creation, access and output feature clauses there is a calculation feature clause that comprises the following features: *right_by*, *left_by*, *up_by*, *down_by*, *add*, *subtract* and *multiply*, to manipulate the coordinates.

EV_REAL_RECTANGLE

Finally there are not only coordinates but there is also the building block behind the bounding boxes and the world coordinate system: This is the class `EV_REAL_RECTANGLE`. Again, as the name suggests, it is a rectangle constructed with `EV_REAL_COORDINATES`.

This brings me to the discussion of the family of classes that inherit from `DRAWABLE_OBJECT` and constitute the graphical objects that the Map Widget can display.

A.2.9 DRAWABLE_OBJECT

`DRAWABLE_OBJECT`s are the graphical objects drawn on the Map Widget. They all have a bounding box that is an `EV_REAL_RECTANGLE`, they are drawn by calling the *draw-object* feature, and they store a reference to the canvas on which they are to be drawn. Furthermore they provide facilities to convert screen coordinates to world coordinates and vice versa. In the following paragraphs, I am going to describe each of the drawable objects that are available for `EV_CANVAS` and I will try to detail the ideas behind the choice for these objects. Of course most of the drawable objects exist because they are required by Traffic's specification.

DRAWABLE_POLYLINE

The `DRAWABLE_POLYLINE` will be used to display the transportation links on the Map Widget as lines that have a width and a color. This class also handles the line width scaling of the transportation links of the city being zoomed in or zoomed out.

DRAWABLE_POLYGON

This class is used to display a simplified representation of the city's geographical features. It can be used to draw a river floating through a city for example. The polygons created using `DRAWABLE_POLYGON` have only one color, the one inherited from `DRAWABLE_OBJECT`.

DRAWABLE_PIXMAP

Just like `DRAWABLE_POLYGON`, the `DRAWABLE_PIXMAP` class is intended to be used as a possibility to display a background of a city. In this case however the whole background is a bitmap image. The bitmap caching that I have explained in subsection [A.2.7](#) on page [31](#) is implemented in this class. Therefore it has also a private feature clause "calculations" that contains features needed to calculate the clipping area of the bitmap file that will be stretched to the Map Widget's size while zooming.

DRAWABLE_ICON

The purpose of `DRAWABLE_ICON` is to enable displaying iconified representations of landmarks on the city map. These icons can be positioned according to their upper left corner.

DRAWABLE_TEXT

DRAWABLE_TEXT enables labelling stops on the transportation links of a city. Of course other things that are displayed on the Map Widget can also be labelled like that. Alike DRAWABLE_ICON, the text labels are positioned according to their upper left corner.

DRAWABLE_CIRCLE

This class can be used to draw an empty or a filled circle on the Map Widget. There is only one color for the edge and the fill of the circle. Note that a circle will be scaling when zooming in or out the map.

DRAWABLE_SPOT

Unlike the circle mentioned above, the DRAWABLE_SPOT does not scale when zooming the Map Widget. It can therefore be used to represent stations or stops along the transportations links on the city's map.

A.2.10 ANIMATABLE

Now I am going to describe the animation framework of EV_CANVAS. Then I will discuss the classes that constitute the animation framework for the Map Widget.

The animation support for EV_CANVAS is based on key frame animation. This means that for any graphical object that one wants to animate, key frames have to be provided in order to specify where the object has to be at what time. The time is taken from a time line that can be stopped, rewound and resumed, the movement of the animated object is linear between two key frames and is interpolated by the animation framework.

Now a more technical description about how to add animations to the EV_CANVAS:

First one needs to create a time line. The time line creation takes a Δt that indicates the time delay between two increments of the time. Time is measured in *integer* values starting from 0 and increasing. After creating the time line one can create an animatable object, for example an ANIMATABLE_SPOT. For all DRAWABLE_OBJECTs it will be possible to add animation support. Once the object to be animated is created one needs to create an ARRAYED_LIST [KEYFRAME]. These have to be added to the animatable object, before the animatable object is added to the time line and to the canvas. From that point on the position of the animatable object is determined by the time of the time line.

The class ANIMATABLE thus has features to set a time line, to set key frames and most importantly a feature called *position* that interpolates the location of an ANIMATABLE on the Map Widget with the information provided by the key frames and the time line.

