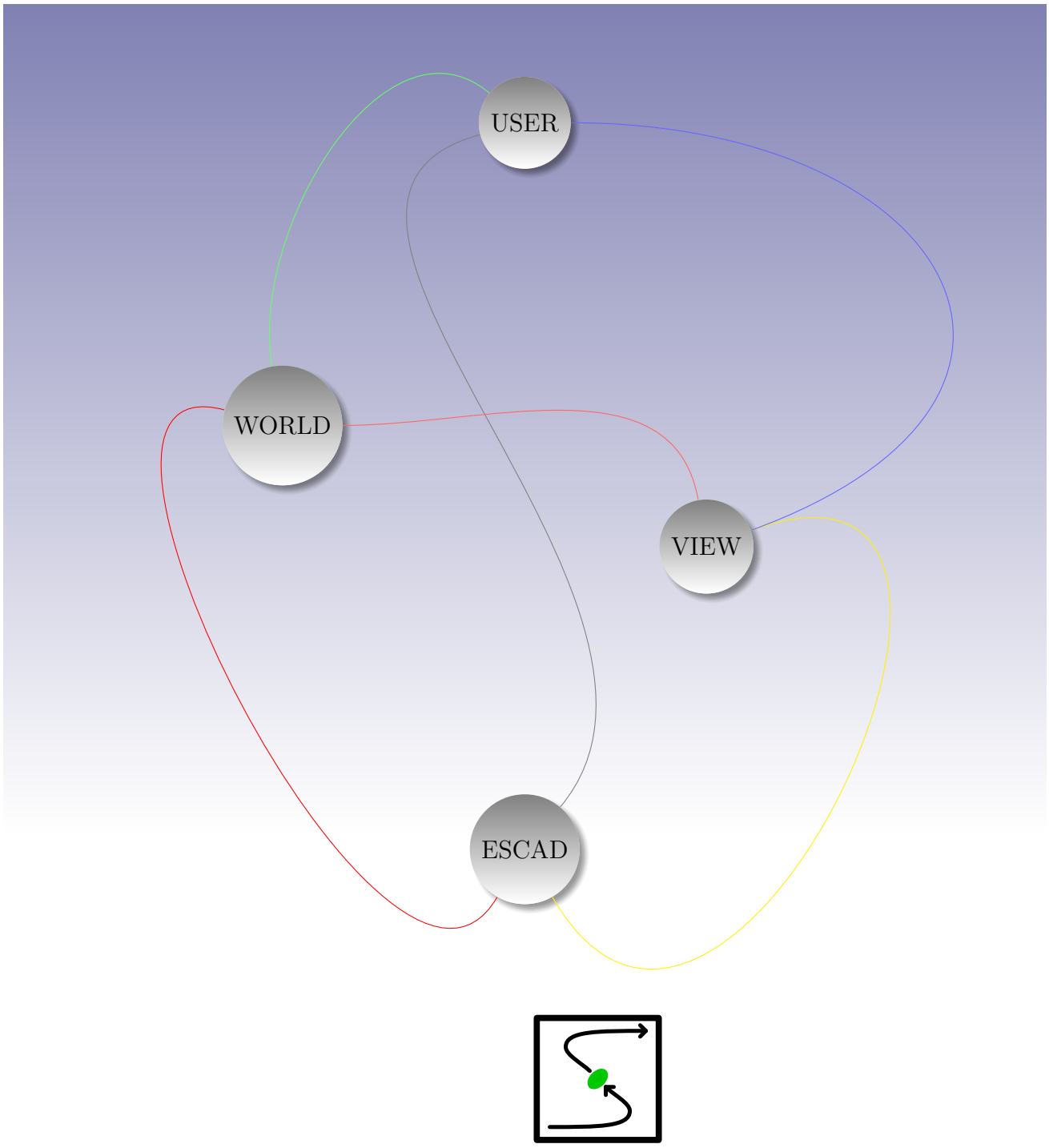
Manual for ESCAD version 0.1.0 - 23rd October 2023

by: Markus Kollmar



Alles hängt mit allem zusammen. Wir haben nur nicht das ganze Bild...

Contents

1 Introduction

You have choosen interest in a friendly libre software. This software is aimed to be a workhorse for many tasks which are useful to do with the help of a graph structure. However to be honest this escad version 0.1.0 shows that there is still much to develop. So some things may not work as expected or there are silly mistakes around. But i think it is better to regularly update.

Documentation lives like the code! I recommend - and think it is very useful - to **read this manual**. Because you get a feeling about escad and its terminology. And unlike some manuals, escad tries a documentation-driven development process, which will be explained in another section. This makes this documentation a up to date view of the current escad and shows you what can be awaited from escad and what not.

1.1 Motivation example "room planing"

Maybe it is easier to get the idea of escad by describing a example task we want do. If you feel sympathic which this described process, then escad may be for you!

Imagine you plan to fetch an cabinet for your living room or a machine for your workshop. Before you need to figure out the best position for the cabinet or machine and to make sure there is enough space for it. For your wife or your boss you have to create a small PDF document with some solutions so that you could together make a decision if or where to place the cabinet or the machine. See figure 1.1 with a sketch of the room. Without special tools you may first

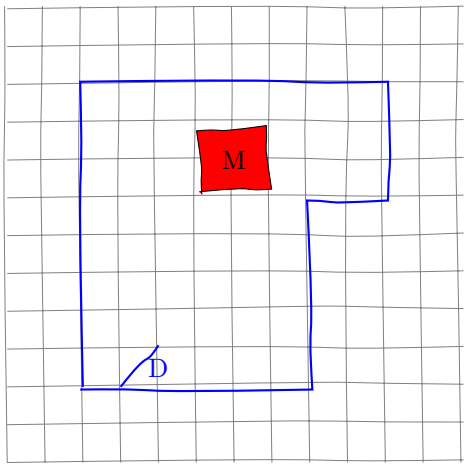


Figure 1.1: Sketch of the room planing example.

draw a two-dimensional drawing of the situation and figure out some solutions. You use a drawing programm and generate some pictures. This pictures you insert in a office-document-program and you enter some description. At the end you generate a PDF-document.

However your wife or your boss say you did a great job, but they are concerned about the third-dimension. They want to see if and how it fits into the three-dimensional space of the living room or the workshop. You now maybe have to reenter the data in a other program which is able

to handle three dimensional data. Maybe you first have to learn it, but even then you have to generate new images for your office-document-program.

Now you are finished and all are happy. But consider if you have to fetch more furnitures or machines and your wife or your boss need a table filled with properties of furniture or machines to get an overview and a total amount of price. In this case you have to reedit your office-docment and to make sure you have consistent data like unique name for the furnitures or machines to properly name them in the pictures and the table.

