# UD-Viz : Urban Data Vizualisation

UD-Viz is a JavaScript library based on [iTowns](https://github.com/itowns/itowns), using [npm](https://www.npmjs.com/) and [published on the npm package repository](https://www.npmjs.com/package/ud-viz), allowing to visualize, analyse and interact with urban data.

A tutorial of the game engine can be found [here](../../UD-Viz/Doc/Devel/LocalGameTutorial.md)

### Install node/npm

For the npm installation refer [here](https://github.com/VCityTeam/UD-SV/blob/master/Tools/ToolNpm.md)

UD-Viz has been reported to work with versions:

* node version 16 (16.13.2)
* npm version: 6.X, 7.X. and 8.X

### Installing the UD-Viz library per se

git clone https://github.com/VCityTeam/UD-Viz.git  
cd UD-Viz  
npm install

### Running the examples

cd PATH\_TO\_UD-Viz  
npm run build  
cd /  
git clone https://github.com/VCityTeam/UD-SimpleServer  
cd UD-SimpleServer  
npm install  
node index.js PATH\_TO\_UD-Viz 8000

* [UD-Viz-Template](https://github.com/VCityTeam/UD-Viz-Template) (demonstration) application,
* online demos are [available here](https://projet.liris.cnrs.fr/vcity/demos/)

## Developers

### Pre-requisite

Developing UD-Viz requires some knowledge about [JS](https://github.com/VCityTeam/UD-SV/blob/master/UD-Doc/Devel/ToolJavaScript.md), [node.js](https://en.wikipedia.org/wiki/Node.js), [npm](https://en.wikipedia.org/wiki/Npm_(software)) and [three.js](https://threejs.org/).

### Developers documentation

After generation, the [browsable documentation](https://vcityteam.github.io/UD-Viz/html/index.html) is [stored within this repository](../../UD-Viz/docs/html/index.html).

Refer to this [README](../../UD-Viz/docs/README.md) to re-generate it.

### Coding style

The JavaScript filees coding style is defined with [eslint](https://eslint.org/) through the [.eslintrc.js configuration file](../../UD-Viz/.eslintrc.js). It can be checked (e.g. prior to a commit) with the npm run eslint command. Notice that UD-Viz coding style uses a unix linebreak-style (aka LF as newline character).

### Tips for Windows developers

As configured, the coding style requires a Linux style newline characters which might be overwritten in Windows environments (both by git and/or your editor) to become CRLF. When such changes happen eslint will warn about “incorrect” newline characters (which can always be fixed with npm run eslint -- --fix but this process quickly gets painfull). In order to avoid such difficulties, the [recommended pratice](https://stackoverflow.com/questions/1967370/git-replacing-lf-with-crlf) consists in 1. setting git’s core.autocrlf to false (e.g. with git config --global core.autocrlf false) 2. configure your editor/IDE to use Unix-style endings

#### Notes for VSCode users

When using [Visual Studio Code](https://code.visualstudio.com/), you can [install the eslint extension](https://www.digitalocean.com/community/tutorials/linting-and-formatting-with-eslint-in-vs-code) allows you e.g. to automatically fix the coding style e.g. [when saving a file](https://www.digitalocean.com/community/tutorials/linting-and-formatting-with-eslint-in-vs-code) .

### Prior to PR-submission 1: assert coding style and build

Before pushing (git push) to the origin repository please make sure to run

npm run travis

(or equivalently npm eslint and npm run build) in order to assert that the coding style is correct (eslint) and that bundle (production) build (webpack) is still effective. When failing to do so the CI won’t check.

Note that when commiting (git commit) you should make sure to provide representative messages because commit messages end-up collected in the PR message and eventually release explanations.

### Prior to PR-submission 2: functionnal testing

Before submitting a pull request, and because [UD-Viz still misses some tests](https://github.com/VCityTeam/UD-SV/issues/34), **non regession testing must be done manually**. A developlper must thus at leas) check that all the [demo examples](https://github.com/VCityTeam/UD-Viz/tree/master/examples) (refer to [their online deployment](https://ud-viz.vcityliris.data.alpha.grandlyon.com/)) are still effective.

### Submitting a Pull Request

When creating a PR (Pull Request) make sure to provide a correct description

## Sources directory layout (organizational principles)

Definitions: - [Component](https://en.wikipedia.org/wiki/Component-based_software_engineering): everything thats is necessary to execute only one aspect of a desired functionality (see also [module](https://en.wikipedia.org/wiki/Modular_programming)). - Extension: a component depending on a [web service](https://github.com/VCityTeam/UD-Viz/blob/master/src/Widgets/Extensions/Geocoding/services/GeocodingService.js" \l "L2) in order to be functionnal. - Widget ([web widget](https://en.wikipedia.org/wiki/Web_widget)): an embedded element of a host web page but which is substantially independent of the host page (having limited or no interaction with the host) - [Template](https://en.wikipedia.org/wiki/Template_method_pattern): a class build on sibling sub-directories (Game, Widgets, Views) components and proposing an application model - View: decorated/enhanced [iTowns Views](https://www.itowns-project.org/itowns/docs/" \l "api/View/View)

UD-Viz (repo)  
├── src # All the js sources of UD-Viz JS library  
| ├── Components # A set of components used by sub-directories at this level  
| ├── Templates # Classes builded with other sub-directory (Game, Widgets, Views) to propose application model  
| ├── Views # Classes of 3D views encapsulating the itowns view  
| ├── Game # A sub-directory offering game engine functionnality  
| | ├── Shared # code that can be executed both and client and server side to simulate a world  
| |   
| └── Widgets # A sub-directory gathering a set web web widgets (UI)   
| ├── Widget\_1  
| ├── Widget\_2  
| ├── ...  
| └── Extensions # Widgets depending on an external web service   
├── ...  
└── webpack.js

Notes: \* The position of a specific component in the sub-folder hierarchy reflects how it is shared/re-used by sub-directories. For example if a given component is only used by a single widget, then it gets defined within that widget folder. But when another component usage is shared by two widgets then its definition directory gets promoted at the level of the two widgets.