TIMELINE

As mentioned above the time line is created with a Δt . However, the most important feature of the time line is the feature *time* that reports the current

time on the time line. Other than that the features *play* and *reset* serve to start the increasing of the time and to stop and reset it again.

KEYFRAME

The KEYFRAME is built from a time (measured in *integers*) and a location (given as an EV_REAL_COORDINATE, see subsection [A.2.8](#) on page 34). It is the main building block for all the animations.

A.3 Tutorial and example application

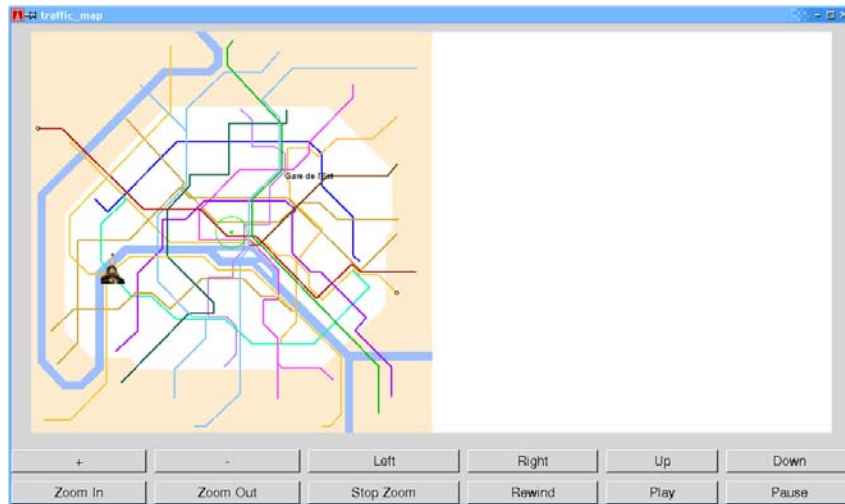


Figure 11: Map Widget example application

In this tutorial I am going to explain how to build an application that uses the Map Widget to display the metro line network of Paris. At the end of this tutorial we will have constructed the application shown in figure 11.

The files required for this tutorial are available on the website mentioned in G.1 on page 56. I suggest that you download all of the tutorial files and use them to complete the tutorial. If you are familiar with building EiffelVision2 applications I suggest you skip step 1 in A.3.1 on the next page and start directly with Step 2 of this tutorial in A.3.2 on page 40.

A.3.1 Step 1, creating a window

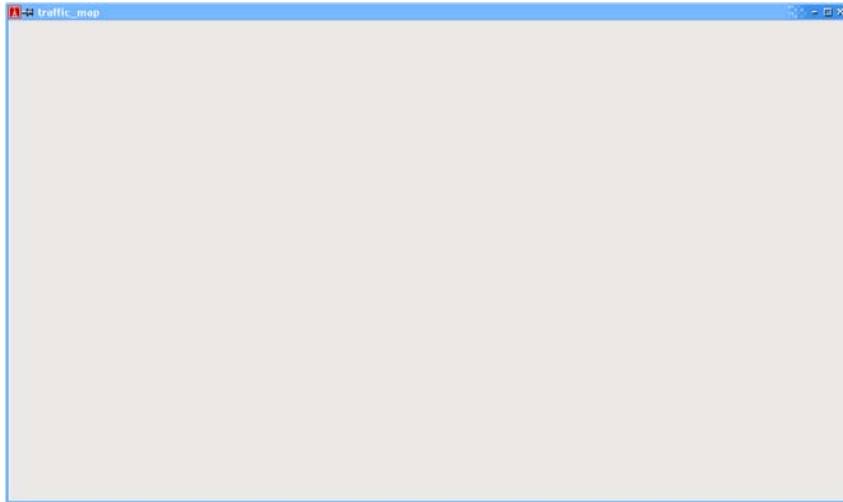


Figure 12: The window after step 1

- Download the files for this step of the tutorial (URL is shown in [G.1](#) on page [56](#)).
- After extracting the files to a local directory you can open and compile the project with EiffelStudio. Use the Ace file corresponding to your platform to do so. Start the application after the compilation has finished. You should see a window similar to the one shown in figure [12](#).
- Now take a look at the clusters view in EiffelStudio. All classes we are going to use in this tutorial are located in the *root_cluster*. Inside the *root_cluster* you can see a cluster called *canvas*. All classes providing the functionality of the Map Widget are located inside *canvas*. We will use these classes to build the application, but we are not going to manipulate them.
- Back in the *root_cluster* is the class MAIN_WINDOW. To demonstrate the Map Widget we are going to add code to this class. Open it in the editor of EiffelStudio.