Imagine instead of the described situation you would have instead following situation:

- You have **one** program where you can work in.
- You enter or link the needed **data one single time**.
- At **new tasks you can stay in this one program** which you are familiar with. To generate a new PDF you not need to switch to different program to first generate new images.
- Also if you want change the property of a furniture or machine (e. g. the height) you not want to do it several times in your document (e. g. in the 3d-model and the property table). You need in general a way to **symbolical describe** your problem and then to have tools which generate a output of it (in our case a PDF).
- If you need a 3D-data of one of your furniture or machine later, you simply want generate it without enter already given data or you want to import data in some 3D-format. If this functions would be not integrated yet in the tool, you may want (or let) write it yourself and thus **expand the tool** to your needs.
- The data is stored in open format in **unicode text**.

Imagine there would be such a program with such aims! Maybe you have found it here with **ESCAD**! ;-)

1.2 Philosophy

History tells us about separation. Separation between different professions was and is common. People have different interests and different knowledge. So this seems natural. Is this a problem? No and yes. Nowadays science goes into a direction where **interdisciplinarity** seems more important. The bounds of our disciplines are drawn by human but it not need to reflect the reality in nature. This seems not a big problem in many cases. But when it comes to documentation, multi domain or knowledge transfer tasks, this can be a big problem. The tools often do not interact with each other, and if so they often feel not integrated well. This problem increases when different manufacturers have tools which interact not or very bad. Furthermore real problems are mostly system-problems or are increasingly seen as such. Systems are combined of different disciplines. But do the (software) tools support this fact? Some may but many do not. Escad does! Escad allows theoretically to model various disciplines in one software tool! Once you have done this you can use this model to generate some output, which mostly may be documents of various type.

Living in todays world is getting more and more complex. There are laws made by human which you have to obey. Additionally mother nature has their ever-lasting rules, overwriting all human made rules. These we should research and obey to keep the environment healthy for us and future mankind. Thus people may have the need to get information about this **complex** system and tools to work with the data. Escad is not good in some things. In fact it is really

not yet a good graph analysis tool (however one could update that functionality). Escad wants to be a worker which allows combining different domains in one model to get something out of your semantic-graph-model. So you avoid switching different tools, create LaTeX , PDF, 3D models, music or other documents, which would be boring to create by hand. You can additionally also use the **scripting** facility of escad.

In short these are the main goals of escad:

- Easy modeling of different domains as graphs.
- Extend the graph whenever needed by another domain.
- Provide a extensions mechanism to get tools for different tasks and for different domains.
- Provide a web-browser based graphical user interface.
- Provide common-lisp programming language for people with scripting needs.
- Integrated help and documentation.
- Free, open source software model.

1.3 Documentation-driven development

Documentation-driven development means here, at first it is defined in this manual what we want to develop. Then the feature will be implemented according this documentation. This is an advantage for the user, since he can see what a feature will look like, and documentation is ahead and not behind the actual implementation. The specified features in the manual, align to the mentioned version of escad. If some feature will be in a later version, this should be mentioned by a version notice and the text is in orange color like this four words . Features which are in the software but not documented here, should not be used or rely on, because they may change or are not ready for production yet.

1.4 State and screenshoots

Currently escad is in (wild) development 😀. This means there are many ideas and concepts which are developed. Some are just a quick hack to show what is possible and some are more detailed. However the current repository code may not work in partly or as whole at every time, only a official release is intended to work. Basically you can use escad in two ways:

- Command-line entering common-lisp code, here in the editor emacs, see figure 1.2.
- Graphical user interface via web browser, see figure 1.3.

Escad use open technologies and provide a wide variety of different domain tools. And if there is (yet) not the thing you need, you can customize escad with common-lisp code. Escad wants to be kind and helpful to the user. **Your help is really welcome**: if you want maintain this manual, increase escad features, write expansions, create logo, manage homepage or just send some feedback - all things are kind to start a escad-community.

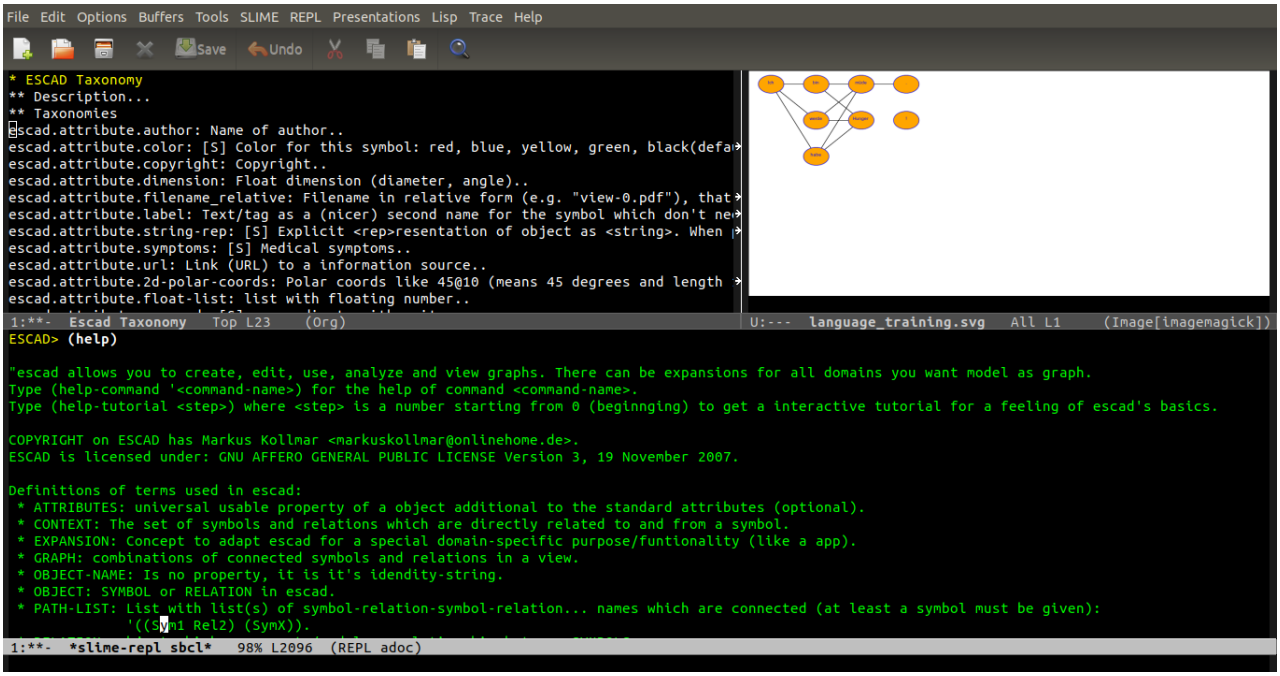


Figure 1.2: escad in emacs (note the appearance may differ from yours).

