Sidings Media Railway Controller

Release STABLE-d1d5906

SMRC Team

Nov 12, 2021

TABLE OF CONTENTS

Table of contents i

1 Repository information 3
- 1.1 Repositories . . . 3
- 1.2 Licence . . . 3

2 Boards 5
- 2.1 Firmware . . . 5
  - 2.1.1 Repository . . . 5
- 2.2 Hardware . . . 5
  - 2.2.1 Repository . . . 5
- 2.3 Bootloader . . . 5
  - 2.3.1 Repository . . . 5
  - 2.3.2 Use of bootloaders . . . 5
  - 2.3.3 General Requirements . . . 6
  - 2.3.4 Main board . . . 6
  - 2.3.5 Other Boards . . . 6

3 Clients 7
- 3.1 Desktop Client . . . 7
  - 3.1.1 Repository . . . 7
  - 3.1.2 Initial Requirements . . . 7

4 For Developers 9
- 4.1 Contributing . . . 9
  - 4.1.1 Getting Started . . . 9
  - 4.1.2 Style guide . . . 11
  - 4.1.3 The legal bit . . . 11
  - 4.1.4 Documenting your contribution . . . 12
  - 4.1.5 Pull Request Guidelines . . . 13
- 4.2 Logos . . . 15
  - 4.2.1 Using the Logos . . . 15
  - 4.2.2 Directory structure . . . 16
- 4.3 Desktop Client . . . 16
  - 4.3.1 Directory structure . . . 16
  - 4.3.2 Building . . . 17
  - 4.3.3 Running . . . 17
  - 4.3.4 Packaging . . . 17
  - 4.3.5 Desktop Client Preload API . . . 17
  - 4.3.6 Main Process Classes . . . 19

Revision d1d5906 on branch stable

Revision d1d5906 on branch stable

Sidings Media Railway Controller aims to provide you a quick and easy way of controlling your model railway from your computer without all of the complications of DCC.

CHAPTER

ONE

REPOSITORY INFORMATION

1.1 Repositories

This project is split into multiple repositories. Each repo can be found in the Sidings Media organization and are prefixed with SMRC. There are repos for firmware, hardware and various clients. An extensive list can be found below. All documentation for the project can be found in this repo under the docs directory.

Repo URL	Description	Maintained
https://github.com/SidingsMedia/SMRC-Desktop-Client	The cross platform desktop client	Yes
https://github.com/SidingsMedia/SMRC-hardware	All board designs and schematics	Yes
https://github.com/SidingsMedia/SMRC-firmware	All firmware for the boards	Yes
https://github.com/SidingsMedia/SMRC-logos	The logos for the project	Yes
https://github.com/SidingsMedia/SMRC-bootloader	The bootloaders for all official SMRC boards	Yes
https://github.com/SidingsMedia/SMRC-template	The template for all SMRC repos	Yes

1.2 Licence

All official SMRC projects (i.e all in the table above), use the REUSE[1] standard in order to communicate the correct licence for the file. For those unfamiliar with the standard the licence for each file can be found in one of three places. The licence will either be in a comment block at the top of the file, in a `.license` file with the same name as the file, or in the dep5 file located in the `.reuse` directory. If you are unsure of the licensing terms please email contact@sidingsmedia.com.

[1] https://reuse.software

Revision d1d5906 on branch stable

CHAPTER

TWO

BOARDS

The boards for SMRC provide the physical bridge between the hardware and software. They are the devices that control the trains on the track and lights in the model houses.

2.1 Firmware

2.1.1 Repository

The source files for the firmware can be located in the SMRC-firmware[2] repository.

2.2 Hardware

2.2.1 Repository

The source files for the hardware can be located in the SMRC-hardware[3] repository.

2.3 Bootloader

2.3.1 Repository

The source files for the bootloaders can be located in the SMRC-bootloader[4] repository.

2.3.2 Use of bootloaders

The bootloader is used on all official SMRC boards (i.e all boards in the SMRC-hardware[5]) repository. It can also be used in your own SMRC boards.