A.3.2 Step 2, placing the canvas and setting the world coordinate system



Figure 13: The window after step 2

Now we will add the actual Map Widget to this empty window and display it. Since we use the Map Widget to display a world that has another coordinate system than our screen we will also need to set this world coordinate system.

```
feature {NONE} -- Map widget

  visible_area: EV_REAL_RECTANGLE
    -- The world coordinate system

  canvas: EV_CANVAS
    -- The Map Widget
```

Listing 2: Declaration of *visible_area* and *canvas*

- Create a feature clause *Map widget*, that is exported to NONE. Declare the *visible_area* and the *canvas* as shown in listing 2. This feature clause is exported to NONE because we do not need to access the *visible_area* and the *canvas* from the outside.
- Navigate to the feature clause *Window population* and select the feature *build_main_container*. To navigate from one feature clause to another you can use the **Features** tool in EiffelStudio. If it is not visible, you can enable it with **View** ▷ **Tools** ▷ **Features** in EiffelStudio. You will have to create the canvas here.

```
feature {NONE} -- Window population

build_main_container is
  -- Create and populate 'main_container'.
  require
    main_container_not_yet_created: main_container = Void
  do
    create main_container
    main_container.set_padding (10)
    create canvas.make
  end
```

Listing 3: Creation of the canvas

- To set the world coordinate system you have to create one that satisfies your needs and then you will have to call the feature *set_visible_area* of the *canvas*. In listing 4 I am creating a world coordinate system of 150 by 150 units.
- Of course you also need to set the screen dimension of the widget. This is done by calling the feature *set_size* with its respective arguments. I chose 500 times 500 pixels to start with. Once you have completed this, you can compile and run the application to obtain the window shown in figure 13 on the page before.

```
visible_area:=
create {EV_REAL_RECTANGLE}.make_from_reals (0, 0, 150.0, 150.0)
canvas.set_visible_area (visible_area)
canvas.set_size (500, 500)
```

Listing 4: Set the world coordinate system

A.3.3 Step 3, adding polygons, lines, icons, spots and circles

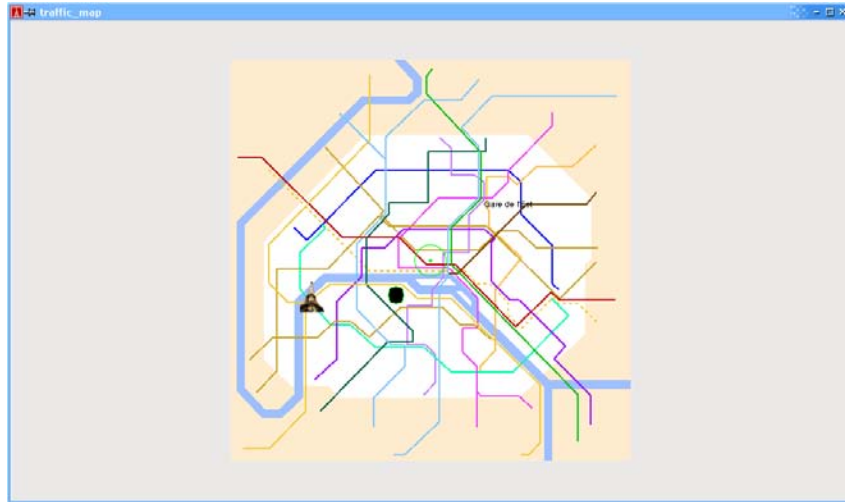


Figure 14: The application with all graphical objects

As a starting point for this application I used the map of Paris's metro system (you can find a picture of Paris metro system on <http://www.citefutee.com/orienter/plans.php>). In this step we are going to add all possible geometrical objects to the Map widget.

