BE USER FRIENDLY, OCCT

Open Cascade Technology

If your work involves interacting with people at any point, you should serve them as well as possible. Good service has to be honestly fought for. Nobody wins fans by cheating; favor comes naturally from making a good product and genuinely caring about people.

Nick Disabato, “Cadence & Slang”

Julia Slyadneva

Content

Preamble

Here documented just some thoughts why OCCT has non user-friendly environment leading to the users outflow and what steps can be done to improve that. These thoughts are not only the personal experience but the quintessence of some more developers' opinion and investigation of the users' activity on the external resources dedicated to OCCT usage (listed in Appendix chapter).

All of it discovered how much effort must be done to make a step inside the OCCT world. The users have to invest the great time and psychic resources to find or get required information, understand the OCCT foundations or try it at least as the official documentation and samples do not meet the users' expectations and needs, OCC representatives are silent to the user's claims on the official forum in most. To make sure in that it is enough to overview the official forum's topics. Moreover, it is one of the reasons why OCCT community works with own OCCT version, develop own samples and write own blogs and wiki pages dedicated to OCCT, best practices of usage and how-to(s). All of it forms a bad reputation of OCCT in the world net and makes the new-comers and potential users to think that they will be alone in their 'fight' with OCCT monster. Consequently, they either buy a ready-to-use solution (not OCCT services of course) or are loyal to OCE branch more than the official one.

Bear in mind, that today's user could be OCCT customer tomorrow and keep your eyes critical.

What is wrong

Documentation

The official documentation has 2 main problems: mess structure and poor quality of articles.

Regarding information architecture – just some confused situations below as an example of incorrect information organization.

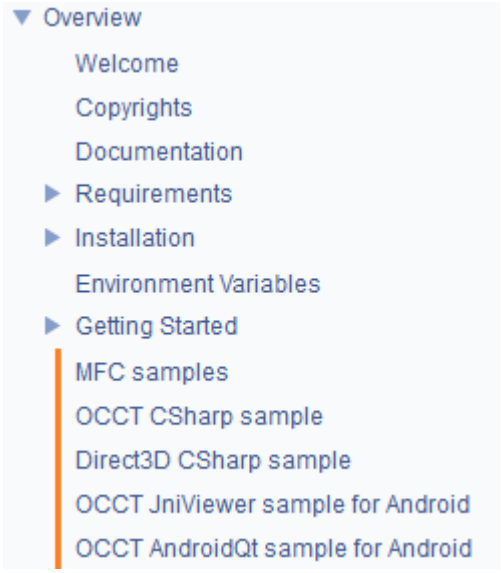
From the first glance to the content structure (see the picture above) the user may have right guess about each chapter is for, except *Technical Overview* for sure. Naturally, it is just an introduction to OCCT library modules represented by *User Guide* in the details. It makes sense to move it into *Overview* in order to avoid focus spraying – one place for primary notions.

OCCT foundations like library organization, handles, memory management, exceptions etc. are hidden somewhere in *User Guides -> Foundation Classes* chapter. Without understanding, at least, OCCT structure and handles the user can hardly go further. So, documentation must highlight these OCCT features more properly; the user must have explicit access to them.

According to the *Installation* chapter the situation when the user needs to build OCCT from source raises often: "*In most cases you need to rebuild OCCT … See Building OCCT from sources for instructions…*" For that he will need build instructions and requirements which located in *Developer Guides* and *Overview* accordingly.

> *Question.* If the situation when the user builds OCCT manually from the sources is often why he has to look these instructions in *Developer Guides*? The user is not OCCT developer – mentality break. The same is for best-practices and hints (for example, '*Debugging tools and hints*' chapter). The user must not be OCCT developer or contribute to OCCT in order to debug his application based on OCCT.

If the user searches the OCCT usage programming samples obviously he goes to the *Tutorial* chapter for the help and stops the searching with a great probability. However, later and accidentally he can face more samples in *Overview* chapter:



And even more in *Getting Started* chapter after looking over Draw items *("Experimenting with Draw commands" is definitely not right thing to start with OCCT).*

Modeling Data chapter contains approximation and interpolation algorithms from *Geom2dAPI* and *GeomAPI* toolkits which are part of *Modeling Algorithms* module according to the *Reference* documentation. Why does *Modeling Data* module define any algorithms? Maybe it makes sense to redesign it or rename at least?

If the user wants to have a full idea about Draw framework and its abilities he has to overview the following chapters: *Overview*, *Technical Overview*, *User Guides* and *Developer Guides*. Does he still want to be in-touch with Draw? Not sure.

The documentation contains the license paper text engaging a content header item named *License*. What for the license text is here? The license type indication in *Overview* is enough for the user.

The documentation has *Namespaces* chapter. What for if *Reference* documentation exists?

"Put a lipstick on a pig but it is still a pig." It is not enough to review the information architecture of the documentation only. The articles content must be revised as well, especially in *User Guides* chapter. Most of articles extremely hard to read and stick to the main idea because of excessive architecture and realization details. They are boring and messed and very long. Try to read articles dedicated to OCCT visualization or OCAF – impossible to be involved.

"Please; is there any simple example with implementation of OCAF tree?; I have been reading OpenCascade Docs, but it is very complicated for me..." *Origin.*

Samples

OCCT samples have a set of problems:

- They pursue the different objectives implicitly for the user. Thus, while MFC samples demonstrate OCCT functional range with possibility to look over the code behind use-cases and copy/paste it into his project environment, Qt/C#/Java samples are just techniques how to bridge OCCT with GUI frameworks. But the user is not informed about that and can miss the opportunities provided by the samples. For example, if he is looking how to implement a certain issue he will not find an answer here with a high probability because MFC framework is a dead framework and the user has a low interest to it;

- MFC samples demonstrated OCCT features are not cross-platform samples. No possibility to play around OCCT on Linux and Mac systems;

- It seems there is no non-regression testing system to support a stable samples state;

- They do their work not properly – not meet the users' expectations and needs.

Look at the samples in more details to argue the last mentioned problem.

MFC samples

MFC projects overview functional variety of all OCCT modules and provide a code logger to see the code base for each use-case/command. Sounds encouraging but…

The access to functionality variety is organized via toolbar looks like at the picture below.