Figure 1.3: escad via graphical web-user-interface.

1.5 License

Escad strictly wants to be open source and helpful for many people. It is licensed under the *GNU AFFERO GENERAL PUBLIC LICENSE Version 3, 19 November 2007*. For more information see the file `LICENSE` in your escad root-directory.

1.6 Installation and starting

Currently there are no preconfigured packages for convenient installation of escad. However installation should not be to difficult, since escad-development tries to minimize not shipped dependencies.

1.6.1 Linux or unix-like systems

1. Get the repository from `https://github.com/mkollmar/escad` by clone it, or download repository as zip-file and extracting it. The escad executable is at the root folder and is called `escad`.

2. Optional: If you want make the excecutable yourself (in the case the preinstalled does not work) or to load the lisp sources in your lisp-implementation und run it, you have to ensure that you have installed the lisp-library dependencies in your lisp implementation and loadable by the common lisp *asd* system approach. Escad is developed currently with **sbcl** common-lisp implementation and additional depending lisp packages according the escad.asd file should be already installed. To install these librarys you can use your distribution-package-method or the **quicklisp** tool. In case you use quicklisp following `https://www.quicklisp.org/beta/` instructions to download and verify quicklisp, load it in sbcl, load the required systems and store them in the sbcl-init file `.sbclrc`: curl -O https://beta.quicklisp.org/quicklisp.lisp; curl -O https://beta.quicklisp.org/quicklisp.lisp.asc; gpg –verify quicklisp.lisp.asc quicklisp.lisp; sbcl –load quicklisp.lisp; (quicklisp-quickstart:install) (ql:system-apropos ”hunchentoot”) (ql:quickload ”hunchentoot”) (ql:quickload ”jazon”) (ql:add-to-init-file) (quit)

   In either case you have to make sure that your asd-system can find all the needed library-packages. Then in the root directory of the repository execute `make executable` and after some while you will get it.

3. Optional: For some PNG/SVG export functionality you need to have/installed **graphviz** and for PDF exports you need to have/install a **lualatex** installed with your distribution-package-method.

4. You can now try to run escad (in terminal-mode) within the escad-root-directory. If you want the command line interface mode then type just `./escad`. If you want use the browser gui then type `./escad` and enter in your browser `127.0.0.1:4242/`.

5. Optional: To make initial settings for escad check the config file `escad_conf.lisp` and edit if necessary. This file will be read and executed at start of escad and contains escad lisp commands, like the one you type in at escad command line.

1.6.2 Other systems

Sorry, for now other systems are currently not tested, but may be possible to work, since common-lisp is available for wide range of os. However some additional tools may be not available or have different behaviour so that not all things work like under linux. Another way is trying to start linux in a virtual environment, but knowledge is needed. If there is more man power it may be possible to improve that situation.

1.7 Concept and terms

To understand a software-system it is often easier to understand the theory behind. In escad this is quite simple, since it is practical use of graph data structure (in informatically or mathematically means). But simple means just in the basic building blocks, not in the power, since graphs can be quite big and nested. Such **graphs** consist only of **symbols** V (vertices, nodes, Knoten) and **relations** E (edges, Kanten). Mathematically (see also Bronstein et al., 2008) you can see this in equations 1.1:

$$\begin{aligned} V &= \{s_0, s_1, s_2, s_3, \_escad\} \\ E &= \{r_0, r_1, r_2, r_3\} = \{(s_0, s_1), (s_1, s_2), (s_2, s_3), (s_3, s_2)\} \end{aligned} \tag{1.1}$$

You see in math you would need no name for relations, as it are ordered pairs of symbols. Escad uses names for symbols and relations. This is because relations can have some optional properties (symbols too), which you can acess via the relation-name. As convenience you can let escad give you automatic generated names, in case you do not care of the exact name. See figure 1.4 about the structure of escad. The three grey blocks are needed for the interaction with the user (commandline or web interface). Thus they are explained not here but will be looked later in detail. In the following sub-chapters the other three parts and their contents are explained in detail. However

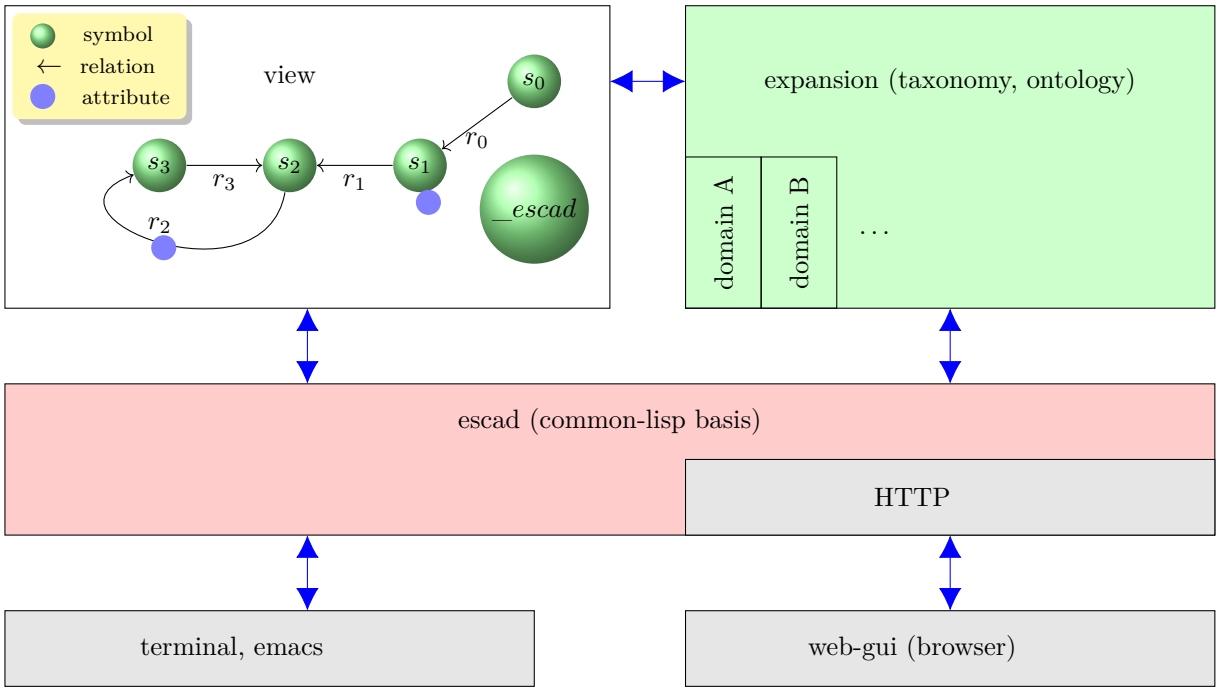


Figure 1.4: Overview of basic escad structure.

a short additional security note on the communication blocks: You can communicate with escad

through common-lisp. This communication can also piped through a local TCP-connection by the user. Note that for security reasons you should not make such network-communications over the internet, since escad is not encrypted. This is also true for the HTTP connection. It is currently only meant for local network use. However you could use technics like VPN if you want secure transfer through the internet.