- First we will construct the background of Paris. Add a polygon to the Map Widget. The polygon is constructed by specifying its corner points. After creating the polygon you have to add it to the list of objects that is drawn on the map widget. In the code that accompanies this step of the tutorial you will find the features that take care of redrawing the map. I am not explaining how they work any further. It is enough if you know that you have to register or add an object that you would like to draw to the list of objects that will be drawn: *object_list*.

```
create background.make (coordinate_array)
background.set_color (a_color)
object_list.extend (background)
```

Listing 5: Create a polygon

- Secondly try to add a line, try colouring it and setting its thickness.
- When you are comfortable adding lines and registering them into the list of objects that will be drawn on the Map Widget, you can go further. Add icons, spots, rounded rectangles and circles.

After compiling you should see the window displayed in figure 14 on the preceding page. It features all the graphical objects but no buttons to manipulate the display of the map. In the next step we will add buttons for zooming and panning.

```
feature {NONE} -- Metro lines

    line_1: DRAWABLE_POLYLINE
        -- Metro line 1

feature {NONE} -- Background

    background: DRAWABLE_POLYGON
        -- Background polygon

feature {NONE} -- Other graphical objects

    label: DRAWABLE_TEXT
        -- A text that will be displayed on the map

    spot: DRAWABLE_SPOT
        -- A non-scaling circle

    circle: DRAWABLE_CIRCLE
        -- A scaling circle

    station: DRAWABLE_ROUNDED_RECTANGLE
        -- A rounded rectangle used to depict stations
```

Listing 6: Declare some graphical objects

A.3.4 Step 4, adding zooming and panning

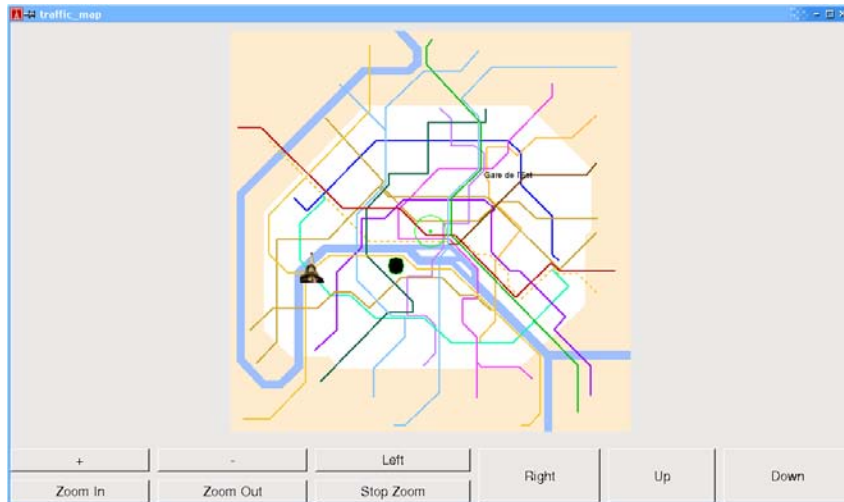


Figure 15: The application with zooming support

Zooming and panning are performed by manipulating the world coordinate system of the Map Widget. In the source code that accompanies this step of the tutorial you will find the code that displays the buttons for zooming and panning as well as the code that performs the changes of the world coordinate system. To explore the possibilities I suggest that you start looking at the features under the feature clause *Map Widget panning actions* in the class MAIN_WINDOW. These features are responsible for panning the map. They are called when clicking the corresponding buttons.

Once you feel comfortable with the panning mechanism you can start looking into the feature clause *Map Widget zooming actions* that groups the features used for zooming in and out together. Manipulating these features is a little bit more complicated because they respect maximal zooming factors.

After compilation you should see the application window shown in figure 15.

A.3.5 Step 5, adding animations

To add animations you will have to draw an ANIMATABLE graphical object on the map. Every graphical object that can be drawn on the Map Widget can be turned into an animatable graphical object. To make things easy I have included the class ANIMATABLE_SPOT in the files that come with this tutorial. As I have explained earlier in the subsection A.2.10 on page 36, you have to create a time line. The time line creation takes a Δt that indicates the time delay between two increments of the time. Time is measured in *integer* values starting from 0 and increasing. Once the object to be animated and the time line are created you will create an ARRAYED_LIST [KEYFRAME]. Add these to the animatable object, before you add the animatable object to the time line and to the *object_list*.