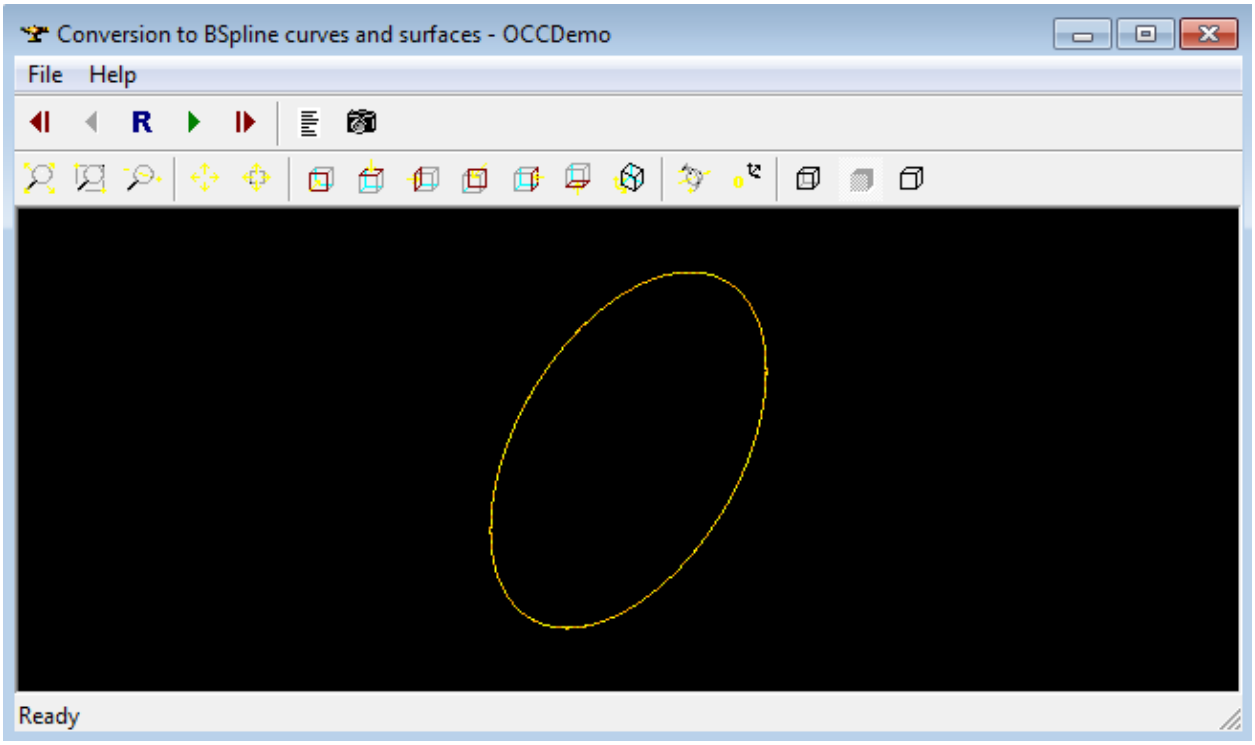
Poor user if he wants to find a certain functionality here! The only aim of the toolbar idiom is to provide a quick access to the application abilities for an experienced user. It is suitable for newcomers in no case.

The code logger becomes more and more useless with each new version of OCCT because the code demonstrated there is just a text block and very doubtfully that anybody here cares about its consistency and freshness.

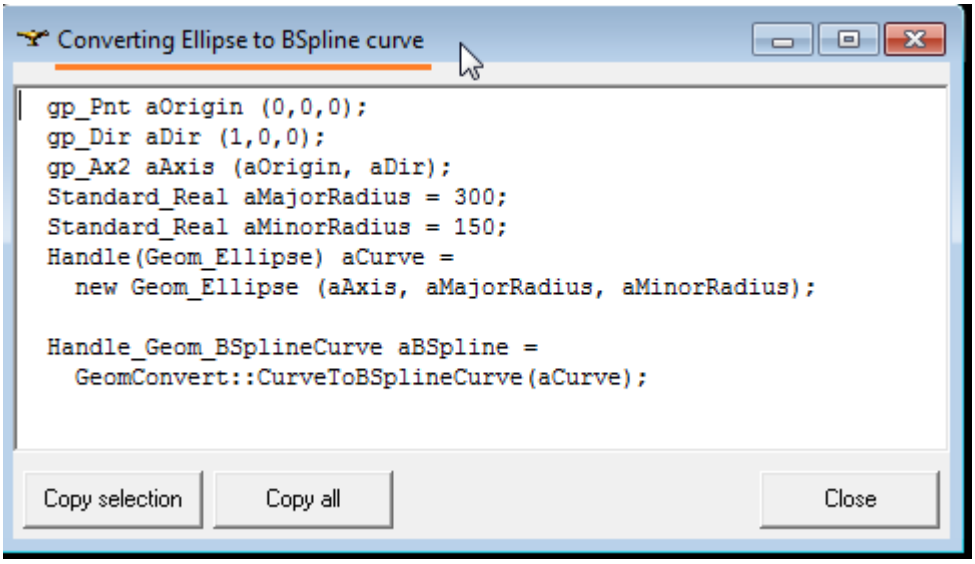
```
···myCResultDialog.SetText("··TopoDS_Shape·aShape;\n"
···"\n"
···"··//·initialize·aShape\n"
···"··//aShape·=·...\n"
···"\n"
···"··//·define·parameter·triangulation\n"
···"··Standard_Real·aDeflection·=·0.1;\n"
···"··\n"
···"··//·removes·all·the·triangulations·of·the·faces·,\n"
···"··//and·all·the·polygons·on·the·triangulations·of·the
```

Moreover, there are tons of other user interaction mistakes that complicate user's involvement into application and consequently his will to continue attempts to find here something useful for him. Just some examples:

Conversion to B-splines



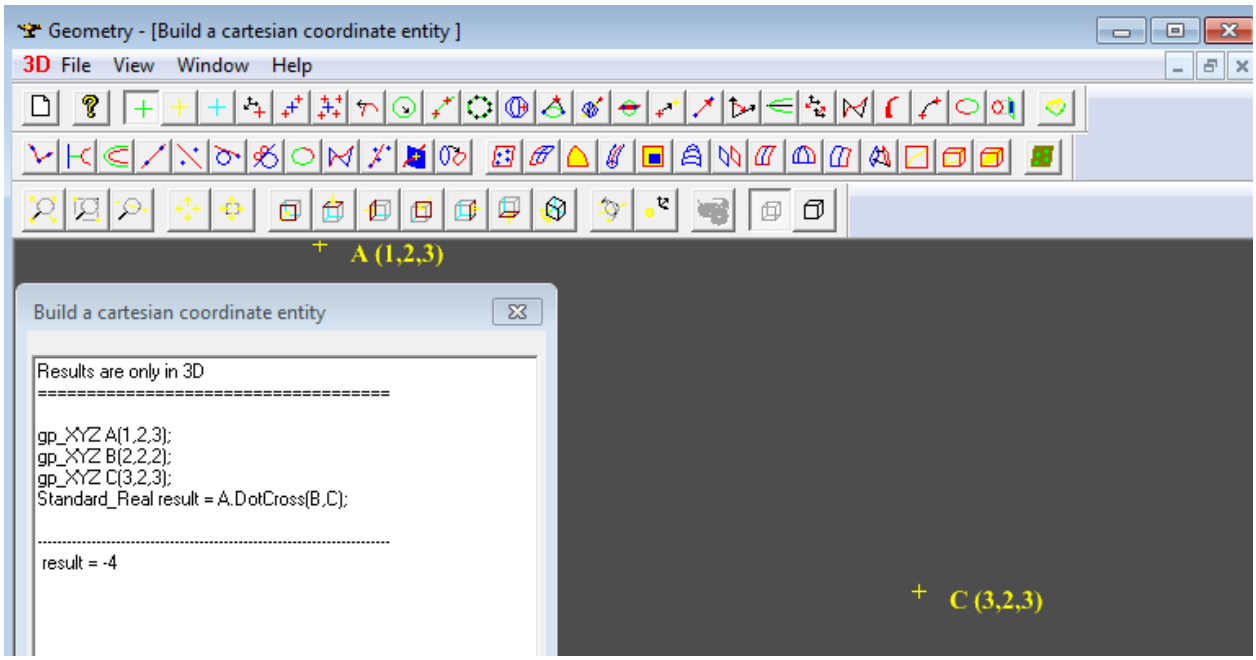
No one hint what to do and why this sample can be useful for the user. The main control listing the source code and explaining what happens in 3D scene is hidden by default.