1.7.1 Symbol - name things to talk about

In many papers a symbol is also called node. However escad has the aim to model things into software - a symbol for a concept. Thus a node is a *variable* for something in our thoughts. This symbol can represent a house, a number, a theory, a joke or whatever you want. This shows the great flexibility of escad. In figure 1.4 symbols are circles with names in it (e.g. s0) which you can name however you want, as long it is a unique name in the current graph. In case you do not care, escad can even generate a unique name for you.

1.7.2 Relation - describe relationships of symbols

A relation (or in computer science also called edge) creates a relationship or dependency between two symbols. You can specify the meaning of this dependency further. Try to use the most specific domain expansion for your model in order to get the most advanced tools. However sometimes the detailed model is not clear in detail so that you can use a most generic relation taxonomy.

1.7.3 Attribute - for storing your hashtags

An attribute in escad is an *optional* specifier of a symbol or relation, mostly used for searching or tagging object(s). There can be a huge discusion whether attributes or whether a symbol and relation should be used instead. Attributes can lower the complexity of an graph. On the other hand attributes are implicite relations. Take your choice and try to keep this choice consistent over your graph. Consider that attributes may not taken into account by some graph functions. Attributes merely are *keywords* which give the symbol or relation additional information. An indicator to use a relation and a symbol instead of an attribute is if you want to supply to such a keyword an value. E.g. to describe that a symbol uses/needs a file, you should create a file-symbol and create a use-relation.

The attribute contains text with a special structure. Every attribute-entry (key) begins with a *hasthag* "#" followed by a string with no spaces in it. You can add many such hashtags by seperate them with at least one space char.

1.7.4 View - where all lives in

You can think of the view as a workplace where you can put in symbols and relations, which model an part of the real world. It is called view because it is the current view of a world modelled in your escad session. However like in the real world there are often multiple and different views of the world. Thus you can acess other views through a view-symbol in the standard expansion mentioned later.

1.7.5 Taxonomy - semantic of objects

With taxonomy you can create in escad a classification (domain-based) to your symbols and relations by just tagging them. Depending of the taxonomy there may result different (graphical) output or behaviour (functionality) by escad. This may sound clear, since without that a computer

can not know what a symbol means just by interpreting its name. But it is merely like a seperator in your folders - you can name and seperate things but you may not do much more. More is possible with the ontology.

1.7.6 Ontology - behaviour of objects

In semantic-knowledge field you may hear often words like ontology. Ontology *uses* the taxonomy to capture and represent the meaning of a domain and has rules what is possible with it and what not. Since escad is aimed to be a practical tool, it has some domain specific ontologie-knowledge included. This knowledge is provided by an expansion.

1.7.7 Expansion - customize your needs

Expansions are a collection of definitions of *symbols*, *relations* and *functions*. They provide mostly domain specific *ontologies* which assure that one can model with a specific taxonomy and rules of who symbols and relation are combinable and which functionality they provide. This ranges from graphic output to new generated graph-elements. There are many possibilities and you can even write your own expansion(s). Those programms live in the escad environment and can use the provided feature, even of other expansions. However currently there is only a limited set of expansion, but this could increase in time. Feel free to write a expansion for your domain specific needs.

When we speak of *activation* of a expansion, what does that mean? You as user give a activation command or right-klick on the symbol, then a special function is executed. What special function is that and what does it make? Well in fact it is the *activate* function. This function can take additional parameters. You can specify them within the Attribute *activation_command*. The default value is *neighbourhood* which does what it says: It shows all in- and outgoing relations and the symbols which they are related to.

1.8 Usage patterns

To get things done with escad, there are several tasks you have or want to do. In this section we provide you with common interaction concepts in escad. It is very useful that you get familiar with them even if you do not know the exact details. So you know what tasks or problems may arise and which facilities escad provides and suggests for this. Detailed examples you will see in the next chapter.

1.8.1 Modeling

Modeling is probably one of the tasks you will do the most time with escad: You describe in escad your topic/task of interest. This is a very important task. If you model something, look at the given examples in the `examples` directory. Use the most approbiate taxonomy. Use expansions to ease your live where possible. Modelling semantic graphs is not a easy task. Of course you can make easily symbols and relations, but it heavily depends how you use them. In the past many semantic projects are died or got not very sucessfull used at a wide range. Barry Smith stated some reasons (`https://www.youtube.com/watch?v=p0buEjR3t8A`) like silo-syndrome, short-half-life-syndrome and reinvent-the-wheel-syndrome. Nowadays there is a sucessfull example in bioinformatic where to show how genes affect our biology (see BFO-ontology). However even with the best ontology you can have problems, if you not use it properly. To use it you should have a clear knowledge on what the terms mean and how to use. Such problems may also occur with escad. To improve

the situation, escad tries to have many examples for several domains and tasks. The aim is to have enough examples so that you have the possibility to adapt it easily to your needs. I believe it makes more sense to use semantics than to have endless discussions what is the "correct" taxonomy/ontology.

1.8.2 Structuring

The more you model, the more your view will get filled and it is hard to keep an overview (especially in the graphical user interface). You need to handle complexity (and in the end also performance). One method is *grouping* symbols which are in close relation to each other. For example if you model a 3D object like a cube, you may group all symbols defining the cube together. This can be done by the group-symbol.

1.8.3 Mounting

Sometimes you want hide complexity by including another view which even may be on another computer/network or in a graph-database/disk/file. To connect this seperate view with your actual view you can mount symbol(s) of it to your current view. This is like a door where you can hide complexity but if you need more you can open and connect your view with it. Mounting is done with a expansion. After usage you can unmount the seperate view. You would want to do this especially when a complex mounted view is connected through a slow network. Then escad will react slowly too, because it has to fetch/write data to it.