After setting the objects to be animated up all that you need to do in order to animate them is to start the time on the time line. The feature clause *Map Widget animation actions* contains the features which do that. Again they are bound to the buttons with the agent mechanism.

A.3.6 Step 6, changing the canvas dimension

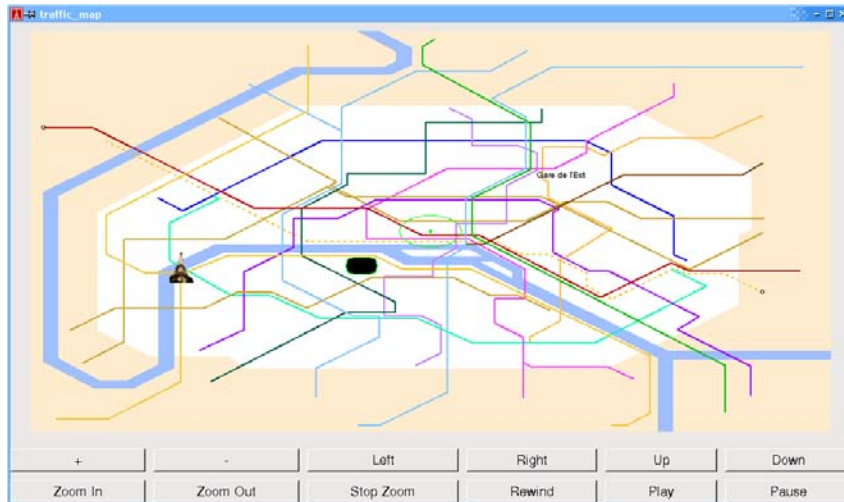


Figure 16: Distorted world coordinates

Imagine the situation where you would like the Map Widget to change its size - for example when resizing the application window. Since there are two coordinate systems involved it is very important that the aspect ratio of the two are not affected. In this very small step of the tutorial I am going to explain how to change the Map Widget's dimension on the screen without affecting the display of the map. This step leads us to the end of this tutorial.

At the moment the screen coordinate system is 500 by 500 pixels big and the world coordinate system measures *Window_x_size* by *Window_y_size* (constants that can be found in `INTERFACE_NAMES`). To adjust the dimension of the widget go to the feature *build_main_container* and find the line where we set the Map Widgets screen dimension.

```
canvas.set_size (500, 500)
```

Change it to read:

```
canvas.set_size (1000, 500)
```

Compile the application and execute it. You will note that there is a distortion of the graphical objects displayed (see figure 16). This distortion is because we didn't adjust the world coordinate system. Do so now by changing the value of *Window_x_size* from 1268.0 to 2536.0 in the class `INTERFACE_NAMES`. If you compile and execute the application now it should look exactly like figure 11 on page 38.

If you are planning to use the Map Widget in your application you can download the library from the website mentioned in G.1 on page 56. I would appreciate getting your feedback if you use the Map Widget - see the contact details on the first page of this document.

B HELLO_WORLD_SDL

indexing

```
description: "[
    HELLO_WORLD_SDL. SDL application
    that is based on the abstraction EWG
    creates of SDL. Displays an image for
    a while.
]"
author: "Andreas Leitner, Till G. Bay and others"
license: "Eiffel Forum License v2 (see forum.txt)"
```

