UI Framework Guidelines

Release 1.5.0

Software and Controls Group

Dec 20, 2018

Contents

Note: The UI framework is currently only supported on MacOS. Linux support will be available in future releases.

The UI Framework introduces a set of libraries and a windowed application that provides a GUI for the OCS. The framework handles three primary concerns

- Rendering (drawing) elements to the screen (DOM)
- Library of re-usable UI components that can be shared across the project
- An App that provides an interface to the OCS

CHAPTER

ONE

ENGINEERING APP

The engineering application provides a GUI to the OCS model. It is launched via the command line and it will spawn an OS window that renders the UI.

The UI engineering app uses your local bundles from `$GMT_LOCAL/etc/bundles` and allows you to see a visual representation of your model files' input/output ports. For now, the Engineering app needs to run in MacOS or a Linux Desktop.

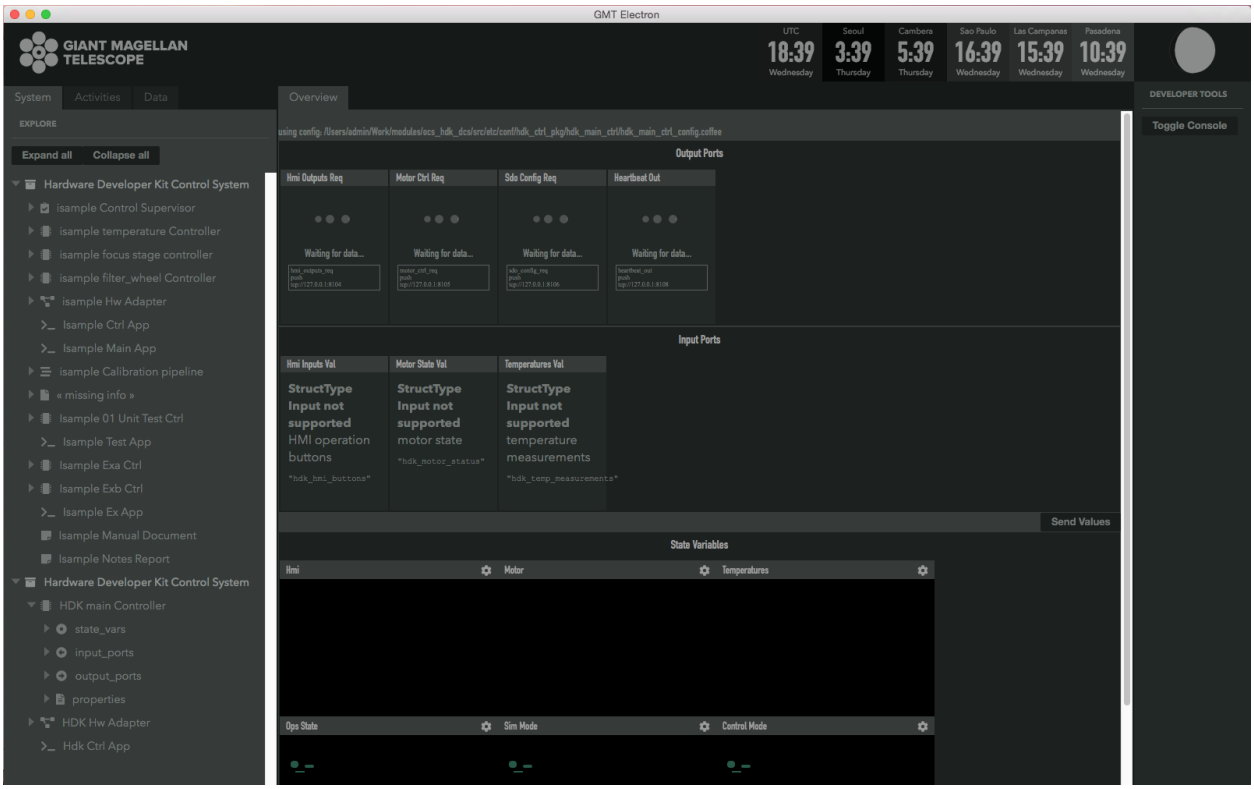
To launch the application, run this in the command line

```
$ navigator
```

This will launch the GUI as a child process of the CLI application. To stop the GUI, stop the CLI app with `CTRL + C`.