[2] https://github.com/SidingsMedia/SMRC-firmware
[3] https://github.com/SidingsMedia/SMRC-hardware
[4] https://github.com/SidingsMedia/SMRC-bootloader
[5] https://github.com/SidingsMedia/SMRC-hardware

2.3.3 General Requirements

- Must support updating firmware over USB
- Must make available to the firmware the current bootloader version

2.3.4 Main board

2.3.4.1 Requirements

- Must verify that the firmware being uploaded is correct for the processor

2.3.5 Other Boards

2.3.5.1 Requirements

- Must verify that the firmware being uploaded is correct for the specific board

CHAPTER

THREE

CLIENTS

Just as the boards provide the physical interface between the software and the trains, the clients provide the bridge between the user and the software. They provide easy to use interfaces to allow the user to control their model railway from the comfort of their device.

3.1 Desktop Client

3.1.1 Repository

The source files for the desktop client can be located in the SMRC-Desktop-Client[6] repository.

3.1.2 Initial Requirements

These are the requirements that the initial version must meet before being released. This is not an extensive list of all future features and more will be added with later releases.

- Can update the firmware of boards over USB
- Can communicate with main board over Wi-Fi
- Can upload new devices to main board
- Can assign addresses to boards over USB
- User can control speed of trains via slider
- User can control relay channels

[6] https://github.com/SidingsMedia/SMRC-Desktop-Client

CHAPTER

FOUR

FOR DEVELOPERS

If you are developing for SMRC or just want to find out more about how the system works then this is the place for you.

4.1 Contributing

Being an open source project, we love it when people want to contribute, after all, it's the best part of being open source, anyone can pitch in to create the best application possible. This section gives you some more information on the whole process from selecting which features to work on to finally getting your code merged into the project. There is no need to worry if it's your first time contributing to a project, we were all new once, if you need any more help then check out the SMRC-help[7] gitter room or email us at contact@sidingsmedia.com. For more information on contributing open source projects in general you should check out How to Contribute to Open Source[8] for an overview and the GitHub docs on how to create a Pull Request[9] for some more technical information.

4.1.1 Getting Started

Getting started on your contribution is often one of the hardest parts of open source, especially if your new. If you have a new idea for a feature or have found a bug then you should create an issue[10] in the appropriate repository. More information on which parts of the project belong to which repository can be found in the *Repositories* section. We ask that you do this so that we can discuss your proposal and decide if we want to incorporate the feature into the project. We don't want you to spend hours working on your contribution only for it to be rejected. Don't worry if we don't think that your idea should be incorporated, this just means that we feel the project isn't quite ready for such a feature, you never know, we may come back to you once the project has progressed a little further to see if you still want to work on your idea.

4.1.1.1 Forking and Cloning the repository

In order to start making your changes you will first need to create a fork[11] of the repository you wish to make changes on. Once you have created the fork you need to clone your version of the repository. This is fairly simple but because we use submodules[12], there are a few more step that you must undertake to successfully pull all of the required files to your local environment. The commands that you should run are as follows:

```
$ git clone https://github.com/(username)/(repository).git
$ git submodule init
$ git submodule update
```

In this example we first clone the repository as normal. The next two commands initialize the submodules and pull them from their repositories. If you are running Git 2.13 or later you can run the following command instead:

[7] https://gitter.im/SidingsMedia/SMRC-help?utm_source=share-link&utm_medium=link&utm_campaign=share-link
[8] https://opensource.guide/how-to-contribute/
[9] https://help.github.com/articles/creating-a-pull-request/
[10] https://docs.github.com/en/issues/tracking-your-work-with-issues/creating-an-issue
[11] https://docs.github.com/en/get-started/quickstart/fork-a-repo
[12] https://git-scm.com/book/en/v2/Git-Tools-Submodules

```
$ git clone --recurse-submodules https://github.com/(username)/(repository).git
```

This command does all that was done with the previous three commands at once. It doesn't really matter which you use as both methods do the same thing.

