Open City Toolkit

technical manual

Jan BEHRENS and BUGYA Titusz
CityScienceLab -- HafenCity University Hamburg

version 0.17
last modification:
BUGYA Titusz, 2020. May. 19.

Under construction: it is not a final version, therefore the text, the layout and the content is not reviewed or revised.

Content

Software requirements and Installation

Cityapp system is a frame system, connecting several external softwares and allowing a coherent user dialogue. The system is flexible, and allow to implement almost any GIS solution, but because of this property, the installation may a bit longer process, installing external components and making some settings too.

External components have to be installed separately. It is because although our recommendation is a Debian-based Linux, you may select your preferred distribution too.

There are no cross-dependencies in the system and between any external components, therefore the installation order has no significante. External components are:

1. GeoServer
2. GRASS GIS
3. Gnuplot
4. Node.js
5. enscript
6. ghostscript
7. Cityapp

There are many other external components too, but those are default parts of a standard Linux distribution (such as *sed*, *cut*, *head*, *tail*, *grep*, *stat*, *inotifywait*).

Operating system

A Linux system is required. Neither the kernel version, nor the flavour has any significance. But, regarding to the web-based approach and server-client architecture, a modern, up-to-date Linux environment is highly recommended.

Installation directory

It is highly recommended to install your Cityapp copy into a home directory of a dedicated user created for this purpose (for example: cityapp_user -- in this manual this will used as username of the dedicated user). This is a simple user directory without any specific properties. It is only to clearly separate the data stored in the cityapp directory and to allow a clear data management through file permissions. In this manual it is expected that the dedicated home directory is */home/cityapp_user*. Of course, any other name is allowed, it is only an example.

With Docker

You can quickly set up a running system via Docker – download the contents of this repository, change to its root directory and build the Docker image:

```
docker build -t cityapp .
```

Now start a container using the newly created image:

```
docker run -ti \
 -v ~/<local path to>/geoserver_data:/root/cityapp/geoserver_data \
 -v ~/<local path to>/grass:/root/cityapp/grass \
 -p 3000:3000 \
 -p 8080:8080 \
 --name cityapp_1 \
 cityapp
```

The app will run on *http://localhost:3000*, and GeoServer will be available at http://localhost:8080/geoserver/.

The geoserver_data and grass directories are mounted as volumes into the container, in order to make their contents persistent.

Using Docker, permissions may request an extra care: it may happened, that the system or some components works not properly In such case first check file permissions.

Without Docker

This way is a bit longer, but allows more flexibility.

Download the tgz file of current version of cityapp, copy in users's root directory and uncompress:

```
tar -xzf cityapp_1.0.tgz
```

This will the base directory of further operations.

Without docker all external components has to be installed separately.

External Components

Except Geoserver, external components has to install separately in Docker mode too.

Geoserver

Use a current stable version of GeoServer, at least version 2.15. The expected path of the GeoServer data directory on your system is /usr/share/geoserver/data_dir/data, therefore our recommendation is to install GeoServer into */usr/share/geoserver*.

1. Download a platform independent, fresh, binary version of geoserver from *geoserver.org*. It is a zipped file, ready to run after unzip.
2. Create a geoserver directory:
   ```
   sudo mkdir /usr/share/geoserver
   ```
3. Allow cityapp user to acces that directory. First change owner, and set cityapp user as owner of geoserver directory:
   ```
   sudo chown cityapp_user /usr/share/geoserver
   ```
4. Copy zipped geoserver to this new directory
   ```
   cp Download/geoserver-2.17.0-bin\(1\).zip /usr/share/geoserver
   ```
5. Unzip:

```
cd /usr/share/geoserver
unzip ./geoserver-2.17.0-bin\(1\).zip
```

6. Set permissions of startup and shutdown script:

   ```
   chmod 744 /usr/share/geoserver/bin/startup.sh
   chmod 744 /usr/share/geoserver/bin/shutdown.sh
   ```

   Geoserver is now ready to run, but not yet ready to use with Cityapp.

7. Cityapp stylesheets are in CSS format, therefore it is requested to use a css extension in Geoserver. For this end first download the extension from the Geoserver site: *https://docs.geoserver.org/latest/en/user/styling/css/install.html*

8. Copy the downloaded zip file to: /usr/share/geoserver/webapps/geoserver/WEB-INF/lib:

   ```
   cp ~/Download/geoserver-2.16.0-css-plugin.zip /usr/share/geoserver/
   webapps/geoserver/WEB-INF/lib/
   ```

9. Unzip copied file:

   ```
   cd /usr/share/geoserver/webapps/geoserver/WEB-INF/lib/
   unzip /usr/share/geoserver/webapps/geoserver/WEB-INF/lib/geoserver-
   2.16.0-css-plugin.zip
   ```

   Geoserver is now able to interpret css stylesheets of Cityapp maps.

10. After installing Geoserver, it is requested to create /usr/share/geoserver/data_dir/data/cityapp link, pointing to ~cityapp/geoserver_data, because Grass will export the results to that directory. :

    ```
    ln -s ~/cityapp/geoserver_data /usr/share/geoserver/data_dir/data/
    cityapp
    ```

    Maps, generated by OCTK request an adequate symbology too, stored as “workspace” in Geoserver’s “workspaces” directory. This new symbology is prepared for OCTK, therefore can’t be found in the default installation of Geoserver. For this end, there is a “~/cityapp/geoserver_workspaces” directory to store these settings in subdirectories named as „raster” and vector”. Those are predefined Geoserver workspaces, and have to link them to workspaces directory of Geoserver.