1.8.4 Exporting

Saves your view (or just parts of it) in a non-escad format. This is usefull for use your view in other programms. In case you can not do some things in escad yet or other tools may do better, you can export your graph. You can export the view to

graphviz-dot is a file format for graphviz, which is powerful in graphically layout and drawing of graphs. Export with a expansion.

SVG is a vector graphic-format viewable through most modern webbrowsers.

PDF Export with a expansion.

Consider that while exporting you can loose some information or the information is frozen. This not has to be an bug or limited functionality of escad. This also occurs if the target format not supports some features or mechanisms.

1.8.5 Importing

Allows you to get a view stored in a non-escad format. Because interoperability is important in todays heterogenic software world, there should be a way to get graphs from other software. Graphviz is powerful in graphically drawing of graphs. You can import those graphs with .dot extension. However only basic functionality of dot is currently supported. Import with a expansion.

1.8.6 Reporting

Means to create/process/analyse new things out of your view. E.g. a report could check your view for data-quality like whether there would occur problems when activate a specific expansion or if there are circular graphs.

1.8.7 Generating

Is used to create new symbols, relations or attributes automatically for you. E.g. if there are many symbols and relations with same taxonomy connected sequentially, you may let generate them by just providing the basic data which differs.

1.8.8 Automating

With *flows* automating in escad is possible without to know how to programm in lisp.

2 Getting started

Here i assume it is easier to get what escad can do for you, by showing escad in work. Note that the look of the following examples may differ a little from the real software, but the semantic should be the same.

2.1 Tutorial “room planing” with graphical browser-interface

We want now do a web-session in escad. Go to the escad root directory and start escad by typing in a shell in your terminal:

```
user@host:~/escad$ ./escad
>
```

This starts escad and a http-server if you have the default settings in your `escad\_conf.lisp` (with the symbol *_web* activated). With your web-browser you can browse the page which at first time loads a web-client in your browser. After that you can view and edit the graph like in figure 1.3.

2.2 Tutorial “room planing” with escad command-line

We want now do a short session in escad. After installation go to the escad root directory and start escad by typing in a shell in your terminal:

```
user@host:~/escad$ ./escad
>
```

This starts escad REPL (no special arguments given). After loading of common-lisp and escad, you should switch from the common-lisp in the escad namespace:

```
> (in-package :escad)
ESCAD>
```

Now you can execute all escad commands directly. To see some possible existing symbols type following escad-command:

```
ESCAD> (ls)
(``_escad'')
```

You can read more about this command at page 24. We see a symbol which escad has already created for us. The symbol can contain settings for our current escad session. However we heard that escad has symbols and relations. Lets look at the relations:

```
ESCAD> (lr)
NIL
```

Upps, we get *NIL* which is the lisp way of saying that there is nothing. But that is ok, since we just have not created anything yet. So lets create three symbols:

```
ESCAD> (dolist (name '(``one'' ``two'' nil)) (cs name))
NIL
```

This produces three new symbols, which names we not see because of the `dolist` nature:

```
ESCAD> (ls)
(``_escad'' ``_view'' ``one'' ``two'' ``s0'')
```

The third symbol name `s0` created escad for us, because we provided `nil`. Now lets create a relation:

```
ESCAD> (cr nil ``one'' ``two'')
``r0''
```
Now we got
```
ESCAD> (lr)
(``r0'')
```
Note that you can not insert a new relation or symbol which name already exists. You now have nearly seen all basic operations in escad. Most functionality can be achieved by this. Instead of learning many new commands, you can use in escad symbols which can also represent actions (e.g. exporting a PDF of your view). But how can you execute such a symbol. This makes the command *as*, which *activates* the given symbol:
```
ESCAD> (as ``s0'')
(``Documentation text...'')
```
You can activate every symbol, but most of them will just print some documentation about themselves, like you have seen in the last command. To get another functionality you have to assign a taxonomy which refers to a expansion. Those expansion is then loaded and executes a defined command (which is defined by the taxonomy). You can easily add taxonomy to a symbol with add a *property*:
```
ESCAD> (s ``s0'' :taxonomy ``export.pdf'')
(``s0'')
```
If you would activate this smybol now, it would produce a pdf with graphic output of your current view. Because you gave no file-name it would generate some. To give a filename you can add a special attribute-property:
```
ESCAD> (s ``s0'' :attribute ``my_file.pdf'')
(``s0'')
```
Now you should get those pdf-file. This are the most basic commands you need to know in escad. To get detailed command info use command help:
```
ESCAD> (help-command 'cmd_name)
(``Documentation text of cmd_name...'')
```
To get basic help type:
```
ESCAD> (help)
(``Documentation text...'')
```

2.3 FAQ

Sometimes questions lead to a fast recognition of a problem or help to understand things better. Not all questions may be frequently asked, but who cares :-)

2.3.1 Scope

Does it not make more sense to store the graph in a graph database? In some cases it could make sense. However escad is not meant to be a high performance graph storage tool. Think escad as a workshop where there are many tools and workpieces. A graph database is just one specific high efficient tool. Escad is the host from which the user can interact in a unique interface with many of such tools. In fact it is planned to implement a mounting facility for graph databases. So that you can additionaly store your view in it.

I really do not understand whether escad is for me? Do you search a nice software to make crazy pictures of your graph in a interactive way? Then you currently may not be right here, just look at something like gephi. Escad is for people who want to work exactly and

reproducible with graphs in a more describing manner and in a second step there will be an output (similar like batch processing).

Can i make complex queries like in a graph database? Perhaps you need a graph database, escad is not a pure graph database. It is a tool, not designed as a speed optimized storage container. However it is planed to have a graph-database connection (e.g. to great arangodb) in order to support persistent huge graphs.

2.3.2 Usage

Is there a difference in the power of the different escad interfaces? Yes. The most powerful is currently the command line interface. New commands appear first there. The aim is to keep the functionality equal across the interfaces in a later step. But each interface has it strength, thus it makes sense to use all. Where the commandline has powerful scripting capabilities, the web-interface allows a more intuitive and haptical way to interact with the graph and shows a fast overview.

Why you have choosen common-lisp as the language for escad? While having some experience with different languages, lisp has one of the clearest syntax for me. The REPL allows interactive rapid prototyping. Lisp is an old but mature language. One can implement easily a domain specific language (DSL) if needed.

2.3.3 Development

Can i help develop escad? Of course, very welcome! This currently is a project done in spare free time beneath work. So just contact the developer in github to improve escad together.

2.3.4 Other

Is there support for other languages as english? No not currently. The main author is german, but has spare time to keep manual in german as well. Currently english is choosen to reach a wide international range of people. If you want contribute with translation for your language feel welcome.