The *Source code* window is a not topmost window so it disappears each time when the user clicks inside the main window.

The *Repeat*, *Next/Previous*, and *First/Last* controls are not informative – no idea about the essence of a case and cases total count. They do not provide the user with a possibility to select a certain case – he must look over all cases. The *Repeat (R)* button is even useless. The list view or combo list idioms will be more appropriate here. At least, status bar could display the current case description.

Geometry



As it was mentioned above the toolbar idiom is aimed to simplify the experienced user's workflow. The user may be experienced only in the application which he uses often. The product demo applications is not that kind of applications.

3D scene does not support mouse-only manipulations. To rotate, zoom and pan the user must deal with mouse in couple with keyboard. Uncommon, poor, not reasonable.

The *View* menu item does not contain an item corresponding to *Source Code* window. So, if the user occasionally closed this window he must guess to click *New* command. Not very obvious.

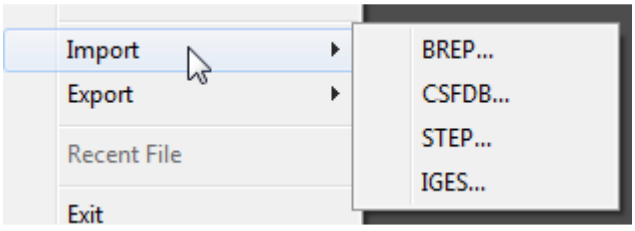
Simplification case (the "dark green rectangle with light green circles on it" icon corresponds to it) fails because of:
"C:\OpenCASCADE6.9.1-vc12-64\opencascade-6.9.1\samples\mfc\standard\win64\vc12\bin\..\..\..\01_Geometry\..\..\Data\shell1.brep was not found. The sample can not be shown." *What is about non-regression system?*

Import/Export

The lined toolbar items have no tooltips, no annotations in the status bar. So, it says like *"well, do not try import/export functionality, nothing interesting!"*

Export functionality makes sense only if the 3D scene is not empty, otherwise the user sees the warning window. Extremely bad and impolite practice. The items corresponding to the export functionality must be disabled until the limitations are respected.

No possibility to activate *Source Code* window.



The picture above demonstrates that the user has no chance for mistake – choose a file format here and once, otherwise close *Open/Save* dialog, go to the menu and correct the choice. The *Open/Save* dialog must provide the user with a format selector instead of a drop-down menu item.

The *Open/Save* dialog does not preserve the path the user defined during the last interaction session.

There is no need to overview other MFC samples – all of them have almost the same limitations of interaction design.

Qt/C#/Java samples

Qt/C#/Java-based samples (except Qt Tutorial) are about Import/Export functionality which unlikely is the point of user's interest. The users need these samples a) to make sure that integration OCCT with different frameworks really works on different OS platforms; b) to understand how to set up OCCT environment; c) to get hints how to wrap OCCT entities by Java/C# calls and d) to get known how to integrate OCCT visualization into application window/widget/control with support of panning, rotation and zooming (*just note that it is one of the frequently asked questions caused by the lack of good quality documentation for OCCT visualization)*. For c) and d) cases the answers can be found in the source files, but for some samples it is not a trivial task because of complex code architecture – several samples are based on one code baseline. That's all takes additional time resources.

Any product's samples are worth a thousand words, but it is not option for OCCT samples. They demonstrate unconcern about user's time and needs and good self-presentation.

"Please don't guide me to see the bundled examples with OpenCascade. I don't think it really helped as it has too many files attached to one working example." *Origin.*

Draw

Despite on the great ability potential of Draw for users' workflow, it unfortunately stays at the shadows. Some users do not recognize how Draw can help them, another consider it too complicated. There are some possible reasons for that:

- The documentation does not accumulate Draw related info in one place, so the user must look over all content items before he finds the most appropriate one. Just to remind, Draw is referenced in *Overview*, *Technical Overview*, *User Guide* and *Developer Guide*. It definitely slows down the user's learning process of Draw opportunities and motivation to deal with it;

- The documentation does not underline the benefits which Draw provides to the users. The most pages start like "*Draw is a test harness for Open CASCADE Technology. It provides a flexible and easy to use means of testing and demonstrating the OCCT modeling libraries."* So, Draw is a tool to test and demo OCCT libraries. Ok. But what does it mean for the user needs? Is it rather valuable for him?

- No fulfill user's manual. No one page describes Draw GUI, in particular, menu bar providing the different options including samples and help center. Not all types of viewers support the direct manipulation of the camera but those which support do it in combination with Ctrl – uncommon interaction approach, the users do not familiar with this idiom. All that features of Draw must be documented as well;

- Draw commands are documented not properly or not documented at all;

```
Draw[9]> help *source*
getsourcefile    : getsourcefile, or getsourcefile command
xsource          : lit les commandes depuis un fichier
Elapsed time: 0 Hours 0 Minutes 0.00471672700951 Seconds
CPU user time: 0.0156001 seconds
CPU system time: 0 seconds
Draw[10]>
```

- It seems there are no naming conventions for Draw commands. One of them can be hardly decrypted intuitively (for instance, some graphic commands ‘u, d, l, r, vr, mu4’), another is hard to remember (for instance, ‘pmirror’, ‘lmirror’, ‘smirror’, ‘dpmirror’, ‘dlmirror’ is named using *subject-action* format while more human-oriented naming approach is *action-subject*), third starts with a capital letter while other not (for instance, OCAF related commands like ‘NewDocument’, ‘IsInSession’, ‘ListDocuments’ etc.);

- For sure, OCCT developers have a tons of best practices and tricks to be share with OCCT users. But no one documented or formed as tutorials. To start, FAQ chapter can be developed answering the questions posted on the forums. For example, see this, this or this.

- Poor UI interaction design and usability;

“Any knows how to use 'tplosttrim' and 'igesparam' commands in Test harness related to reading of IGEs files. 'igesparam' is not documented in user's guide too.” Origin.

What can be done?

Below there are some ideas how to make OCCT environment more user-oriented.

Documentation structure

Sticking to the point that as a rule the user appeals to the guidance when he already has a certain question or problem (but not just for fun or read it as a book at night), the documentation structure must correspond to the user’s mentality search model as much as possible. For example, the screenshot below shows how the documentation breaks this rule:

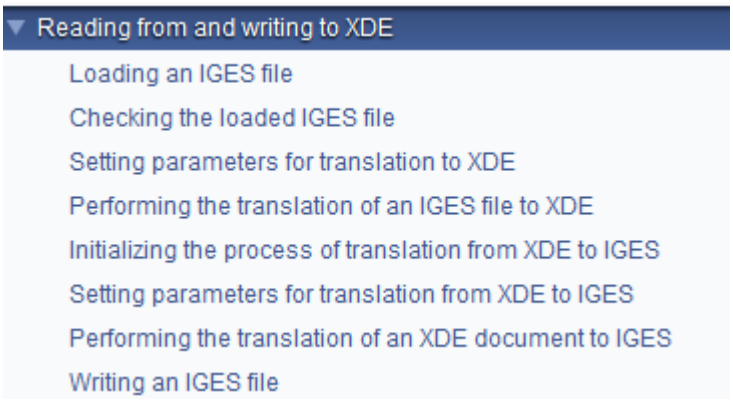
If the user needs to build one of the 3rd-party libraries he keeps in mind "*build-platform-library_name*" chain but not "*build-platform- significance-library_name*". He absolutely does not care if it is mandatory or not, he is not interested in it. What is mandatory for compilation is not mandatory for user. Moreover, in minimal compilation there is nothing mandatory at all (you can have full featured geometric modeling without any additional libraries if Draw is excluded; that's an important thing as OCCT is self-contained in contrast to, for example, ACIS).