    ```
    ln -s ~/cityapp/geoserver_workspaces/raster /usr/share/geoserver/
    data_dir/workspaces/raster
    ln -s ~/cityapp/geoserver_workspaces/vector /usr/share/geoserver/
    data_dir/workspaces/vector
    ```

Grass GIS

GRASS GIS is the core component of the backend: a highly developed generic purpose, cross-platform GIS system. It is required to install GRASS GIS version 7.1 or newer. The GRASS GIS installation path has no importance. Download GRASS GIS from: https://grass.osgeo.org/

On a Debian-based system:

```
apt-get install grass
```

For other systems and for further info, please visit: https://grasswiki.osgeo.org/wiki/Installation_Guide

Gnuplot

Gnuplot is used to create various data visualizations and export them into PNG format, allowing the browser to display them. Gnuplot is a default component of most Linux distributions, but if your installed system does not contain it, it can be downloaded from http://www.gnuplot.info/.

On a Debian-based system:

apt-get install gnuplot

Node.js

Node.js is a crucial component to run the frontend, therefore it has to be installed properly. Version 12 or higher is required. The recommnded way of installing Node.js in Linux is via NodeSource (follow the instructions for your distribution).

On a Debian system:

```
curl -sL https://deb.nodesource.com/setup_12.x | bash -
apt-get install -y nodejs
```

Now change to the webapp directory:

```
cd ~/cityapp/webapp
```

and, before you start the server for the first time, you must run:

```
npm install
```

Enscript

Enscript is a command line tool to convert text files to PostScript, HTML, RTF, ANSI. It is used to create a statistic output. If not installed, then:

```
apt-get install enscript.
```

Ghostscript

Ghostscript is a package containing tools to manage postscript files -- including a ps to pdf converter too. If not installed, then:

```
apt-get install enscript ghostscript .
```

2 Running Cityapp

Before the first launch of the application, please run in a console:

```
~/cityapp/scripts/modules/checker/checker.sh
```

this is to check if all requested external components are available. When a component is not found, report is generated in ~/cityapp/error_log file. It contains the name of missing requirements. Before continue, install them.

2.1 Simple run

To start the application, run:

```
~/cityapp/scripts/base/ca_starter.sh
```

It launch the application, the scheduler, the Geoserver and the node.js too (for front-end communication)

To stop it:

```
~/cityapp/scripts/base/ca_shutdown.sh
```

Thi will stop (shut down) all components and modules, started by ca_starter.sh or by Cityapp, therefore the user launched Cityapp module will stopped too.

Starting components separately

It may necessary for error management, to check/monitor if a component works properly or not. In this case do not launch ca_starter.sh, but launch its components separately, in four separated console (a sipmle *xterm*, for example):

1. Start the signal generator:

   ```
   ~/cititapp/scripts/base/cityapp_onesecond.sh
   ```

2. Start the modul launching manager:

   ```
   ~/cititapp/scripts/base/cityapp_module_launcher.sh
   ```

3. Start Geoserver:

   ```
   /usr/share/geoserver/bin/startup.sh
   ```

   Geoserver administration interface is now accessible thru the port 8080. To acces it, open a browser and type the URL:

   ```
   http://localhost:8080/geoserver/web/
   ```

4. Start Node.js

   ```
   cd ~/cityapp/webapp
   node app.js
   ```

5. Now open a browser at http://localhost:3000 and you should see the app's user interface.

   If you want to use the node.js server in production, it is recommended to use the process manager *pm2*. To install it, run:

```
sudo npm install -g pm2
```

To start a process for Cityapp:

```
pm2 start ~/cityapp/webapp/app.js --name=cityapp
```

System design

The idea of the software is relatively simple. There is a frontend communicating to the user and displaying results as maps, data, and graphs, and there is a backend, processing calculations, queries, data storage, and serving maps. In between there is a software layer as interface allowing the web-based user interface to communicate with the GIS backend.

The frontend is a HTML and JavaScript-based dashboard, running in a simple web browser (such as Firefox), using the Leaflet library to display various maps.

The backend has two pillars. GeoServer is to serve maps, and GRASS GIS is to process calculations. GRASS GIS map outputs are exported into the GeoServer's data directory, and GeoServer only serves maps from the GRASS GIS. Allowing dialogue between the bckend and frontend and uploading data for analysis, Node.js is used as an interface layer. Thus, all requests, input data and dialogue messages are managed by the Node.js server.

Data are basically stored in GRASS GIS mapsets, allowing direct access for calculations.

This architecture makes the system flexible: since front-end and back-end are separated from each other, threfore any of them can changed (or completly replaced) without any changes in the other. The only thread between the two parts, the user dialogue as messages. Messages are generated by back-end and transferred by node.js to front-end. When the user select from a list, type an input data, or upload a file, that is also transferred by node.js to back-end.

Each module the user may launch, has a step-by-step approach: user is guided thru the whole process (selecting options, giving input parameters, upload files), where the steps are well-defined and only a user-level knowledge is expected (for example: user is never asked about projection system or character encoding). In practice, this approach means, that user only has to select a predefined function from a menu, and the selected module will ask the requested parameters and/files one-by-one. There are no generic menus allowing to start a dedicated GIS function, there are only menus to start a module. I this termiology, module is a separate script dedicated to solve a predefined, complex task, requesting only user-competence parameters, therefore:

- only EPSG 4326 projection is supported.
- only UTF-8 character encoding is supported.
- only osm (Open street map) files are accepted to create a location.
- only a very limited set of fileformats are supported: geotiff, geojson, geopackage and osm.

Directories and files

~/cityapp

folder contains all necessary files, directories, including and data files and folders too. System files and folders of external components are not under *~/cityapp*.