class

```
HELLO_WORLD_SDL
```

inherit

```
KL_SHARED_EXCEPTIONS
  export
    {NONE} all
  end

SDL_FUNCTIONS_EXTERNAL
  export
    {NONE} all
  end

SDL_VIDEO_FUNCTIONS_EXTERNAL
  export
    {NONE} all
  end

SDL_IMAGE_FUNCTIONS_EXTERNAL
  export
    {NONE} all
  end

SDL_TIMER_FUNCTIONS_EXTERNAL
  export
    {NONE} all
  end

STDLIB_FUNCTIONS_EXTERNAL
  export
```

```
        {NONE} all
    end

create

    make

feature -- Initialization

    make is
        -- Create an SDL window and display
        -- an image in it. Wait a bit and quit.
    local
        i: INTEGER
        p: POINTER
    do
        i := sdl_init_external (sdl_init_video)
        if i < 0 then
            print ("Unable to init SDL%N")
            exceptions.die (1)
        end
        i := atexit_external (get_sdl_quit_pointer_external)
        if i /= 0 then
            print ("Unable to register SDL_Quit%N")
            exceptions.die (1)
        end
        p := sdl_set_video_mode_external (283, 274, 16, sdl_swsurface)
        if p = default_pointer then
            print ("Unable to set 100x100 video%N")
            exceptions.die (1)
        end
        create display.make_unshared (p)
        create c_string.make_unshared_from_string (image_file_name)
        p := img_load_external (c_string.item)
        if p = default_pointer then
            print ("Couldn't load " + image_file_name + "%N")
            exceptions.die (1)
        end
        end
        create image.make_shared (p)
        i := sdl_upper_blit_external (image.item, default_pointer, display.item,
            default_pointer)
        i := sdl_flip_external (display.item)
        sdl_delay_external (10000)
        sdl_free_surface_external (image.item)
    end
end
```

```
display: SDL_SURFACE_STRUCT
    -- Struct wrapper for an SDL display handle

image: SDL_SURFACE_STRUCT
    -- Struct wrapper for an SDL image handle

c_string: WG_ZERO_TERMINATED_STRING
    -- C string wrapper

feature {NONE} -- macro constants (not yet generated)

Sdl_init_video: INTEGER is 32
    -- Macro constant for video initialization

Sdl_swsurface: INTEGER is 0
    -- Macro constant for video RAM

feature {NONE} -- Implementation

Image_file_name: STRING is "hello.world.bmp"
    -- Name of image we want to display

end
```

C HELLO_WORLD_ESDL

indexing

```
description: "[
    HELLO_WORLD_ESDL. ESDL application.
    Displays an image until user closes the
    application.
]"
author: "Till G. Bay, tillbay@student.ethz.ch"
```

class

```
HELLO_WORLD_ESDL
```

inherit

```
ESDL_CONSTANTS
export
  {NONE} all
end
```

create

```
make
```

feature -- Initialization

```
make is
  -- Create the main application.
local
  esdl: ESDL
do
  create esdl.make
  esdl.set_video_surface_width (width)
  esdl.set_video_surface_height (height)
  esdl.set_video_bpp (resolution)
  esdl.initialize_video_surface
  screen := esdl.video_surface
  create image.make_from_image (image_file_name)
  screen.blit_surface (image, 0, 0)
  screen.redraw
  create event_loop.make_wait
  event_loop.quit_event.subscribe (agent handle_quit_event (?))
  event_loop.dispatch
  image.free
```

```
        screen.free
        esdl.quit
    end

feature -- Event handling

    handle_quit_event (a_quit_event: ESDL_QUIT_EVENT) is
        -- Handle 'a_quit_event'.
    require
        a_quit_event_not_void: a_quit_event /= Void
    do
        event_loop.stop
    end

feature {NONE} -- Implementation

    screen: ESDL_SURFACE
        -- The screen

    image: ESDL_SURFACE
        -- The image

    event_loop: ESDL_EVENT_LOOP
        -- The event loop

    Width: INTEGER is 283
        -- The width of the surface

    Height: INTEGER is 274
        -- The height of the surface

    Resolution: INTEGER is 24
        -- The resolution of the surface

    Image_file_name: STRING is "hello_world.bmp"
        -- Name of image we want to display

end
```

D Other applications

D.1 Coordinate application

The coordinate application was used to extract coordinates from a plan of the metro lines of Paris. The idea was that it is easier to extract the coordinates of the metro lines with an application that will take care of aligning two subsequent coordinates according to their x coordinate value or according to their y coordinate value. It is important to have all the horizontal and vertical parts of a metro line aligned, because the Vision2 library that we use in the Map Widget to visualize the metro lines does not support anti-aliasing and therefore it would look very awkward to have almost horizontal or almost vertical lines. As you can see in figure 17 there are buttons to align the coordinates and also a button to remove the last coordinate that has been added to the list of coordinates. The line that is recorded is shown in light green. After having selected all points that are relevant for one metro line, the gathered data can be put out to a text file. This application was built for saving a lot of time for anybody charged with the task of collecting geographical information of a city for the traffic project.