Such content structure interrupts the user's workflow – he has to stop, think about that dilemma and likely, look over both items. Moreover, it is unlikely that he is going to search 3rd parties build instructions inside OCCT build instructions as they are independent from OCCT. Moreover, the fact that OCCT build instructions contain 3rd parties build instructions may even make the user to think that before OCCT building he has to build all 3rd parties.

How the documentation can be re-structured to optimize the information search process and highlight the workflow steps is presented in Appendix: Documentation structure.

Some more useful techniques can be applied:

- Avoid too heavy content. If the user is able to browse the article *(its subpart)* by wheeling the mouse 1-2 times do not add more content items. For example, see *Reading from and writing to XDE* where 3-5 sentences of description for each sub-item;



- Avoid too long names. For example, "*Mapping of IGES entities to Open CASCADE Technology shapes*" from *User Guides -> IGES Support -> Reading IGES*;

- Avoid deep nesting, try to keep no more 4 nesting levels;

Documentation articles

Proofreading. First of all, fix inaccurate and invalid information, update not fresh and complete not full information. Just some excerpts:

- *Requirements* chapter does not include Doxygen, Graphviz, Inkscape, PDFLatex tools;

- Invalid links:

- http://sourceforge.net/projects/autoopencas/,
- http://cdn.mathjax.org/mathjax/latest
- http://www.mathjax.org/download/

– *"There are four sample files provided in the directory 'OpenCasCade/ros/samples/ocafsamples'"* – obsolete instructions, no such directory;

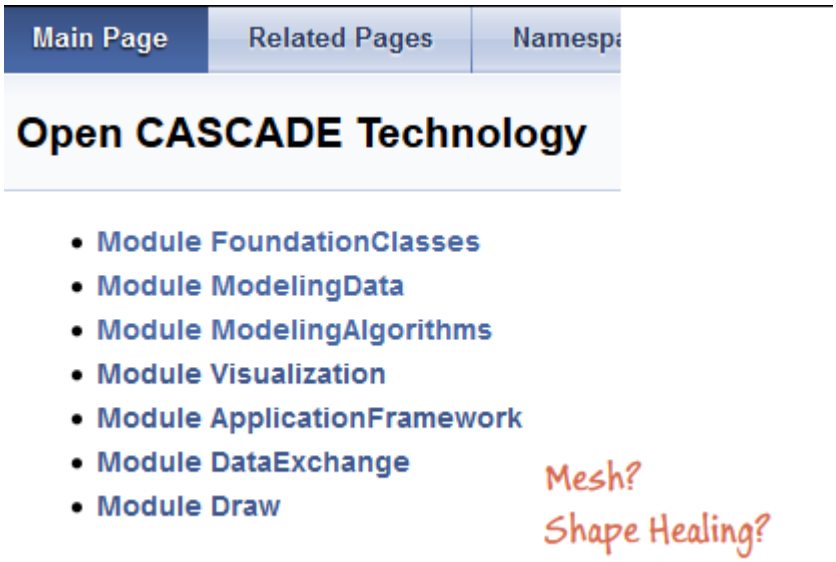
– *"See file* occt.natvis *for example" – where?* not full instructions*;*

– *"OCCT automatic testing system is organized around DRAW Test Harness DRAW Test Harness"* – misprints*;*

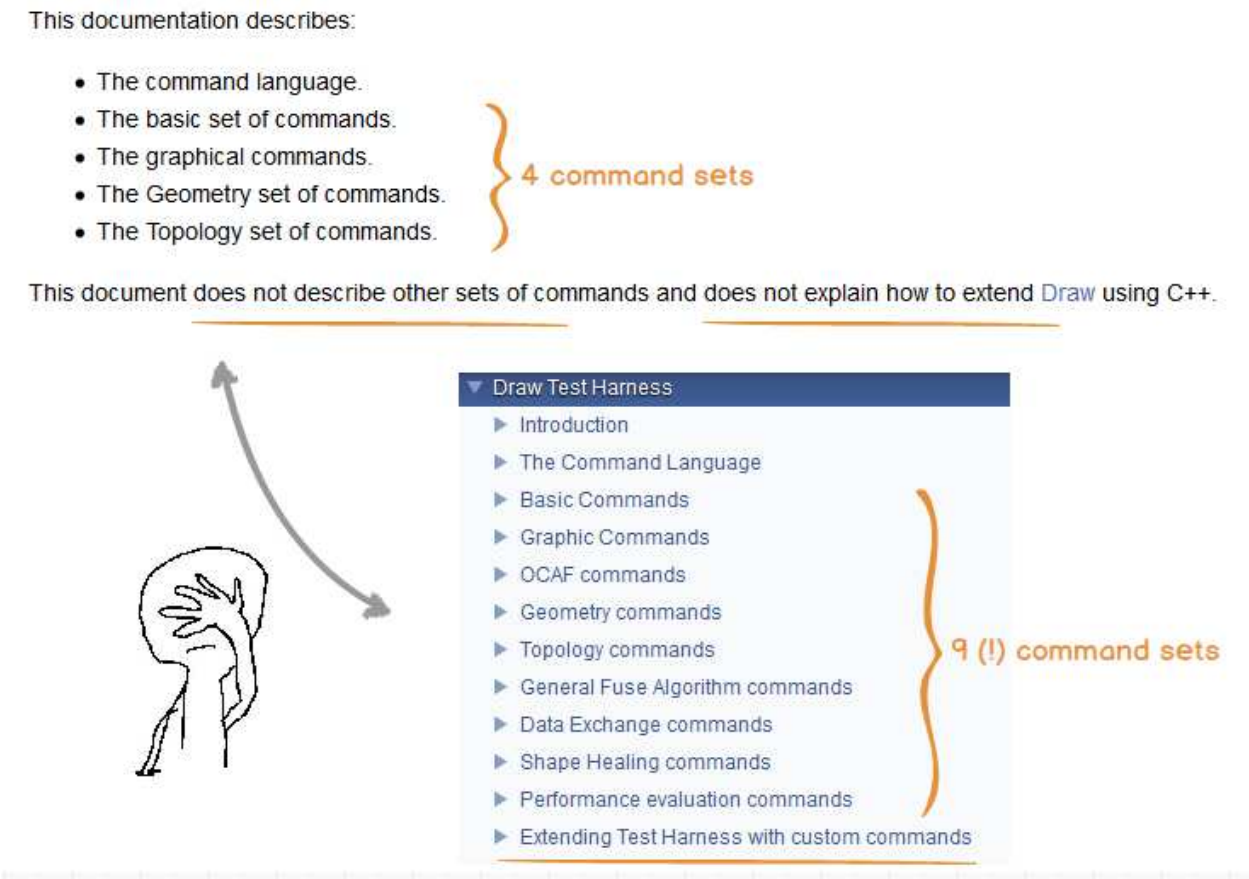
– *"OCAF is much more than just one toolkit among many in the CAS.CADE Object Libraries." "hey, 90s is left";*