~/cityapp/data_from_browser

Requests and data files from user (from user's browser) are placed here. Those are : request files , uploaded maps, leave_session file, EXIT file.

~/cityapp/data_to_client

System messages (message files and info files) from cityapp to user, are placed in this folder.

~/cityapp/geoserver_data

This folder serves as data folder of Geoserver. /usr/share/geoserver/data_dir/data is a link, pointing to this directory.

~/cityapp/geoserver_workspaces

This is a symbology folder of Geoserver. /usr/share/geoserver/data_dir/workspaces is a link, pointing to this directory.

~/cityapp/grass

By Grass GIS's terminology, it is the database directory of Grass GIS. This database has a single location.

- *global*

  By Grass GIS's terminology, it is a Grass GIS location, the only in its database.

  - *PERMANENT*

    By Grass GIS's terminology, it is the main (root) mapset of "global" location. Projection data, unit declaration and base maps are stored in subdirectories of this directory. This mapset is automatically created when a new location is added. Selections are also stored here.

  - *module_x*

    Directories with this name pattern (where "*x"* is a number, such as *module_1*, *module_2*, etc) are mapsets of "global" location and, automatically created and managed by modules with the same name. When a module has to perform a GIS process, it is normally taken in its own mapset (for example, *module_1* performs processes on maps of mapset *module_1*).

- *skel*

  A template directory -- this directory is used when a Cityapp module create a new mapset in *~/cityapp/grass/global* directory. Is such case this directory is copied and renamed to the module's name.

- *skel_permanent*

  A template directory -- this is only used when a new PERMANENT mapset is created. In this case the directory is copied under "*global*" directory and renamed to PERMANENT

~/cityapp/saved_results

This is to store outputs in pdf format. Modules of Cityapp outputs a pfd file too, and those files are placed here. These pdf files has a unique name, with the pattern:

module_name_current_date_current_time.pdf

~/cityapp/scripts

All Cityapp scripts, variables and messages are stored here.

- *base*

  Scripts to start and stop the system, launch modules, and sending signal (see in chapter "Running Cityapp")

- *modules*

  All cityapp modules are located here. Modules with name pattern "module_x", where x is a number, are user modules: those are to analyze or query maps or processing calculations on demand of user. Other modules are accesoire modules, or system-wide modules. For further information, refer to chapter "Modules".

  - *add_location*

  - *add_map*

  - *checker*

  - *make_selection*

  - *process_check*

- *resolution_setting*

- *running_check*

- *select_lang*

- *module_1*

- *module_2*

-*shared*

A directory to store shared files: dialogue messages, system messages, temp files, variables and constant values.

- *messages*

Containing sudirectories for messages in various languages. Each language has a separate directory to store message files. English is default, others are optional.

- *en*

Default (english) user- and system messages. Each module has a messages file, with the same name the module has. Files of this directory has the same name

- *variables*

System wide and/or more-than-one-module-related variables, temp files.

- *webapp*

Core directory of node.js environment

- *node_modules*

node.js modules only, no user-related files

- *public*

Directory for decorations

- *images*

Logo of contributors and north arrow file in png format

- *views*

This directory is to store frontend files and to set its behaving.

- *launch*

Base look of the front-end and ith behaving

- *map*

Leaflet settings for the front-end (the central, square-shaped part of the front-end)

Back-end

Grass GIS

The role of GRASS GIS (“Grass”) is crucial.

Grass is a separate, open source GIS tool, therefore it has to be installed separately (see “*Software requirements and Installation*”). When Grass is properly installed, there are three ways to run:

- *GUI*: typing a simple command “grass” in a console, or clicking the icon in the launcher menu will launch Grass in GUI mode. In GUI mode, functions are avilables thru a menu system. However, a Grass command line is also available, allowing to run commands directly in a Grass console.
- *Text*: Only a text-based console (command line) is availabe as user interface. It is recommended only for advanced Grass users.
- *Bypass mode*: without an exlicit launch of Grass, user may launch a grass command with its parameters from a simple console or from a script. This mode allows a very flexible use of Grass, and this is the way Cityapp uses Grass’s capabilities.

Backend of Open City Toolkit use Grass in this last mentioned mode, therefore Grass is running only for request, processig only one or a few request, then exit. Sice Grass installation includes the GUI too, it is possible to lauch Grass in GUI mode -- it is helpful, when you want to know more (or learn) about the Grass. Durig a normal use of OCTK, users and administrators will not directly meet Grass, and will not run that separately, therefore it is not necessary to kow more about this GIS system. But, when the reader want to develop a new module, or modify an already existing, then it is higly requested to learn more about Grass. For a detailed manual, tutorials, help and trainig dataset go to grass.osgeo.org. Grass uses ~/cityapp/grass directory, therefore it is a mandatory directory: do not remove. Directory has subdirectories:

- global -- Grass location folder, containing raster and vector maps, and attribute data. This directory and data files are automatically managed by Grass. Initially, this directory is not exist: it is created by “Select location” modue (cityapp_location_selector.sh), using the user-selected OSM data, and “skel_permanent” directory as pattern. Without this directory Grass can not to work, therefore an error message will sent (for example, by module_1: “No valid location found. Run Location selector to create a valid location. ”). Therefore, when this directory is removed, run cityapp_location_selector.sh again.
- skel -- This directory contains initial data and files to create a new Grass mapset (see Grass manual), on demand of cityapp. Do not remove or modify.
- skel_permanent -- This directory contains initial data and files to create a new PERMANENT location (see Grass manual), on demand of cityapp. Do not remove or modify.