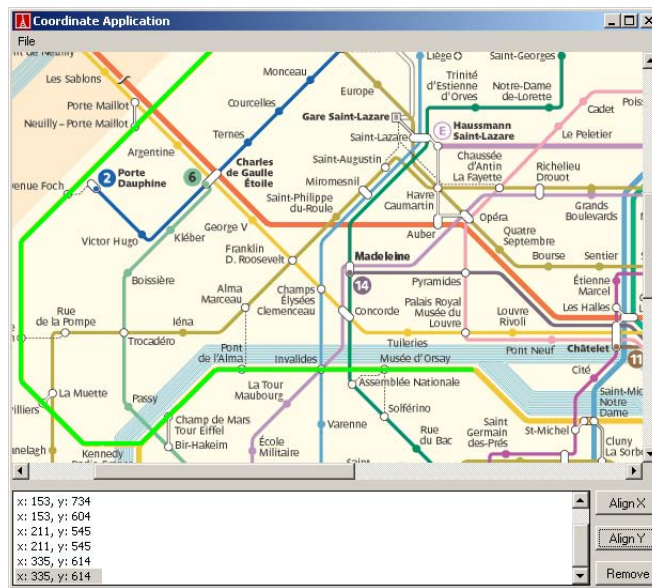


Figure 17: Coordinate application

D.2 Fractal application

During the time of my diploma thesis we also hosted a show for our future students. Future students from schools all over Switzerland visited ETH during two days. To introduce the Chair of Software Engineering of ETH we presented a fractal application. The application is shown in figure 18. As you can see there are three input fields: L-System, Angle and Iterations. Using these three fields, students were able to enter L-System fractals by specifying a base pattern, that the application can draw recursively. The field “L-System” takes the base fractal pattern, the field “Angle” takes an angle for the turns that are specified with “+” or “-” and “Iterations” is used to tell the application how many replacing cycles should be performed with the grammar. The screenshot shows the famous triangle of Sierpinski.

Legend

F: forward

+: turn right by angle

-: turn left by angle

=: definition

[]: go back to the starting point before []

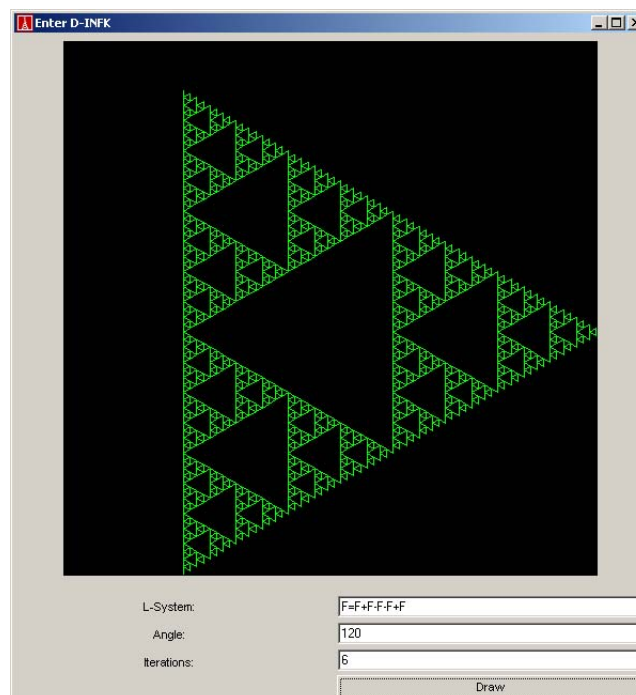


Figure 18: Fractal application

E Acronyms

ACE	Assembly of Classes in Eiffel
ANSI	American National Standards Institute
API	Application Program Interface - a set of routines, protocols and tools for building software applications
ESDL	Eiffel Simple DirectMedia Layer - the name of this diploma thesis
EWG	Eiffel Wrapper Generator
GUI	Graphical User Interface
GTK	The Gimp Tool Kit
ISE	Interactive Software Engineering - creators of EiffelStudio
ISO	International Organization for Standardization
Jegl	Jan's Eiffel game library - one of the SDL wrappers
MB	Mega Byte
MCI	Media Control Interface
MVC	Model View Controller
SDL	Simple DirectMedia Layer

F Versions of libraries used

EiffelVision2 5.3

EWG 0.5.1, May 22, 2003

jegl 0.1.1, July 20, 2000

SDL 1.2.5

GOBO > 3.1, CVS from June 4, 2003

G Additional resources

In this appendix I am listing additional material that accompanies this diploma thesis - most of this additional material will be accessible via internet and the reader is invited to explore these resources.