4.1.1.2 Creating your issue

This is a fairly simple process. Most of the repos will have pre-made templates for you to fill in when creating an issue. If this is the case then when you create an issue you will be given the option of creating a feature or bug issue. Pick whichever one is appropriate. This will give you a form to fill in. Just fill this in to the best of your ability, remember, try to be detailed yet concise, we won't know what your talking about if you just write something like `Update request to board` but then again, we can't spend an hour reading one issue so don't be too in depth.

Giving your issue a title

This is one of the most important parts of creating your issue as this is the first thing we see when we scan down the list of new issues and pull requests. Your issue title should be brief and to the point, whilst also giving some idea about what you want to do. Using the example from before, a good title would be something like:

```
[FEATURE]: Update AJAX requests to use fetch()
```

Notice the [FEATURE]: section of the title. This should be pre-filled if you use the templates but if it isn't it should be one of the following:

- [BUG]:
- [FEATURE]:

You should choose the appropriate option for your issue.

Describing your bug or feature

This is key to helping us understand what you want to do. As we said before, you should try and be as detailed as possible whilst still being brief. Something like this is great:

```
I want to update the AJAX requests in the desktop client to use the
new fetch method instead of XMLHttpRequest. I think that this would
be beneficial as the fetch API is more geared towards the modern web
app and provides numerous helpful features.
```

This clearly tells us what the contributor wants to do, replace all of the AJAX calls with a new method, and it tells us why they think it would be beneficial to the project, a more modern API designed for web apps.

Describing alternatives

Although this may seem a bit unnecessary, it is actually very important as it allows us to see if your idea is the best way of approaching the issue. This allows us to get the overall best possible solution for the project. Something like this is good:

```
I have considered keeping the original XMLHttpRequest calls. This
would mean that large portions of the request code would not need to
be rewritten but overtime these old sections of code would become
harder to maintain.
```

This is good because it details both the pros and cons of the alternative. This allows us to form a balanced assessment of what is best for the project.

4.1.1.3 What to do next

After you have created your issue, all you need to do is wait. We know this may seem a bit boring but we will try to respond to your issue within a couple of days. It may take us a little longer if we are really busy with a new release or lots of issues are coming in. We may not give you a definite answer at first and we may want to discuss it further with you. Don't worry, we are just trying to find out more about your idea or bug so we can make a decision on it.

Once you have gotten the go ahead for your idea then you can start making your changes. Just fork the repository and make your changes there. Once your done you can *Pull Request Guidelines* proposing to merge your changes into the `develop` branch. You can also create a pull request when you start working on your feature by creating a *Work in progress* pull request. This isn't essential but it just makes it a bit easier for us to see how your feature is progressing and assist you if you need any help.

4.1.2 Style guide

4.1.2.1 Code

In order to help keep our code readable and easy to understand, we apply style guides to all of the SMRC repositories. These are the same guides that are used by Google and a list of all of them can be found in their google/styleguide[13] repository. To make it easier there is a table below that contains all of the language style guides that apply.

Language	Style sheet
TypeScript	https://google.github.io/styleguide/tsguide.html
C++	https://google.github.io/styleguide/cppguide.html
HTML/CSS	https://google.github.io/styleguide/htmlcssguide.html

4.1.2.2 Schematics

All of the schematics for the project should use the template file located in the root of the SMRC-hardware[14] repository called `pageLayout.kicad_wks`. This helps to make sure that all of our schematics follow the same design and makes sure that all the required information is included.

4.1.3 The legal bit

4.1.3.1 Contributor Covenant Code of Conduct

This project operates a code of conduct in order to ensure that the project remains an inclusive and supportive environment for anyone and everyone who wishes to contribute to the project in any way. You can find a copy of the code of conduct[15] in the main repository for the project, SidingsMedia/Sidings-Media-Railway-Controller[16]. By participating in this project you agree to abide by its terms.