Notes: if you want to use these mapsets in your native Grass, then launch Grass with GUI, and set:

- ~/cityapp/grass/ as “Grass data directory”,
- “global” as location,
- “PERMANENT” as mapset.

Geoserver

Geoserver is to provide maps for client browser on demand of Leaflet. Geoserver is a separeate application with separate settings, and its default installation path is /usr/share/geoserver. Geoserver’s starter file is /usr/share/geoserver/bin/startup.sh and the shutdown.sh in the same directory is to stop the service. Using Cityapp, a such manual start and stop is not necessary. Launching Cityapp (~/cityapp/scripts/base/ca_starter.sh) start automatically Geoserver too, and when Cityapp exit, Geoserver is exited automatically. While Cityapp is running, Geoserver is runnig continuously too.

This automatic process only works, when the cityapp user (the user, who manages cityapp, installed in his own directory) may acces to Geoserver’s files. Therefore, the administrator has to allows an execute acces to those files.

When -- for any reason -- it is necessary to start and stop Geoserver manually, while Cityapp is running, it is possible thru the Geoserver’s own starter and shutdown scripts. It affects no Ciytapp, but while Geoserver is inactive, maps are unavailables, therefore the client browser will unable to display them.

Gnuplot

Gnuplot is a separate application for data visualization. It has a Bash-like script language, allowing an automatized data processing. Cityapp use Gnuplot to run scripts, which are crated by a Cytiapp module. Gnuplot outputs are png files, loaded by Node.js to display graphs and plots in the frontend browser window. After once installed, no further modifications or settings are requested.

Further accesoires

Front-end

Front-end -- back-end Communication

Indicators

Indicators are to indicate if a module or a process is running/in progress. Indicators are simple text files, created in data_to_client directory. Indicators has a specific name, referring to a module or process. Indicators are changed in every second.

- When a module is started, first it create a file in data_to_client directory, with the name pattern: *modulename.running*

  modulename = name of the module, indicated by the file,

  running = word identifing if the indicitor relates to a module

  Example: *cityapp_add_map.running*

  File contetnt is an integer number, increased by 1 in each second (initial value is 1). When the moduleexit, indicator file is removed

- When a modul launch an internal process (for example: launching a GRASS task), during the process (until the process reaches its end), there will a file in data_to_client_directory, to indicate the process is running. Filename patern is as: *processdescription.processing*

  processdescription = an arbitrary description of the process (for example: import_vector)

  processing = word „processing” is to define if it is not a module indicator, but a process indicator

  integer = an integer number, increased by 1 in each second (initial value is 1)

  Example: add_raster.processing

  File contetnt is an integer number, increased by 1 in each second (initial value is 1). When the process is finished, the indicator file is removed.

Messages

Node.js is monitoring data_to_client directory for dedicated files. Files in this directory may only written by the backend and only read by the front-end. When a new file is detected with the filename pattern "module_name.id.message". For example, "location_selector.9.message." means:

- it is a message file,
- created by location_selector module,
- in the module each message has its own id number, and it is the message number 9.

The front-end (node.js) read its content, and perform the adequate action. Message file is a simple UTF-8 coded text file with JSON stucture. The file may contain the text message itself, the modal type of the message, the requested action, and supplementary components too, for example a list to display. "Modal type" is to describe the expected behave for the front-end and "Action" is to describe the possible options user may select among. "Action" is practically the set of displayed buttons user may select -- the button's text is the Action itself. It is useful, because this way of communication allows a back-end module to send ny kind of message, not only yes-no-cancel (for example: accept, refuse, etc.). Possible modal types:

- question -- question to answer or a simple message to acknowledge; actions are "Yes", "No", "Cancel".
- error -- error message, only acknowledge is allowed, action: "ok".
- select -- select one or more item from a list. List is included in the message file. Actions: "Yes"
- input -- numeric or string input value is expected, Action is "Yes".
- upload -- map uploading, osm, gpkg, geosjon, geotiff formats are only accepted. Action: "Yes"

For example, structure of message. location_selector.9:

```
{
"text":
"Process finished. No you can exit CityApp Location selector ",
"modalType": "question",
"actions": ["Yes"]
}
```

- The dialogue message has to be displayed: "Process finished. No you can exit CityApp Location selector"
- Type of the message: question,
- expected action: displaying only a "Yes" button (in this case it is enough, because only has to acknowledge the action.

Any running back-end module can create message file in the data_to_client directory, using Send_Message function (see later in the Function section). Leaflet functions, planted in the front-end allows a direct access to the back-end's Geoserver. Tehrefore those displayed maps comes not from data_to_client directory.

The other interface directory is data_from_browser directory, monitored by the back-end. Frontend only create files in this directory, and those files are read by the back-end. These files are created by the front-end java script, running in the browser. Filenames are case-sensitives! Accepted files:

- *request* -- Request file is a UTF-8 text file, containing a feedback from user, and its name is: "request". May contain text or numeric data too. User selection (yes, no, cancel, ok,) are stored in the file in simple text format, no JSON structure is requested. When the feedback is a selected item of a list, only the selected item is in the file. When the feedback is a list (for example: average speed on the different levels of the road network), the file contains this list in simple csv format. When the feedback is a numeric data, the file only contain the number. If the number is a floating point type number, comma (.) has to be used as decimal separator.
- *leave_session* -- this is an empty file with this name. When the file detected, the currently running module exit. Launching an other module (or relaunch the previous) is allowed.
- *EXIT* -- this is an empty file with this name. When the file is detected, the backend completly exit.

- *input maps* -- User may direcly select maps to upload, when it is requested, and frontend may create maps to store user selected area. Files are identified by its extensions. Accepted formats are (accepted extension):
  - open street map (osm, OSM)
  - geojson (geojson, GEOJSON)
  - geopackage (gpkg, GPKG)
  - geotiff (tif, TIF, tiff, TIFF, gtif, GTIF).