1.1 Configuration

The User Interface needs to be configured to connect to the correct control components to receive data. Without proper configuration, the application may look like this:



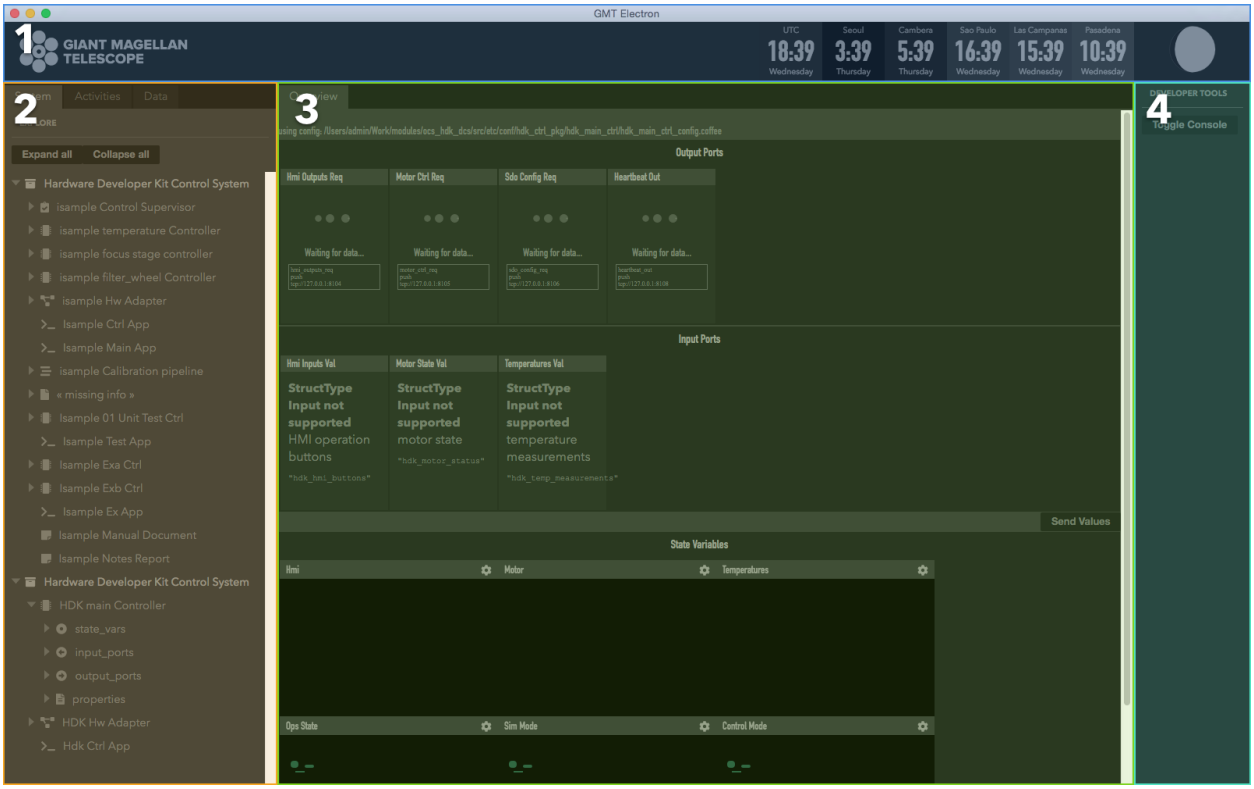
Edit the appropriate config files in the `src/etc/conf` folder to point to the correct IP address for input and output ports. For example,

```
$ cd $GMT_LOCAL/modules/ocs_hdk_dcs/src/etc/conf/hdk_ctrl_pkg/hdk_main_ctrl/
$ sed -i '' "s/172.0.0.1/172.16.10.31/g" hdk_main_ctrl_config.coffee
```

See the Troubleshooting section below for more help with connection issues.

Restart the Navigator application for changes to take affect.

1.2 Guide



The navigator application contains four regions.

1. **Header** The header contains timezone information, as well as a moon cycle representation.
2. **Navigation** This area contains the navigation tree. The tree is a representation of your model and is built from information found in your local bundles.
3. **Context** Items selected in the tree will display here. Currently, it will display a representation of your model's input/output ports and state variables.
4. **Developer tools** This area is reserved for developer tools. The *Toggle Console* button will show the developer console.