– *Minus* characters are used instead of *dashes* mostly. This sounds not important, but it clearly demonstrates the poor quality of our typesetting (the poor culture of working with texts);

– *Technical Overview* introduces *Mesh* and *Shape Healing as* separate OCCT modules while it is not so, at least *Reference* doc does not contain such modules;



– Draw Test Harness documentation contradicts itself;



Simplify User's Guide. *User Guides* articles must follow a common logical structure – outline the basic concepts and introduce API without deep architecture and realization details, without explanation why it was developed by using this tool/approach but not that. Small example, *"TCL is an interpreted command language, not a structured language like C, Pascal, LISP or Basic. It uses a shell similar to that of csh. TCL is, however, easier to use than csh because control structures and procedures are easier to define. As well, because TCL does not assign a process to each command, it is faster than csh."* Does the user really care why OCCT uses TCL instead of csh? Why does he have to read that? All that kind of details must be collected in a separate chapter like *Implementation Guide* for developers or advanced users willing to debug/contribute into OCCT.

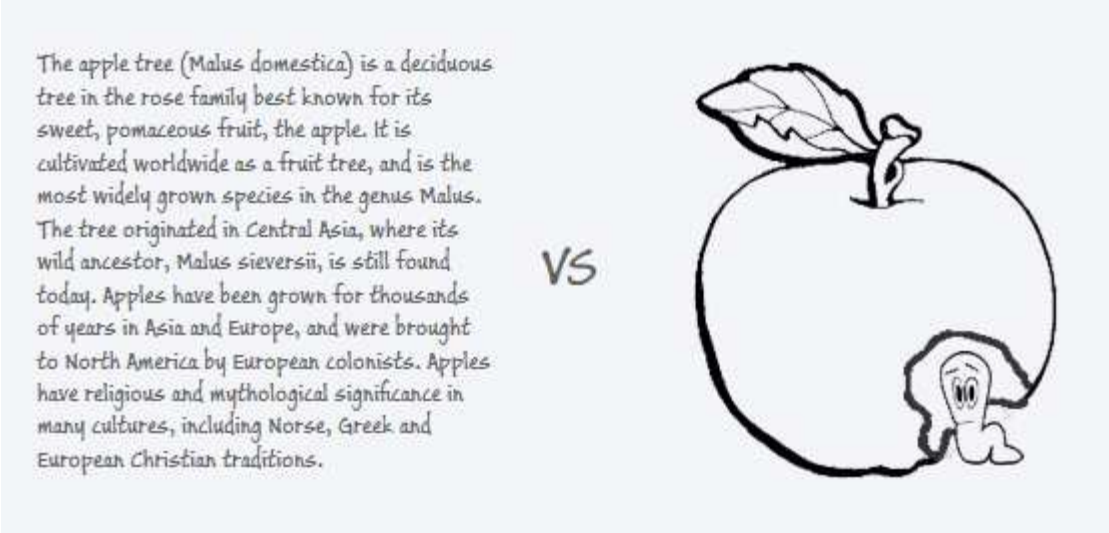
Review User's Guide articles. The teams responsible for OCCT development have to review and update the corresponding chapters of *User Guides*. It also makes sense to involve the persons who take part in OCCT trainings into review process and gain their experience as well.

Purpose at first. The articles dedicated to the things unfamiliar for the user, like OCAF, XDE, TObj, VIS and Draw, must start with perfectly clear description of the reasons why the user may need them, their weaknesses and advantages in comparison with alternatives in case of existence the last. In fact, when the human sees something new the first he thinks is "How can I use it? Do I need it? Is this thing better than mine?" If he does not get the clear answers in short time, he will leave this thing with a high probability. So, to not lose the possible users of OCCT products/components/services precede the articles by information he is looking.

For example, see *TObj package*, in particular, *Introduction* and *Applicability* chapters. Why *Applicability* (the most important part for the person unfamiliar with this package) goes after *Introduction* dedicated to the article's structure? The user has to skip *Introduction*, go to *Applicability*, recognize that he is interested in *TObj*, move back to *Introduction*, read it then skip *Applicability* and go further. Seems as non-optimal way, isn't it?

Newcomer's feedback as a measure of quality. The persons who are new in OCCT (OCCT probationers for example) must be asked to find questions in the documentation at first. Their feedback (whether the person was able to find answers there or comes back with the same questions; whether the questions nature is changed and goes deep etc.) may be considered as a measure of documentation narration quality.

More explanation graphics. It would be valuable to overview articles in terms of determine the places needed more explanation pictures and schemes. In fact, the human analyses graphics better and faster than text. Pictures convey information more efficiently and effectively than words do. So, support the words flow by graphics to improve the clarity.



Update graphics. The existing graphics would be good to refresh and uniform – use one color scheme, fonts, line widths, arrows template; remove visual noise and artifacts; use vector graphics.

Annotate graphics. Annotate the screenshots by a reference to the application where they were done, for example via Advanced Sample Demos or Draw. It may help to extend the audience of tools provided by OCC.

Documentation interactivity

– Auto expand content items when the user selects them;

– The content must indicate the current user's state inside document. It means when the user scrolls down the document the content highlights the item owning the article in the user's focus at that moment;

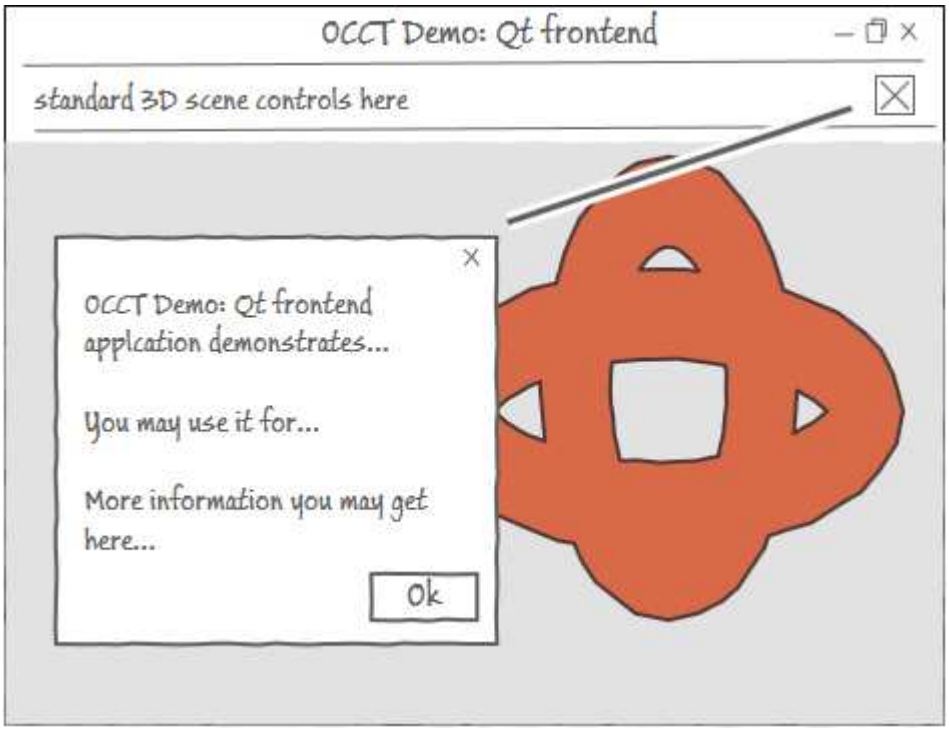
– *Table of content* in the upper right corner is nearly useless navigation control because it disappears after some scroll down movements. Just auto expand the selected item in the main content to provide the whole overview. The same effect with no usability loss;

Samples

OCCT samples should be separated on OCCT *feature* programming samples and samples demonstrating how to *bridge* OCCT with GUI frameworks.