Data-flow (not yet elaborated chapter!)

Figure xx clearly shows the data flow (blue arrows). In this regard, “data” means files. In the front-end side, the data flow is linear and simple. Inthe back-

Since this directory is used only to send dialogue messages to user

Maps are directly served by Geoserver from its separate data directory (see later)

When back-end send a dialogue message to user, it is by a mess

User contacts the system thru the frontend only, displaying the dialogue messages, list, maps, map-tools, text and plot outputs. This frontend allows data entry and selections as well. No calculations a data processing delegated to the frontend, its only task is to communicate the user.

The backend is fully separated from the frontend. Back end is to store data, perform data processing and calculations. Outputs are passed to the frontend. The only interface between the frontend and the backend are data interchange directories:

data_from_browser and data_to_client. Communication between the front-end and the back-end is only by dedicated files created in these directories.

data_from_browser: accept input data, selection and commands from the user, data_to_client: query output files, and message files.

Modules

Base modules

Base modules performs no analysis or any query on maps, these are to manage mapsets and to set the system.

- add_location
- add_map
- checker
- make_selction
- process_check
- resolution_setting
- running_check
- select_lang

add_location

Description

Allows to set a new location by adding a new osm map data file. When a new location is set, center coordinates are exporeted into an external file, allowing to set the map center automatically to location center.

Requested inputs

An already downloaded osm file is requested to create the new location.

Communication

1. First is to check if there is a valid location. It means to check if there is a PERMANENT mapset of the current userbox.

If PERMANENT mapset not exist:

- text: No valid location found. First have to add a location to the dataset. Without such location, CityApp will not work. Adding a new location may take a long time, depending on the file size. If you want to continue, click Yes.

- message id: add_location.2.message
- modal type: question
- actions: [\"Yes\",\"No\"]
- expectation: A request file with yes or no text.
- consequence:

  If answer is NO, then location_selector send a message and when the message is acknowledged, exit:

  - text: Exit process, click OK.
  - message id: add_location.3.message
  - modal type: question
  - actions: [\"OK\"]
  - expectation: A request file with OK text.

  If answer is YES:

  - text: Select a map to add to CityApp. Map has to be in Open Street Map format -- osm is the only accepted format.
  - message id: add_location.4.message
  - modal type: upload
  - actions: [\"Yes\"]
  - expectation: Finding an uploaded osm file in data_from_browser directory. Request file is not expected, and therefore it is not neccessary to create.
  - consequence: No specific consequences

2. If there is an already existing PERMANENT mapset:

- text: There is an already added location, and it is not allowed to add further locations. If you want to add a new location, the already existing location will automatically removed. If you want to store the already existing location, save manually (refer to the manual, please). Do you want to add a new location? If yes, click OK.
- message id: add_location.1.message
- modal type: question
- actions: [\"Yes\",\"No\"]
- expectation: A request file with yes or no text
- consequence: If answer is yes:
  - text: Select a map to add to CityApp. Map has to be in Open Street Map format -- osm is the only accepted format.
  - message id: add_location.4.message
  - modal type: upload
  - actions: [\"OK\"]
  - expectation: Finding an uploaded osm file in data_from_browser directory. Request file is not expected, and tehrefore it is not neccessary to create.
  - consequence: No specific consequences

  If answer is no:

  - text: Exit process, click OK.

- message id: add_location.3.message
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with OK text.
- consequence: No specific consequences

3. Closing module

- text: New location is set. To exit, click OK.
- message id: add_location.5.message
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with OK text
- consequence: Module exit when message is acknowledged

add_map

Description

To add a new map to the current userbox. Calculations can only processed on already added map. When a map is added to the current mapset, it means, the selected map is converted into GRASS GIS binary format and stored in the PERMANENT mapset of the current userbox.

Requested inputs

User only has to define the file to add as a new map. Accepted formats are:

- geojson,
- geopackage,
- openstreetmap,
- geotiff

Filename extensions is requested to identify the format. Accepted extensions:

- geojson, GEOJSON
- gpkg, GPKG,
- osm, OSM,
- tif, TIF, tiff, TIFF, gtif, GTIFF

Since this module process no calculations, therefore no other specific input parameter is requested.

Communication

1. First is to check if there is a PERMANENT mapset in the current userbox.

If PERMANENT mapset not exist:

- text: "Selection" map not found. Before adding a new layer, first you have to define a location and a selection. For this end please, use Location Selector tool of CityApp. Add_Map modul now quit.
- message id: message.add_map.1
- modal type: error
- actions: [\"Ok\"]
- expectation: A request file with text OK
- consequence: Since no valid selection, the module exit after the user acknowledge the message.

2. If selection file found (It means, PERMANENT found):

- text: Select a map to add CityApp. Only gpkg (geopackage), geojson and openstreetmap vector files and geotiff (gtif or tif) raster files are accepted.
- message id: message.add_map.2
- modal type: upload
- actions: [\"OK\"]
- expectation: An uploaded file with a supported filename extension in data_from_browser directory. Request file is not expected, the question is only to draw the user's focus to the next step (select a file). Therefore in this case the trigger for the back-end is the presence of the uploaded file (and not a request file)
- consequence: When the selected file is uploaded succesfully, there is a new message:
  - text: Please, define an output map name. Name can contain only english characters, numbers, or underline character. Space and other specific characters are not allowed. For first character a letter only accepted.
  - message id: message.add_map.3
  - modal type: input
  - actions: [\"OK\"]
  - expectation: a request file with a single word as output name, defined by the user