Note: The **Context** area will optimistically render your model. Not all model data can be currently rendered. Some items like *properties* and detailed port views are currently not supported.

Warning: Input ports are rendered as data inputs based on the type data encoded in your model. The *StructType* input is not yet supported. When you press the `Send Value` button the app will attempt send the data in *all* the input fields to each respective port defined.

1.3 Launching Custom panels

The `navigator` application can also launch custom panels that are defined in a DCS' **_vis_pkg* folders. When defined, it's possible to launch standalone panels with the following parameters

```
$ navigator --panel emf_custom_weather_view --port 9098
```

The `--panel` flag specifies an exported panel in some vis package. The `--port` flag is currently required to avoid port collisions (for now).

The engineering app reserves port `9199`. Custom panel launches of the application need to specify a different port for each instance.

Note: When running a custom panel, only the **Header** and **Context** regions will be shown. The panel content is rendered in the **Context** region.

1.4 Troubleshooting

The engineering app loads the local bundles defined in `$GMT_LOCAL/etc/bundles`. It currently uses the model generated config files to read data for your package. Those config files are created in `$GMT_LOCAL/modules/<your_module>/src/etc/conf/<your_package>_pkg/<component>_config.coffee`; it's useful to see what's in those configs when troubleshooting data availability issues. The availability of data to the UI largely depends on those config files. The values generated will depend on how you write your model, but a sample of a config file might look like

```
module.exports =
    properties:
        uri: { name: 'uri', default_value: 'gmt://hdk_dcs/hdk_main_ctrl/hdk_
↪main_ctrl' , type: 'String', desc: 'Uri path for the component' }
        # other fields ommited

    state_vars:
        hmi: { name: 'hmi',    }
        motor: { name: 'motor',    }

    input_ports:
        hmi_goal:            { name: 'hmi_goal',              protocol: 'pull
↪',  url: 'tcp://127.0.0.1:8116', blocking_mode: 'async', max_rate: 1000, ␣
↪nom_rate: 1      }
        motor_goal:          { name: 'motor_goal',            protocol: 'pull
↪',  url: 'tcp://172.16.10.31:8117', blocking_mode: 'async', max_rate: 1000,
↪  nom_rate: 1      }

    output_ports:
        hmi_value:           { name: 'hmi_value',             protocol: 'pub
↪',   url: 'tcp://127.0.0.1:8122', blocking_mode: 'async', max_rate: 1000, ␣
↪nom_rate: 1      }
        motor_value:         { name: 'motor_value',           protocol: 'pub
↪',   url: 'tcp://172.16.10.31:8123', blocking_mode: 'async', max_rate:␣
↪1000,  nom_rate: 1      }
```

When troubleshooting it's important to note the *protocol* and the *url* keys for a given port. For example the `hmi_value.url` value is `tcp://127.0.0.1:8122` this means you're trying to connect to port number `8122`

on the address `127.0.0.1` (which is typically your local machine). Whereas the `motor_value.url` is trying to connect to a different computer with an IP address `172.16.10.31` on port number `8123`.

If the computer at `172.16.10.31` is firewalled and not allowing connections to port `8123`, you will not be able to see data. You will need to allow incoming connections to that port. Likewise, if your component is running at the computer at `172.16.10.31` and you are trying to read data from `127.0.0.1`, you will not see any data. You will need to change the IP to match the computer where your component is running.

Additionally, the UI can only read data from ports configured with the `pub` protocol.

If you make changes to the config file, you will need to restart the command line app; you can do this by pressing `CTRL + C`.

- **MacOS - nvm command not found**: If, after installing nvm, the command `nvm install` fails with the message, `command not found`, check the `~/.bash_profile` file to ensure that it contains a command for loading nvm. The .bash_profile file should contain the following lines:

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
```

- **Unresponsive UI**: in some case if the UI becomes unresponsive, press `CMD+R` to refresh. If that fails to solve the problem, restart the CLI app. You can stop the CLI app with `CTRL+C`.
- **No navigation tree**: the navigation tree is rendered off the local bundles in `$GMT_LOCAL\etc\bundles`. The bundles described there need to have been built with webpack.
- **No data**: Ensure that the ports used by the controllers to publish data are accessible through the firewall. The following command should be used on the Device Control Computer to open the applicable range of ports (8122 - 8124):

```
$ sudo firewall-cmd --add-port=8122-8124/tcp
```