[13] https://github.com/google/styleguide
[14] https://github.com/SidingsMedia/SMRC-hardware
[15] https://github.com/SidingsMedia/Sidings-Media-Railway-Controller/blob/main/CODE_OF_CONDUCT.md
[16] https://github.com/SidingsMedia/Sidings-Media-Railway-Controller

4.1.3.2 Developer Certificate of Origin

In order to ensure that all of the code contributed to the repositories is the work of the contributor or that the contributor has the correct permissions to submit the work. It is the same DCO that is used for the Linux kernel and it can be found in the file called `DCO_V1.1` in the root of each official SMRC repository. You can also see the terms at developercertificate.org[17]. To demonstrate that you accept the DCO and that you are permitted to contribute the work, you should add a sign off line at the end of your commit message in the following form:

```
Signed-off-by: Full Name <email>
```

You can automatically add this to the end of your commit message by adding the `--signoff` option to `git commit`. If you use VS Code then you can enable the `git.alwaysSignOff` option in your settings. For more information on how to contribute to a project that uses a DCO you should see the Developer Certificate of Origin (DCO)[18] documentation made available by The Linux Foundation[19].

4.1.4 Documenting your contribution

Documentation is vital to any project. Without it how would anyone know how to use it? This is why, if you make changes that require modifying the documentation, you must update the docs as well.

4.1.4.1 What changes need documenting

Whilst this is not an extensive list of what changes would need documenting, it should give you an idea of what types of changes would need docs. If you are still unsure then feel free to ask a question on the SMRC-help[20] Gitter channel or send an email to contact@sidingsmedia.com. You could also add a comment on your initial issue but it will probably take longer for you to get a response. The sort of changes that would require documentation changes are:

- API changes
- Class method or property changes
- User interface changes

4.1.4.2 Creating a documentation pull request

Note: This is only relevant for creating pull requests for documentation modifications created as a result of pull requests in other repositories. This does not cover creating ordinary pull requests. See the *Pull Request Guidelines* for more information on creating normal pull requests.

When you make contributions to the project, you may have to update the projects documentation. To do this you will need to open a separate pull request in the `Sidings Media Railway Controller` repository. This pull request does not have to follow the same standards as stipulated in *Pull Request Guidelines*. Instead you should follow the formatting standards below.

[17] https://developercertificate.org/
[18] https://wiki.linuxfoundation.org/dco
[19] https://linuxfoundation.org/
[20] https://gitter.im/SidingsMedia/SMRC-help?utm_source=share-link&utm_medium=link&utm_campaign=share-link

Naming your PR

In your PR title you should include the name and pull request number of the PR which this documentation links to in the following format:

```
[PR]: SidingsMedia/<repo>#<pr number>
```

This helps us track which documentation is about which contribution. Notice the `[PR]` prefix, this helps us differentiate a contribution pull request from the general `[FEATURE]` and `[BUG]` which can also appear in the docs repository. These two prefixes are used for when a PR does not relate to a contribution in another repository.

Describing your PR

You should give a short description of what you have changed. There is no need to repeat what you put in your contribution PR here, all we need is a short description of what you have changed or added. You should also add a link to the PR in the other repo here as well. You should check out linking pull requests from other issues[21] for more information on how to do this.

Checks

As with all other pull requests, you must make sure that your PR does not break any checks. The most common issue you will have with breaking the checks is if you have misspelled any words in your documentation. It may be that you have actually misspelled some words in which case you should go back and fix your errors, you can see which words are marked as incorrect by viewing the workflow output[22]. It may be the case that the words marked as errors are not errors but technical language that is not included in the standard dictionaries. In this case you should add the word to one of the dictionary files in the `dictionaries/` directory.

You may also run into issues with reuse compliance. This just means that you have not added the correct licence descriptors to each file. To find out how to do this you should consult the reuse spec[23] for more details on how to make a file reuse compliant.

4.1.5 Pull Request Guidelines

4.1.5.1 Creating your PR

For information on creating a pull request you should check out the GitHub documentation on creating a pull request[24]. This will give you the basics on how to create a PR from your branch to the repository.