3. Closing module

- text: Selected map is now succesfully added to your mapset. Add map module now exit
- message id: message.add_map.4
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with text OK
- consequence: Module exit after user acknowledge the message.

checker

Description

This module is to check if all requested external components are installed. It can only launched in a simple console (for example: xterm), no interactive UI (browser UI) it has. Checking for:

- geoserver
- node
- grass
- inotifywait
- enscript
- ghostscript
- gnuplot
- sed
- cut
- stat
- head
- tail
- grep

Results are reported on the console, and when there are not fund components, an error log is created:

```
~/cityapp/error_log
```

Requested inputs

No input is rquested

Communication

No user communication, only reportin on the console and createing an error log file, when it is necessary.

make_selection

Description

Allows to set or refine a selection. Selection is a selected part of the location. Calculations will only perform for the selection's area. When a new selection is taken, center coordinates are exporeted into an external file, allowing to set the map center automatically to location center.

Requested inputs

To set or refine the selection, a geojson file is requested. An integer type numeric value is also requested to set the resolution.

Communication

1. First is to check if there is a valid location. It means to check if there is a PERMANENT mapset of the current userbox.

If PERMANENT mapset not exist:

- text:No valid location found. First have to add a location to the dataset. Without such location, CityApp will not work. To add a location, use Add Location menu. Now click OK to exit.
- message id: make_selection.1.message
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with OK text.
- consequence: Exit the module.

2. If there is an already existing PERMANENT mapset:

- text: Now zoom to area of your interest, then use drawing tool to define your location. Next, save your selection.
- message id: make_selection.2.message
- modal type: question
- actions: [\"OK\"]
- expectation: Finding an uploaded goejson file in data_from_browser directory. This file is created by the browser, when the user define interactively the selection area. Request file is not expected, and therefore it is not neccessary to create.
- consequence: No specific consequences

3. Resolution Setting. This is by calling an other module (cityapp_resolution_setting.sh). therefore the message name is now refers to that module.

- text: Type the resolution in meters, you want to use. For further details see manual.
- message id: message.resolution_setting.1

  modal type: input
- actions: [\"OK\"]
- expectation: A request file with a positive number.
- consequence: If user gives a negative number, then UNTIL number is greater than zero:
  - text: Resolution has to be an integer number, greater than 0. Please, define the resolution for calculations in meters.
  - message id: message.resolution_setting.2
  - modal type: input
  - actions: [\"OK\"]
  - expectation: A request file with a positive number.
  - consequence: No specific consequences

4. Closing module

- text: Process finished, selection is saved. To process exit, click OK.
- message id: make_selection.3.message
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with OK text
- consequence: Module exit when message is acknowledged

process_check

Description

It is a system module, user has no business with: this module is automatically started and stopped by a currently running cityapp module. It is to indicate for the node.js, that there is process which may take a longer time to run. For this end, while process_check is active, there is an indicator file in ~/cityapp/data_to_client directory with the name pattern:

```
process_description.processing
```

process_description is an arbitrary identifier, for example:

```
map_calculation.processing
```

This file is refreshed in each second, and the uptime of the process in seconds is written into the file as an integer number. While the file is present, the front-end displays a process message, which is a rotating 3/4 circle and the “processing...” text.

When process is finished, processing file is automativally removed.

Requested inputs

No input is requested.

resolution_setting

Description

Allows to set the resolution in meters for the entire location. Resolution will considered by other modules which processes calculations. This module first check if ~/cityapp/scripts/shared/variables/subprocess exist or not. If exits, it means, cityapp_resolution_setting.sh was launched by an other modeule, and therefore it is not requested to send message resolution_setting.3.message at the end of the process. If “subprocess” file not found, then “business as usual”, and the last message will sent too.

Requested inputs

An positive numeric value is requested to set the resolution.

Communication

1. Giving a resolution value

- text: Type the resolution in meters, you want to use. For further details see manual.
- message id: message.resolution_setting.1
- modal type: input
- actions: [\"OK\"]
- expectation: A request file with a positive number.
- consequence: If user gives a negative number, then UNTIL number is greater than zero:
  - text: Resolution has to be an integer number, greater than 0. Please, define the resolution for calculations in meters.
  - message id: message.resolution_setting.2
  - modal type: input
  - actions: [\"OK\"]
  - expectation: A request file with a positive number.
  - consequence: No specific consequences

2. Closing module

- text: Resolution is now set, to exit, click OK.
- message id: resolution_setting.3.message

- modal type: question
- actions: [\"OK\"]
- expectation: A request file with text OK
- consequence: Module exit after user acknowledge the message.

running_check

Description

It is a system module, user has no business with that: this module is automatically started and stopped by currently running cityapp module. It is to indicate for the node.js, that there is running module. For this end, while running_check is active (the module is running), there is an indicator file in ~/cityapp/data_to_client directory with the name pattern:

```
module_name.running
```

module_name is the module script name, for example:

```
cityapp_module_1.running
```

This file is refreshed in each second, and the uptime of the module (in seconds) is written into the file as an integer number.

When module is finished, “running” file is automativally removed.

Requested inputs

No input is requested.

select_lang

Description

It is to set the language text for user dialogues. When the module is starting, first check subdirectories of ~/cityapp/scripts/shared/messages. Result is written into ~/cityapp/scripts/modules/select_lang/available_languages file. Now, the module send a dialgue message to user to select one of those items. Selection is written in ~/cityapp/scripts/shared/variables/lang file. From now, that is the dialogue language for each module.

Requested inputs

A selection from list of available languages -- a string value of the language (county code), which is a single word.