3 Reference

Note in order to understand the reference you should be familiar with the section 1.7. Here you find the detailed information to work with escad, such as available commands or interfaces. Also you should get some basic things in order to get the idea in how you can extend escad or help in development. See figure 1.4 for how escad is structured. Basically you get with escad two parts of software. The lisp side as *server* and in case of the gui the javascript side, which acts as a *client*.

3.1 View in detail

The figure 3.1 shows the default view in detail. It lives in the server and holds all your symbols and relations. You can see the default automatically generated symbols which are explained later. For the first left relation there are the reference slot names given (printed in red). The symbol *_escad* has no incoming references, thus there is just the *ref_to* slot with relations. Further you see the first relation which has both references bound to a symbol and this is usual for relations. This relations references to the symbol *_configuration* which thus has just *ref_from* slot filled. Note that the view is your main workplace. There may exist other escad views in other computers.

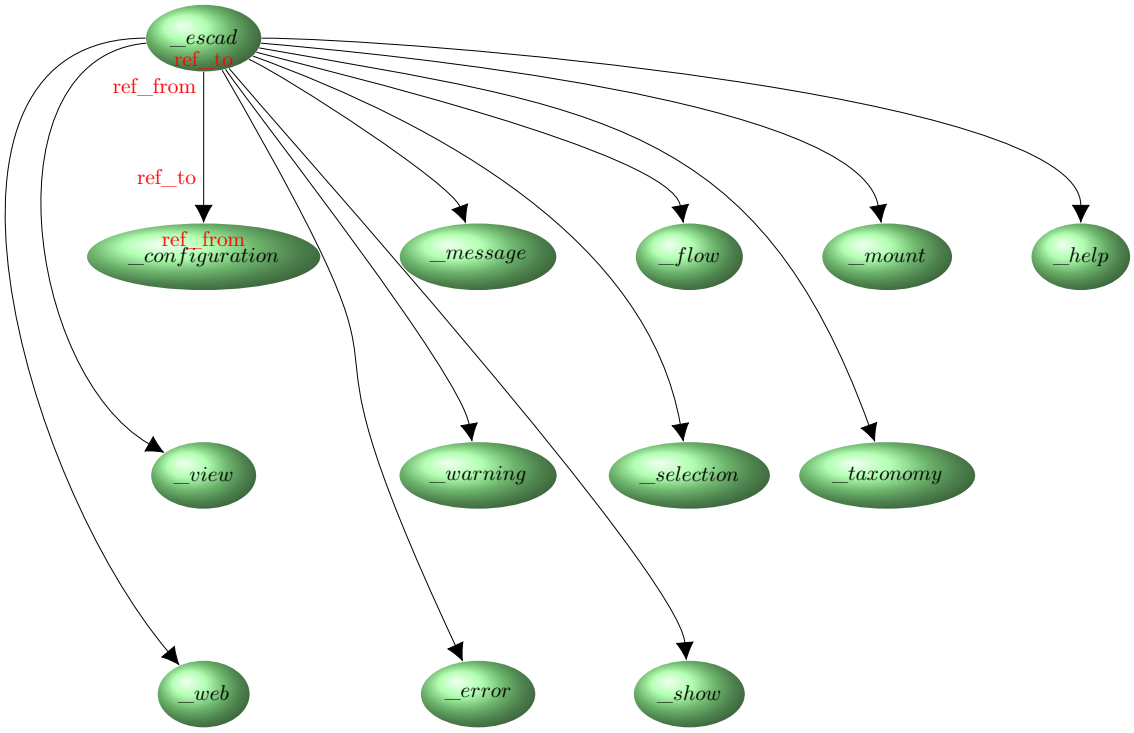


Figure 3.1: Default view and reference-slot names (red colored).

In future escad-versions you may even mount this other views (from file or internet) or from other software tools. In order to acess all the symbols and relations, escad requires a *unique ID*. Even if symbols or relations could technically have the same id, it is adviced to not do this. Some graph-databases do not allow same id for symbols or relations. In that case data exchange with that tools would not work or cause additional work to solve this.

3.2 Symbol in detail

Table 3.1 shows the symbol slots (properties) meant for editing by the user and their meaning.

slot (property)	description
activate	When a symbol is activated this slot-content is submitted to the activation function.
attribute	Used sometimes by expansions for their work. It can hold different objects, just text or even complex objects.
comment	Tells something additional about the symbol. This is only meant for the user and should not changed by some automatic code.
label	Alternative name which has not to be unique. This can be used by the user to provide longer names or multiple identical names (for whatever reason) but however it is possible to use same name like symbol-id too.
representation	Text representing the symbol. The main data slot which contains data by the semantic which is classified by the taxonomy.
taxonomy	Classifies the symbol and determines which activation-function will be called. If there is no taxonomy given, then symbol-activation does nothing.
visual_cli	Data how a symbol is visualized in the command line interface.
visual_web	A JSON string specifying data how a symbol is visualized in the web interface (position,....).
weight	A number which denotes, depending on the usage, the importance / weight / order of the symbol.

Table 3.1: Slots of the symbol object.

3.3 Relation in detail

In escad every relation is technically *directed*. But this is just to store the direction in case you need it. Of course you can also model undirected or bidirectional graphs in escad. To do this you can use expansions which ignore the direction of the relation or you can use a special undirected- or bidirectional taxonomy or you can create two relations with the opposite direction to model bidirectionally. Escad also supports *multigraphs*, so that there are multiple relations to/from the same target/source.

Table 3.2 shows the relation slots (properties) meant for editing by the user and their meaning.

3.4 Internal symbols (“_”<name>)

Escad pushes it's concept of *all can be described as graph* clearly forward. So even escad itself shows errors or other informations about itself with *auto generated symbols* beginning with the character _. Thus you can for example check if an command created an error by search for the symbol *_error*. In the slot *representation* you can see the detailed error message. You have not to delete it after the error occurs but you can (in case to know that a new error exists another time). If a new error occurs the symbol will be populated with new message. All this symbols

slot (property)	description
attribute	Used sometimes by expansions for their work.
comment	Tells something additional about the symbol.
label	Alternative name which has not to be unique.
representation	Text representing the symbol.
taxonomy	Classifies the symbol and shows to which expansion it belongs.
visual_cli	Data how a relation is visualized in the command line interface.
visual_web	A JSON string specifying data how a relation is visualized in the web interface (position,....).
weight	A number which denotes depending on the usage the importance / weight / order of the symbol.

Table 3.2: Slots of the relation object.

can be populated from escad in the graph and are connected by a relation to the parent symbol *_escad*. Table 3.3 shows all internal symbols in the current version of escad.