Integration OCCT with GUI frameworks

Qt/C#/Java samples are good launch pads to start application development. By the other words, they are ready-to-use tutorials about OCCT visualization which is a common pain point for many users indeed. Basing on this idea and the idea that OCCT samples 'for free' will be unlikely properly supported and correspond to UI tendencies sometime, it is proposed to minimize a functionality set like here:



The application focuses only on visualization bases. The tested shape is imported in background mode. At first (!) application start the user sees a welcome screen with the introduction info, useful hints and links. No more controls and functionality interrupted the user's workflow.

Such concept eliminates the architecture complexity of sample applications, interaction design mistakes and support involvement. So, the user has both a white sheet for development and

simple tutorial about visualization notions. Other cases can be easily covered by the feature programming samples which the user is free to reuse and embed into his app.

In addition to the existing desktop samples it is proposed to implement a hybrid app sample. For more details, about hybrid applications see Appendix: Hybrid app sample.

Feature Programming Samples

It is proposed to have *feature* programming samples as a part of Draw framework where each sample is a separate Draw command.

This concept allows to keep the programming samples in a consistent mode with updated OCCT sources; use Draw abilities for non-regression testing of sample base; provide the user with a common demo platform for all OCCT modules working on the most popular OS platforms; relieve him from the configuring and compiling samples projects (*for case when OCCT is downloaded as a source package*); integrate new sample cases easily and discover Draw framework to wider range of users.

Moreover, several feature programming samples (for example, one per OCCT module) must be formed as a step-by-step tutorial and posted in the documentation. As “official” forum shows ‘*Make bottle*’ tutorial is in great request.

Each feature programming sample must hint the user about used lib/dll(s) and include files as it also raises a lot of misunderstandings when the users copy/paste a code from documentation or samples into their environment.

About feature programming samples in scope of Draw framework see the next chapter.

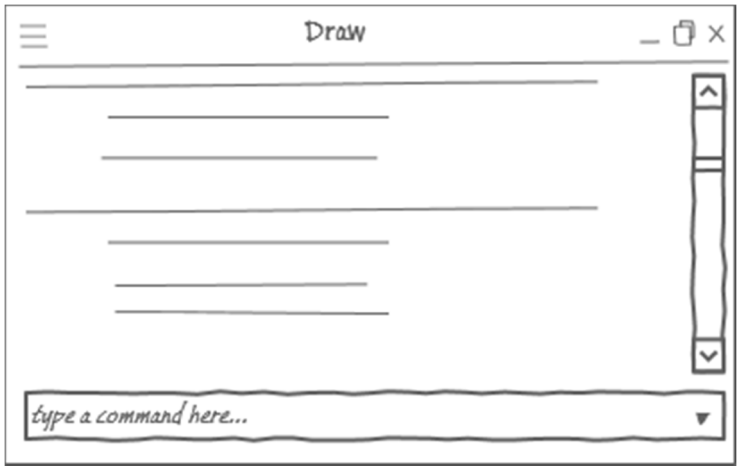
Draw interaction redesign

Draw importance can be hardly overestimated for any development based on OCCT. It has great potential of learning, prototyping and (auto-) testing abilities. It should be as a decencies to use Draw for all who deals with OCCT anyhow. It undeservedly stays in the shadow while it must be OCCT face – welcome entry point. But the welcome entry point must be in a good shape, attractive and trustworthy, so it is proposed to do a ‘face lifting’ procedure for Draw application.

The proposal beneath illustrates how Draw UI can be redesigned to improve its interaction skills, usability and attractiveness. This sketchy overview does not contain the detailed analysis and specification, only observation of the main ideas: keep into account the newcomers needs without annoying the experienced users; provide new features to expand Draw abilities; eliminate interaction design mistakes and provide non-steep learning curve.

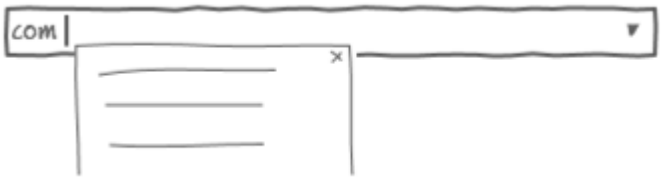
Sketch Overview

The default view:

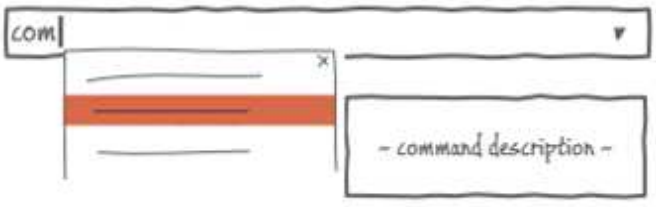


There is a command line to communicate with OCCT, and *History logger* to observe the history of this communication.

Operating with huge number of commands is a complicated job not only for newcomers but for experienced users as well, especially when they have to deal with a new set of commands. So, to help the users the command line must support an intelligent input (*autocompletion* and *autoguess*). When the user types the command name he sees the list of commands started the same and, below, the commands which are similar to the typed one. For example, if the user writes *step* he will see all commands related to STEP data interoperability: *ReadStep*, *stepwrite*, *stepread*, etc.

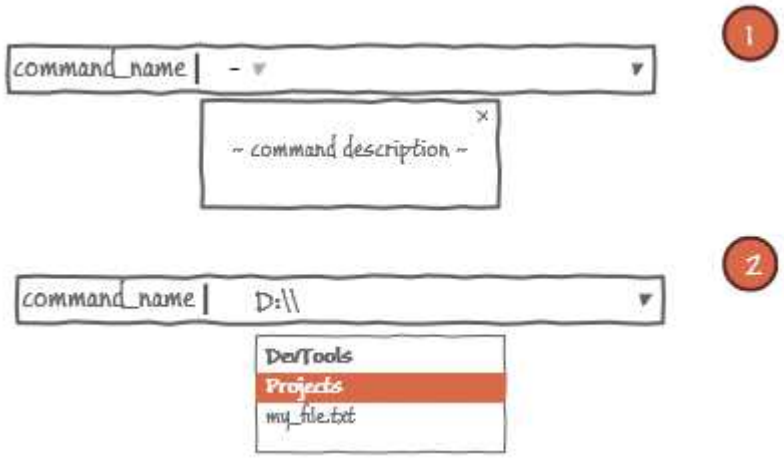


The user is free to consult all them and choose the desired one. When the command is focused the user sees a command detailed description – signature, what for and how to use.