Communication

1. Selecting a language

- text: Select language. Available languages are:
- message id: select_lang_1.message
- modal type: input
- actions: [\"OK\"]
- expectation: A request file with a single word (string).
- consequence: If user has selected from the list, the value will written in ~/cityapp/scripts/shared/variables/lang file.

  If user left the selection field blank (seleced nothing), then fallback to default value: selection is English.

  - text: No language selected, setting default value (English) for user dialogue. Restart Lagnuage Settings module to set an other languages. To exit, click OK.
  - message id: select_lang_2.message
  - modal type: error
  - actions: [\"OK\"]
  - expectation: An empty request file
  - consequence: English is set as language and module exit after user clicked “OK” button.

2. Closing module

- text: Selected language is now set. To exit, click OK.
- message id: resolution_setting.3.message
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with text OK
- consequence: Module exit after user acknowledge the message.

module_1

Description

To calculate the time to reach "target" points on road network from "start" points thru "via" points. Using default *target point* value will calculate time to reach all points of the road network from “start” points, covering the entire “selection” area.

Requested inputs

- *road map* -- line vector map, already added OSM map. User may not select a road map, since the road map is automatically created when the new location is created by location_selector.sh
- *start point* -- at least one, but it is possible to add more points too. Point vector map. Points may generated by the browser as a geojson file, if points are selected interactively by user. User may select an already existing point map too, to use its points as start points. There are no default start points -- without start point(s), the module can not to calculate an output.

Optional inputs

- *via point* -- at least one, but it is possible to add more points too. Point vector map. Points may generated by the browser as a geojson file, if points are selected interactively by user. User may select an already existing point map too, to use its points as via points. No default points. If no via points are defined, calculation ill continue without considering via points.
- *target point* -- at least one, but it is possible to add more points too. Point vector map. Points may generated by the browser as a geojson file, if points are selected interactively by user. User may select an already existing point map too, to use its points as targer points. If target point is selected, the calculation will finished when the target point is reached. When there are more than one target point, the calculation wil finished when the first is reached.

Default value exists. If no target points are selected, default value is all points on the road network: the module will calculate the time for each point of the road network of selected area.

- *stricken area* -- Are, with limited speed on the network. No default values. User may define interactively an area on the map, in this case the browser create a geojson are map as output in data_from_browser directory. It is also possible to select an already existing area map.

If stricken area is selected, user has to define a speed reduction value too. Value is a floating point number, greater than 0 (zero). Zero is not allowed. This number represent the speed reduction in the stricken area. For example, if the value i 0.1, it means, that in the zone the average speed is 1/10 of the original values for each road level. If the original value for the primary roads was 40 km/h, now in the stricken area this will only 4 km/h. Stricken area is not requeste, but when once defined, the speed reduction value is requested to continue the calculations.

- *speed on the road network* -- numeric parameters. Defining an average speed for each road level in km/h unit. Since maps are OSM maps, therefore the possible levels are:
  - Motorways
  - Trunk and primary
  - Secondary roads
  - Tertiary roads
  - Connecting roads
  - Service and track roads
  - Residential roads
  - Living_streets
  - Footways

  Even, there are default values, the user onlyy asked to use the previous speed settings or not (and, when the module is used at the first time, the defaut values are user as „previous” values)

Communication

1. First is to check if there is a valid PERMANENT mapset (GRASS data set). If not exist:

- text: No valid location found. Run Location selector to create a valid location. Module is now exiting.
- message id: message.module_1.11
- modal type: error
- actions: [\"Ok\"]
- expectation: request file with Ok text
- consequence: Since no valid mapset found the module exit after the user acknowledge the message.

2. Start point definition.

- text: Start points are required. Do you want to draw start points on the basemap now? If yes, click Yes, then draw one or more point and click Save button. If you want to use an already existing map, select No.
- message id: message.module_1.1
- modal type: question
- actions: [\"Yes\",\"No\"]
- expectation: request file with text Yes or No
- consequence: If answer is "yes", the module is waiting for a geojson file in data_from_browser. Module only goes to the next step, when geojson file is created.

If answer is "no", module send a new message:

  - text: Select a map (only point maps are supported). Avilable maps are:
  - message id: message.module_1.2
  - modal type: select
  - actions: [\"OK\"]
  - expectation: request file with the select item only.

    Since „message.module_1.2” contains a list (list items are the availabe maps), user has to select one of them. The modal type is select, therefore the answer (new request file) conatains only the selected item (in this case: a map name). It is not expected to create a separate request file containig "yes".

3. Via points definition

- text: Via points are optional. If you want to select 'via' points from the map, click Yes. If you want to use an already existing map, select No. If you do not want to use via points, click Cancel.
- message id: message.module_1.3
- modal type: question
- actions: [\"Yes\",\"No\",\"Cancel\"]
- expectation: request file with text yes or no or cancel.
- consequence: If answer is "yes", the module is waiting for a geojson file in data_from_browser. Module only goes to the next step, when geojson file is created.

If answer is "no", module send a new message:

  - text: Select a map (only point maps are supported). Avilable maps are:
  - message id: message.module_1.4

  - modal type: select
  - actions: [\"OK\"]
  - expectation: request file with the select item only.

  Since „message.module_1.4" contains a list (list items are the availabe maps), user has to select one of them. The modal type is select, therefore the answer (new request file) conatains only the selected item (in this case: a map name). It is not expected to create a separate request file containig "yes".

If answer is cancel:

  Nothing to do.