3.5 Expansion

Expansions are a domain specific extension to escad. Basically a expansion contains taxonomy for symbols and relations. This is merely a classification which determines how escad works with those objects. If a symbol has no taxonomy then it is considered as the most general symbol. A taxonomy narrows the semantic of a symbol. The more detailed taxonomy, the more functionality you may get with an expansion.

There are expansions which are included/shipped with escad. Table 3.4 shows the available expansions included in the current version of escad. Of course there can exist many more not shipped one. This external expansions you have to copy into expansion dir and to load with the load expansion. After that you can check which additional expansions are loaded in escad yourself by exploring the *_escad* symbol (how this is done you can see in the 2). Escad not atomatically reloads changed files, to do this just activate once more the load expansion to get the most up to date state. To include external expansions, it is the same process like including the included internal expansions.

In the following subsections this included expansion are explained in detail, whereby following table headings are used:

object is either a symbol also denoted as **s**, or a relation which is denoted as **r**.

taxonomy is the unique taxonomy-name of the object.

in/out shows which objects are allowed, according the ontology, to connect with the object. If there is nothing given, it is not specified and you can do whatever you want. A dash like in "-/-" denotes there should be nothing incoming and outgoing, and this is of course only possible in case if the object is an symbol (not relation!). Multiple in- or outgoing objects are seperated by comma.

out shows which relations are allowed to point out of the expansion to other symbols/expansions. If there is none, the symbol is meant to stay unique.

attribute is either a **F** which denotes a file-name or a string **S** with explained meaning in the desccription-column. In case of an optional attribute it is placed inside brackets ([ ]). A empty field means a given attribute is ignored.

symbol	description
_configuration	Holds the path to the configuration file.
_error	An error occured. If the error is generated from a expansion, a relation will be added from this symbol to the corresponding expansion symbol.
_escad	Root symbol from which all other internal escad symbols are connected.
_flow	Connect symbols with this symbol to generate automatic symbol-activation of connected symbols, just like a (work-) flow.
_help	Root symbol for help about escad.
_message	Holds actual generated info messages about escad.
_mount	Childrens of this symbol show all mounted views (if any).
_selection	You can preselect symbols or relations for further actions. The selection for symbols and relations is different. by creating a symbol with taxonomy *std.symbol* or *std.relation* with the object name you want select in slot *representation*. After that create relation to the symbol *_selection*.
_show	Escad at default tries to show you all available symbols. However in case your graph is big, you may want just to see a part of the graph (this also improves performance). To do this, you can create a relation from this symbol to the relations you want see. Note that there is the attribute-slot which controls the depth of child-symbols you can see (default is *all* which shows all, *1* would show the given symbol and it first child-symbols). Escad always shows the *_escad* symbol, so you not have to relate to it.
_taxonomy	This is the root symbol from which you cann explorer all actual loaded expansions und thus taxonomies.
_view	Root symbol of view related things. You can find all *mounted* views there als child nodes.
_warning	Warning message.
_web	Controls the server for the graphical web-interface. Activated with *activate-slot* `start` starts server, *activate-slot* set to `stop` ends the server. The *attribute-slot* can hold the root directory for the web-server (where your index.html lives).

Table 3.3: Auto generated internal symbols of escad.

topic	file	description
standard	`standard_expansion.lisp`	Provides common functionality for broad usage even for other expansions.
import	`import_expansion.lisp`	Import graphviz dot-file in view.
export	`export_expansion.lisp`	Export graph to PDF and SVG.

Table 3.4: Contained expansions in escad and their functionality.

description tells what the expansion does.

3.5.1 std expansion

In the following table 3.5 the included std-expansion is shown. It contains symbols and relations which are helpful in many domains and helps the user to be faster in generating graphs.

object	taxonomy	in/out	attribute	description
s	std.load		**F**	Load a lisp file (e.g. expansion).

Table 3.5: Standard expansion.

3.5.2 import expansion

The table 3.6 shows the included graph-import facilities for escad.

object	taxonomy	in/out	attribute	description
s	import.dot	-/-	**F**	import from dot (graphviz).

Table 3.6: Import expansion.

3.5.3 export expansion

Table 3.7 shows the included export-expansion for exporting the view into different document-formats.

object	taxonomy	in/out	attribute	description
s	export.dot	-	**F**	exports view to dot (graphviz).
s	export.pdf	-	**F**	exports view to pdf.
s	export.svg	-	**F**	exports view to svg.

Table 3.7: Export expansion.

3.5.4 document.2d expansion

Table 3.8 shows the included expansion for describing two dimensional features like text, pictures or tables.

object	taxonomy	in/out	attribute	description
s	text	-	**F**	Simple text.
s	simple.2d	-	**F**	Describe simple two dimensional line sketch.
s	table	-	**F**	Describes a table.

Table 3.8: Export expansion.

3.6 HTML-webclient usage

Take your browser to communicate with escad in a graphical intuitive way. This starts only a http-server if you have the default settings in your `escad\_conf.lisp` (with the symbol *_web* activated). With your web-browser you can browse the page at `127.0.0.1:4242/` which at first time loads a web-client in your browser. After that you can view and edit the graph like in figure 1.3.

3.7 LISP command-line and server usage

The command-line provides the full functionality. This means you got a common-lisp REPL with escad-package loaded. This gives you the full power of common-lisp with the ability of the escad-commands to work with a simple graph environment.

3.7.1 Lifecycle and syntax

In a unix manpage manner, escad could be shortly described like in 3.2. In following the lifecycle of an escad session is explained.

1. Escad is started.
2. Init view.
3. If an optional given argument is given evaluate it.
4. Read `escad_conf.lisp` if provided in the current directory of the escad-excecutable or a given configfile (with path) in the symbol *_configuration*.
5. Init escad REPL.
6. Escad is ended by user or error.

3.7.2 ESCAD common-lisp subcommands

The table 3.9 explains used symbols, abbreviations and the data-structures in the reference. In general escad provides two flavours of commands: object- or text-based. There is no superior flavour, just use the one which fits your concrete needs the best. This command reference is sorted alphabetically, so the order of commands does not say something about the quality or importance of them.