The users experienced in Draw can switch off the tips using a special control or pressing the corresponding shortcuts.

More hints:



Case 1. The down arrow is shown when the user types '-' symbol preceding a key value. To see the available keys the user presses *Ctrl+Down,* for instance.

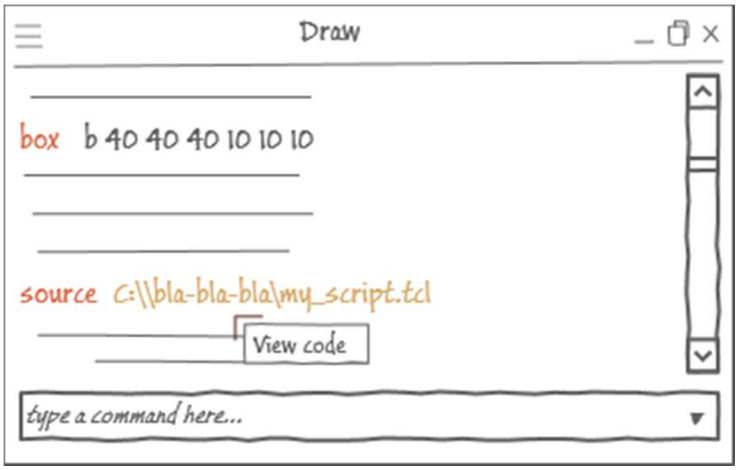
Case 2. When the user types a path to directory or file he sees the content of currently typed directory as a hint under the command line. The hint control is interactive – support navigation via mouse scrolling or keyboard arrows, *Enter* to accept a choice. If the user types *cd* command or shortcut *Ctrl+O,* for example, he sees the available hard and removable disk drivers.

The user may ignore the tips and type all that needed manually.

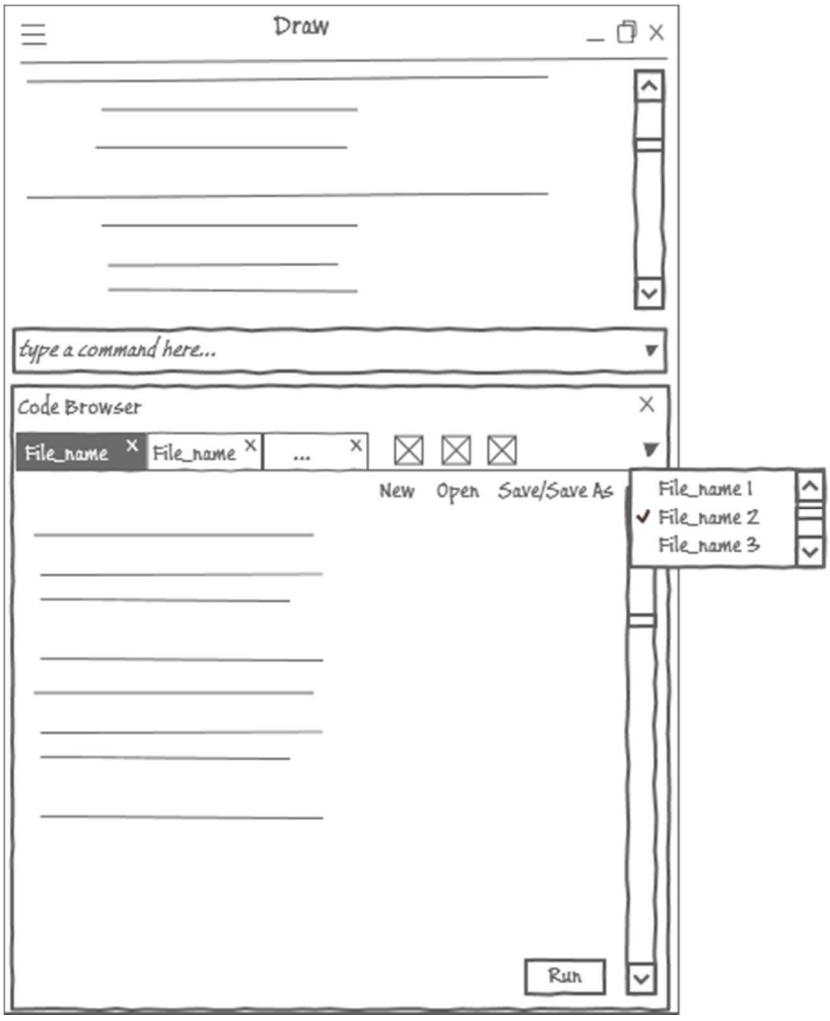
'*Up arrow*' key (or another accustomed combination for command line apps) repeats the previously entered commands.

The entered commands with the output messages the user can observe in *History logger*. The command itself and its output must visually distinguish using different colors or fonts or indents for example.

The Draw commands and Tcl scripts as parameter are marked by a special color/font/size, etc. and sensitive to the mouse pointer, like a push button behavior, to stress their interactivity – option to overview code behind. To view Tcl script code the user clicks a corresponding context menu (or flight-mode menu) item and sees the code in *Code Browser*.



The *Code Browser* is used as Tcl scripts editor. The user can create new, open, modify, save and run the Tcl scripts. Clicking *Run* button means calling *source* command, so the history logger is populated with a corresponding entry.



The code of Draw command can be opened in *Code Editor* too but it is not editable, only overview is allowed. The code listing is scrolled to the API function corresponding to the currently overviewed Draw command.

Drag-&-Drop support is as an alternative to *View code* calling. Also it would be nice to support cross-processes drag-&-drop: drag Draw command name and drop it under any code editor icon in the task bar to open the corresponding source file in this editor.

Clicking on Draw command, Tcl script or directory/file path opens a file explorer a destination where this file/script is located.

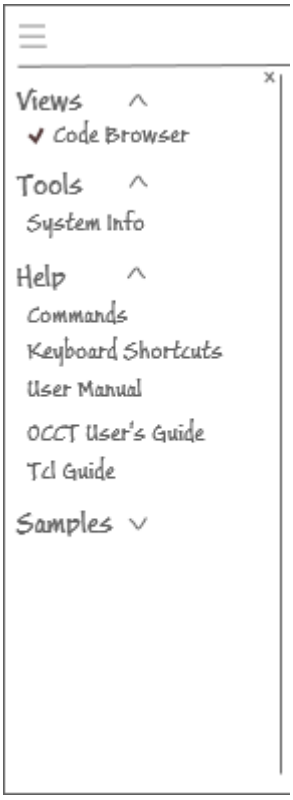
The created views are also the tabbed controls:

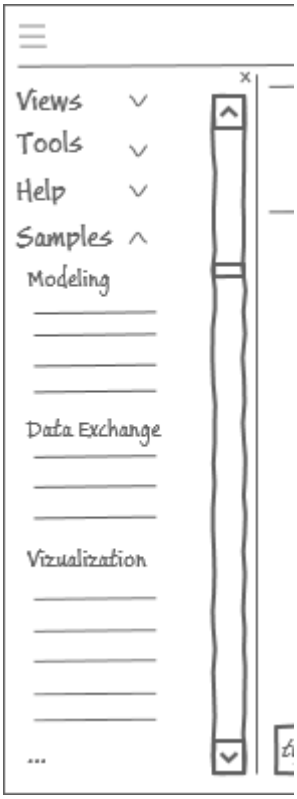
As the view names are not informative for the user – he can hardly distinguish axo, perspective, 3D views etc. basing on their names only, the tab headers may visually indicate the type of view contained in this tab or show the detailed info when the header is under mouse cursor.