4. Target points

- text: Target points are required. If you want to select target points from the map, click Yes. If you want to use an already existing map containing target points, click No. If you want to use the default target points map, click Cancel.
- message id: message.module_1.5
- modal type: question
- actions: [\"Yes\",\"No\",\"Cancel\"]
- expectation: request file with text yes or no or cancel.
- consequence: If answer is "yes", the module is waiting for a geojson file in data_from_browser. Module only goes to the next step, when geojson file is created.

If answer is "no", module send a new message:

  - text: Select a map (only point maps are supported). Avilable maps are:
  - message id: message.module_1.6
  - modal type: select
  - actions: [\"OK\"]
  - expectation: request file with the select item only

  Since „message.module_1.6" contains a list (list items are the availabe maps), user has to select one of them. The modal type is select, therefore the answer (new request file) conatains only the selected item (in this case: a map name). It is not expected to create a separate request file containig "yes".

If answer is cancel:

  Nothing to do.

5. Stricken area

- text: Optionally you may define stricken area. If you want to draw area on the map, click Yes. If you want to select a map already containing area, click No. If you do not want to use any area, click Cancel.
- message id: message.module_1.7

  modal type: question
- actions: [\"Yes\",\"No\",\"Cancel\"]
- expectation: request file with text yes or no or cancel.
- consequence: If answer is "yes", the module is waiting for a geojson file in data_from_browser. Module only goes to the next step, when geojson file is created.

If answer is "no", module send a new message:

  - text: Select a map (only area maps are supported). Avilable maps are:
  - message id: message.module_1.13

  - modal type: select
  - actions: [\"OK\"]
  - expectation: request file with the select item only

    Since „message.module_1.13” contains a list (list items are the availabe maps), user has to select one of them. The modal type is select, therefore the answer (new request file) conatains only the selected item (in this case: a map name). It is not expected to create a separate request file containig "yes"

If answer is cancel:

  Nothing to do.

6. Speed reduction for stricken area (if stricken area is created or selected):

- text: Set speed for roads of stricken area.
- message id: message.module_1.12
- modal type: input
- actions: [\"OK\"]
- expectation: reqest file with single a floating point numeric value

7. Setting average speed values

- text: Do you want to set the speed on the road network? If not, the current values will used.
- message id: message.module_1.9
- modal type: question
- actions: [\"Yes\",\"No\"]
- expectation: request file with a single yes or no.
- consequence: If answer is "yes", there is a new message:
  - text: Message Now you can change the speed values. Current values are:
  - message id: message.module_1.10
  - modal type: select
  - actions: [\"OK\"]
  - expectation: It is a bit different than the other "select" types. The message file itself containes a list, showing the road levels and its average speed values. But now expectation is NOT only a selected item! The expected answer (request file) is now the same list, containing all lines in the original order AND the speed values (even, only one or a few are changed). OR, it is also possible, to create a request file, containing only the numeric values, each, in the original order.

8. Finished

- text: Calculations are ready, display output time maps.
- message id: message.module_1.14
- modal type: question
- actions: [\"OK\"]
- expectation: A request file with a single "OK" word
- consequence: After the user acknowledge the message, the module exit.

module_2

Description

Query module. It allows to query a vector map. User may select one or more arbitrary area and the module will output its statistics, and a map, displaying features, matches to the user-defined criterias. Output also written in a pdf file (see message 3.).

Requested input

- Query area -- at least one area, covering a part of the selection (see location_selector). Drawn by user on the map. It is also allowed to draw more than one disjunct area.
- Map to query -- a vector map name to query. Only point or area maps are supported, which has at least one numeric column, except the cat (category) column.
- Criteria -- User may define criteria to select features from the map. Only a very simple SQL statement can be taken, but in most case, that will probably eligible.

Communication

1. Query area

- text: Draw an area to query
- message id: module_2.1.message
- modal type: question
- actions: [\"Ok\"]
- expectation: Finding an uploaded goejson file in data_from_browser directory. This file is created by the browser, when the user define interactively the selection area. Request file is not expected, and therefore it is not neccessary to create.
- consequence: No specific consequences.

2. Select a map to query

- text: What is the map you want to query? Available maps are:
- message id: module_2.2.message
- modal type: select
- actions: [\"Ok\"]
- expectation: request file with the select item only

  Since this message contains a list (list items are the availabe maps), user has to select one of them. The modal type is select, therefore the answer (new request file) conatains only the selected item (in this case: a map name). It is not expected to create a separate request file containig "yes"
- consequence: No specific consequences.

3. Fill the query form

- text: Fill the form and press save.
- message id: module_2.2.message
- modal type: form
- actions: [\"Ok\"]
- expectation: request file with a filled form in json format. It is not expected to create a separate request file containig "yes"

  „form" is a new modal type: it is an empty, preformatted form and user have to fill the empty fields, or by arbitrary values, or by selecting from a list. This form is typical: first field is the name of the column to query. This is a list, containig avilable columns of map to query (see point 2.) . There are fields, containing predefined elements: < > = ( ). The other fields are left to the user to fill using column names or numeric values.

3. Close

- text: Statistics output are ready. When click Close, the results will deleted from the screen! Only click Close, when you want to exit the module. Results are already saved into a pdf file. File is located in ~/cityapp/saved_results. Filename pattern is: query_results_yy_mm_dd_hh_mm.pdf (year_month_day_hours_minutes).
- message id: module_2.4.message
- modal type: question
- actions: [\"Close\"]
- expectation: request file with „Close" word.
- consequence: When request file is written, the module exit, and files from data_to_client directory will removed.

4. Info

Info type output means, that its content have to displayed in the right panel (info panel). Formatting depends on the frontend. Since „info" is a very generic type, may content different structures, lists, text, numeric output. This info file is displayed right afret the previous message.

- message id: module_2.1.info
- expectation: No specific request