PATTERN	**DESCRIPTION**	**EXAMPLE**
cmd	command name	**asa**
NIL	common lisp nil means not true/-done	`NIL`
RN	relation name string	``r0''
SN	symbol name string	``s0''
RO	relation object	``r0''
SO	symbol object	``s0''
STRING	string with unspecified or multiple semantic	``a message string...''
+	previous content can occur at least once or multiple times	
*	previous content can occur not or multiple times	
...	previous pattern can be continued	`(0 1 2 ...)`
()	basic common-lisp list	`(1 ``Hello'')`
[ ]	optional argument(s)	`(cmd [ ])`
function argument	function argument	
function result	function result, multiple values seperated by comma are possible	

Table 3.9: Explained abbreviations and symbols.

```
ESCAD(1)                                                    ESCAD(1)

NAME
   escad    - the command-line interface with integrated server for
   web-interface.


SYNOPSIS
   escad [escad-lisp-commands]


DESCRIPTION
   Escad is the <e>xpansible <s>ymbolic <c>omputer <a>ided <d>
   description tool. It provides you a common-lisp repl after
   starting. For initial commands you can use the argument
   escad-lisp-commands which is executed in the escad-namespace.
   A integrated web-server is provided too, so that you can work
   with your escad view also in a graphical user interface. Choose
   the one which fits best for your workflow. Note that you can
   also use stdin to execute escad scripts.
```

Figure 3.2: escad unix manpage.

as

```
[ SN ]
```

```
STRING
```

<a>ctivate <s>ymbol in current view. What happens depends on the taxonomy of the symbol. Many symbols print out a string as their contents. Symbols which represent expansions will execute the configured function of the expansion.

cr

```
RN | nil SN SN [ :attribute :comment :taxonomy :weight ]
```

```
RN | nil
```

<C>reate <r>elation with given name and possible additional values in view. If the relation-name already exists do nothing and return nil. Default type is undirected relation. To make a directed or bidirected relation, set the appropriate taxonomy (note that ref_from and ref_to are only technical terms meaning you first tie the relation from that symbol to another. it can mean that is directe, but it is not guaranted that the author means that unless he makes that explicit with a relation that declares that).

cs

```
SN | nil [ :attribute :comment :taxonomy :weight ]
```

```
SN | nil
```

<C>reate <s>ymbol with given name and possible additional values in view. If the symbol-name already exists do nothing and return nil.

lr

```
[ :filter :exclude-taxonomy ]
```

```
(RN*)
```

<L>ist all <r>elations in current schematic which name match the filter. Additionally exclude relations which match the exclude-taxonomy.

ls

```
[ :filter :exclude-taxonomy ]
```

```
(SN*)
```

<L>ist all <s>ymbols in current schematic which name match the filter. Additionally exclude symbols which match the exclude-taxonomy. See example at page 14 for usage.

r

```
RN [ :comment :ref_from :ref_to :taxonomy :weight ]
```

```
RO | nil
```

Get/set <r>elation object.

s

```
SN [ :comment :taxonomy :weight ]
```

```
SO | nil
```

Get/set <s>ymbol object.

rr

```
RN
```

```
RN | nil
```

<R>emove <r>elation.

rs

```
SN
```

```
SN | nil
```

<R>emove <s>ymbol.

3.8 Library usage

To use escad as a library in your common lisp programm, just load `package.lisp`.

4 Development

Everyone is needed and welcome for escad development. If you are a graphical designer, you are got in documenting or you like to program in lisp or you are interested in web-programming - all is required in escad. :-) The project is managed with the famous source-code management tool *git* (also known as used in the linux kernel development). The repository is hosted under `https://github.com/mkollmar/escad`.

A *Makefile* contains the basic task to generate the code or documentation. So it may be a good choice to look into it in order to get a basic idea how all works. Nevertheless the next sections contain how the development process is (currently) structured. It follows a brief overview over the directory structure and a explanation what it contains (or should contain in the future). The last section provides information in how to program a expansion and the interface

4.1 Development process and quality

As mentioned at page 6 escads development follows the documentation-driven approach. All follows this process in short:

1. Set/increment version according to semantic versioning (Preston-Werner, 2022)
2. Update/create documentation (this manual)
3. Develop code and test along documentation and go back to the previous step when needed.
4. Test and go back one or two steps when needed.
5. Tag version and deliver it.

A note to tests, as it seems to become one big factor of good code quality. Surely test are a good method to check well defined interfaces. So here we implement tests too. However because at the early development stage of escad and limited man power, tests will not take as much room as there could be. Since interfaces may change often we will concentrate at often used functions and in more stable versions of escad the tests should be increased.

4.2 Files and directories

The directory structure and some basic files of the repository will be explained in this section. In the root of the repository there are following files with their purpose:

Makefile contains the instructions for the *make* programm and controls the most part of escad development process (generating documentation, binary or other things). Just enter `make` in the command line to see available options.

LICENSE holds the license to which terms you have to use escad.

README.md is mostly for github where escad is hosted and contains shor overview about the project.

doc/ contains the escad documentation for user and developer. Currently this is mainly this manual written in *latex*. But in future a unix man-page, more examples or online documentation would be great, too.

lisp/ contains the escad-lisp files with the implementation.

lisp/escad_conf.lisp can contain user settings which are executed at startup.

lisp/expansion/ contains all escad-expansions which are available.

lisp/examples/ contains examples which can be loaded in escad. But note that currently not all may work yet, they are more a print what a future interface should look like.

misc/ contains miscellaneous things which have no other place where it fits.

test/ has unit tests, which execution may be controlled with make.

web/ with the web based graphical user interface (gui) things, which are executed in the web-browser, live here. `index.html` is the starting point for the gui and will be served first bei the server.

4.3 Programming your own expansion

Writing escad expansions means simply regular common-lisp CLOS-usage. That means defining classes and methods. Go through following steps carefully and you will have a good basis for writing your own. The code in listing 4.1 shows a minimal expansion.

Listing 4.1: Minimal expansion code of a taxonomy called “test”.

```
1  (in-package "COMMON-LISP-USER")
2  (defpackage :de.markus-herbert-kollmar.escad.expansion.test
3    (:use :common-lisp :escad)
4    (:export :my-function)
5    (:documentation "This is my expansion establishing a new taxonomy."))
6
7  (in-package :de.markus-herbert-kollmar.escad.expansion.test)
8  (defclass my-function (symbol-name-string)
9  "This function will be executed by calling this expansion.")
10 (escad:register-expansion test)
```

Following things you should keep in mind:

- Choose namespace.
- Provide documentation within your expansion.

You are welcome if you want include your expansion in this escad distribution.

Index

Bibliography

Bronstein, I. N. et al. (2008). *Taschenbuch der Mathematik*. 7th ed. Harri Deutsch GmbH. ISBN: 978-3-8171-2017-8.

Preston-Werner, Tom (2022). *Semantic Versioning 2.0.0*. URL: `https://semver.org/` (visited on 23/01/2022).

List of Figures

List of Tables