The user can undock a tab to the float state if he needs to compare the views scenes.

All type of views must support a direct interactivity for rotation, zooming, panning, selection etc. actions. The navigation via command line can be useful only for a precise camera positioning, in other cases it irritates the user and slows down his performance in times.

The main menu looks as follows:





All items are collapsed by default.

Samples menu item contains a set of demo Tcl scripts lighting all OCCT modules. As soon as the user clicks one of the sample name *Code Browser* opens the script in a separate tab and executes it with journaling the corresponding command in the *History logger*. The menu stays open until the user closes it manually. For other menu items it closes automatically.

If sample shows not only viewer but more data controls, for example OCAF tree, these controls are rendered in the view scene directly (if technically hard to achieve, place in one tab with viewer). All controls having command alternatives, for example, switching shading/wireframe mode, *History logger* journals them when they are activated.

Appendix: Documentation structure

Introduction		
	~~ content ~~	
Requirements		
	Windows	
	Linux	
	Mac OS X	
	Android	
	Graphic card	
Building, Installation, Upgrade		
	Environment variables	
	Building OCCT from sources	
		Cross-platform. CMake
		Windows. MS Visual Studio
		Linux. Autotools
		Mac OS X. Code::Blocks
		Mac OS X. Xcode
		Android. CMake with ADT
	Building 3rd parties from sources	
		Windows
		Linux
	Install OCCT on Windows	
	Documentation System	
	Upgrade to new OCCT version	
User's Guide		
	General Information	
		~~ content ~~
	Foundations	
		~~ content ~~
	Modeling Data	
		~~ content ~~
	Modeling Algorithms	
		~~ content ~~
	Visualization	
		~~ content ~~
		VIS component
	Data Exchange	
		Overview
		BRep

		~~ content ~~
		STEP
		~~ content ~~
		IGES
		~~ content ~~
		Shape Healing
		~~ content ~~
		Extended Data Exchange (XDE)
		~~ content ~~
	Application Framework (OCAF)	
		~~ content ~~
	Draw Framework	
		~~ content ~~
Development Board		
	Implementation Guide	
		~~ content ~~
	Enable extended debug messages	
	JIT debugger on exception	
	Functions for debugger	
	Visual Studio debugger tricks	
	Performance measurement tools	
	BOP self-diagnostics	
Contribution		
	Git	
		Installing Tools
		~~ content ~~
		Get access to the repository
		Clone official repository
		General workflow with repository
		~~ content ~~
		Propagate fix between OCCT versions
	Issue Tracker System	
		Access level
		General workflow
		Issues attributes
		Documentation evaluation
	Coding Rules	
		Documentation conventions
		Naming conventions
		Formatting conventions
		General C/C++ rules

		Application design
		Portability issues
		Stability issues
		Performance issues
		Draw commands
		Samples of well-documented code
Draw Framework		
	Introduction	
	Command Language	
		~~ List of Commands ~~
	Basic Commands	
		~~ List of Commands ~~
	Geometry Commands	
		~~ List of Commands ~~
	Topology Commands	
		~~ List of Commands ~~
		General Fuse Commands
	Data Exchange Commands	
		~~ List of Commands ~~
		Shape Healing Commands
	Visualization Commands	
		~~ List of Commands ~~
	OCAF Commands	
		~~ List of Commands ~~
	Performance Evaluation Commands	
		~~ List of Commands ~~
	Extending with Custom Commands	
	Automated Testing	
Samples and Tutorials		
	Integration with GUI frameworks	
		OCCT and WinForms using C#
		OCCT and WPF using C#
		OCCT and QML for Linux
		OCCT and Qt for Android
		OCCT and Java for Android
	Modeling	
	Data Exchange	
	OCAF	
	Visualization	

Appendix: Hybrid app sample

Last time the web technologies are taking more and more popularity for building desktop applications. Such applications are also known as hybrid or web-desktop apps. Technically, that means a fully functional web browser embedded inside a local application shell. Content is served up fresh from the standard web servers and stored locally in standard files. It combines elements both native and web applications.

Hybrid apps are taking advantages of web technologies for UI creating in particular, using HTML5, CSS and JavaScript strong weights. This approach allows providing a consistent user experience on all operating systems and devices preventing native specific rendering; allows the integration of tons of web services and technologies; be in trend with the graphic design tendencies like flat animated design (so-called metro or win8 style) or propagate company/brand style features into the application design. Thanks to the great experience of web-developers there are more opportunities to create more human-oriented, non-standard and interesting interaction solutions than pure native frameworks provide.

Being abstracted from the underlying operating system, the browser embedded in the client allows implementing once and having it available across Windows, Mac and Linux.

For sure using web-technologies for desktops is not a panacea and has disadvantages as well like performance decrease, more amount of CPU and RAM usage while starting and rendering, more amount of executable package, need to be skilled in a great number of web technologies, etc. More info can be easily found in www and critically investigated as everything is changing.

There are many technical options for implementing a hybrid app. One of the most famous is Adobe Integrated Runtime (AIR) developed by Adobe Company. The company calls it a "*cross-operating system runtime that lets developers combine HTML, Ajax, Adobe Flash, and Flex technologies to deploy rich Internet applications (RIAs) on the desktop.*" AIR is a foundation for Adobe Photoshop, Illustrator, LightRoom, InDesign applications development. AIR is a free product but does not support Linux system and in most oriented for mobile devices. Qt WebKit is also one of the options but it seems not for free.

Below the most popular free frameworks supporting Microsoft, Linux and Mac OS X platforms:

- NW.js, formerly node-webkit, is one of the most popular and mature options today. It is an app runtime based on Chromium and node.js. It offers the best assortment of features, great community support, and easily searchable online questions. NW.js lets you call all Node.js modules directly from DOM and enables a new way of writing applications with all Web technologies. It is created and developed in the Intel Open Source Technology Center.
- Electron.js, the platform initially developed for GitHub's Atom editor. Since that time it has been used to create applications by companies like Microsoft (Visual Studio Code), Facebook, Slack, and Docker. It solves the same problems as NW.js but it does it somewhat differently. It does not integrate Chromium, but instead uses libchromiumcontent to access Chromium's Content API.

So, taking into account the growing interest to the hybrid apps it would be farseeing decision and big move towards OCCT community to supplement OCCT demo samples with a web-based one. OCE community experience can be considered as a start point for that. Moreover, it is a good chance to improve and extend skills of OCC GUI expertise.

Appendix: OCCT related resources

- http://www.opencascade.com/forums
- http://tracker.dev.opencascade.org
- https://groups.google.com/forum/#!forum/oce-dev
- http://diyhpl.us/wiki/cad/opencascade/
- http://www.cppblog.com/eryar/archive/2013/08/18/202617.html
- http://opencascade.wikidot.com/start
- http://opencascade.wikidot.com/tutorial-started-by-semikin:drawexe-very-simple-examples-no