G.1 Map Widget tutorial files

Map Widget Library http://n.ethz.ch/student/bayt/download/map_widget_library.zip

Tutorial Step 1 http://n.ethz.ch/student/bayt/download/step_1.zip

Tutorial Step 2 http://n.ethz.ch/student/bayt/download/step_2.zip

Tutorial Step 3 http://n.ethz.ch/student/bayt/download/step_3.zip

Tutorial Step 4 http://n.ethz.ch/student/bayt/download/step_4.zip

Tutorial Step 5 http://n.ethz.ch/student/bayt/download/step_5.zip

Tutorial Step 6 http://n.ethz.ch/student/bayt/download/step_6.zip

References

- [1] Arnout K. Arslan V., Nienaltowski P. Event library: an object-oriented library for event-driven design.
<http://se.inf.ethz.ch/people/arslan/index.html>.
- [2] Eric Bezault. GOBO, retrieved May 2003.
www.gobosoft.com.
- [3] EiffelOpenGL, retrieved May 2003.
<http://eifogl.sourceforge.net>.
- [4] EiffelStruggle. EiffelStruggle, retrieved May 2003.
eiffel-nice.org/eiffelstruggle/2003/judging.html.
- [5] EiffelZone. EiffelFURY, retrieved May 2003.
<http://eiffelzone.sourceforge.net/eiffelfury/eiffelfury.html>.
- [6] Erich Gamma et Al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [7] GLU. OpenGL Utility Library, retrieved May 2003.
<ftp://ftp.sgi.com/opengl/doc/opengl1.2/glu1.3.pdf>.
- [8] GLUT. OpenGL Utility Toolkit, retrieved May 2003.
192.48.159.181/developers/documentation/glut/spec3/spec3.html.
- [9] GTK+. The gimp toolkit, retrieved september 2003.
www.gtk.org.
- [10] American National Standards Institute. American National Standard for Information Systems#173;Programming Language C, 1989. X3.159-1989.
- [11] Jegl. Jan's Eiffel game library, retrieved May 2003.
<http://jegl.sourceforge.net>.
- [12] Andreas Leitner. EWG - Eiffel Wrapper Generator, retrieved May 2003.
<http://ewg.sourceforge.net>.
- [13] LyX. The Document Processor, retrieved May 2003.
www.lyx.org.
- [14] Bertrand Meyer. *Eiffel: The Language*. Prentice Hall PTR, 1992.
- [15] Bertrand Meyer. *Object-Oriented Software Construction, 2nd edition*. Prentice Hall PTR, 1997.
- [16] Bertrand Meyer. The outside-in method of teaching introductory programming, 2003.
http://sourceforge.net/project/showfiles.php?group_id=66388.
- [17] OpenGL, retrieved May 2003.
<http://www.opengl.org>.
- [18] SDL. Simple DirectMedia Layer, retrieved May 2003.
<http://www.libsdl.org>.

- [19] SmartEiffel, retrieved May 2003.
<http://smarteiffel.loria.fr>.
- [20] Eiffel Software. Eiffel, retrieved May 2003.
<http://www.eiffel.com>.
- [21] Bib TeX, retrieved May 2003.
www.ecst.csuchico.edu/~jacobsd/bib/formats/bibtex.html.
- [22] Steve Thompson. Eiffel_SDL (Colorado Eiffel User Group), 2003.
http://sourceforge.net/project/showfiles.php?group_id=66388.
- [23] Steve Thompson. Colorado eiffel user group, retrieved August 2003.
<http://sourceforge.net/projects/ceug/>.
- [24] W3C. Scalable Vector Graphics (SVG), retrieved May 2003.
<http://www.w3.org/TR/SVG>.
- [25] GTK SDL Widget, retrieved September 2003.
<http://gtkSDL.sourceforge.net>.