[21] https://docs.github.com/en/github/writing-on-github/working-with-advanced-formatting/autolinked-references-and-urls#issues-and-pull-requests

[22] https://docs.github.com/en/actions/monitoring-and-troubleshooting-workflows/using-workflow-run-logs#viewing-logs-to-diagnose-failures

[23] https://reuse.software/spec/#copyright-and-licensing-information

[24] https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request

Naming your PR

Your PR name should be in the following format

```
<prefix>: <issue number> <issue name>
```

The prefix should be one of `[FEATURE]` or `[BUG]`. These are the same as when *creating an issue*. This prefix should be followed by the issue number of the issue you created when you proposed your feature. This is followed by the title of that same issue. In the following example we are creating a pull request for the feature entitled `Update AJAX requests to use fetch()` with an issue number of

```
[FEATURE]: #3 Update AJAX requests to use fetch()
```

By referencing the issue number it allows GitHub to automatically link the issue to the pull request making tracing the origin of features far easier. The repetition of the issue title in the PR title makes it easier to manually link the PR to an issue. Note that if you are creating a pull request for documentation that corresponds to a feature you are adding in a separate repository you should refer to the *Creating a documentation pull request* section of *Documenting your contribution*.

Describing your PR

When you describe your PR to us, you should give us both a technical description of how your feature works as well as giving us a more user friendly description of the feature. You should make sure to describe any UI changes that you have made, if applicable and should also make any corresponding changes to the documentation. See *Documenting your contribution* for some more information on how to create and submit documentation for your contribution.

Describing your tests

This is vital to getting your PR across the line. If you can't prove that your contribution works and that it doesn't break anything else then we won't be able to accept your contribution. When describing your testing you should include your test configuration. Things like operating system and node version are good things to include here if they are relevant. You should include any relevant outputs that show the feature working, screenshots and animated GIFs are great here. You should also confirm that all existing tests complete as normal and detail any new tests you have added to the test suite.

Referencing your issue

As well as referencing your initial issue in the PR title, you should also reference it in the body of your PR. Something like this is fine:

```
Added #3
```

It can either be `Added` or `Fixed` depending upon whether the PR is for a feature or a bug.

4.1.5.2 Work in progress

When you are working on your addition to the project, it may be helpful for you to create a work in progress pull request. This allows us to more easily see who is working on what features and it makes it easier for us to give you a hand if you need any help. When creating a work in progress PR there is no need to fill out all of the details in the template. These only need to be filled in when you mark the PR ready for review. You should replace the `[FEATURE]:` or `[BUG]:` prefix with a `[WIP]:` prefix. You should also mark the PR as a draft. You can do this whilst creating a pull request[25] or by converting a pull request to draft[26]. When you have finished making your changes then you should mark the PR as ready for review and change the title prefix from `[WIP]:` to one of `[FEATURE]:` or `[BUG]:`.

4.1.5.3 Checks

As with all other pull requests, you must make sure that your PR does not break any checks. The most common errors that you will probably run into when working on the project are issues with reuse compliance. This just means that you have not added the correct licence descriptors to each file. To find out how to do this you should consult the reuse spec[27] for more details on how to make a file reuse compliant.

4.2 Logos

All logos for the SMRC project are located in their own repository. This is to help with keeping the logos up to date across the entire project and allows them to be imported into other repositories via submodules. The logos can be found in the SMRC-logos[28] repository.

4.2.1 Using the Logos

4.2.1.1 Adding the repository

Using the logos in the SMRC-logos repository is fairly simple. All you need to do is add the repository as a submodule in your projects repository using the following commands:

```
$ git submodule add https://github.com/SidingsMedia/SMRC-logos
```

You can then commit your changes and start using the logos.

More information about git submodule and instructions on some advanced features can be found in Git's documentation[29].

After you have added the logos repo as a submodule you can reference the logos in your code, just use the path to the logos directory within your project.

[25] https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request
[26] https://github.blog/changelog/2020-04-08-convert-pull-request-to-draft/
[27] https://reuse.software/spec/#copyright-and-licensing-information
[28] https://github.com/SidingsMedia/SMRC-logos
[29] https://git-scm.com/book/en/v2/Git-Tools-Submodules

4.2.1.2 Updating the repository

To update your submodule to the latest commit you can run the following command:

```
$ git submodule update
```

After you have updated the submodule you will just need to commit it and then you will be able to use the latest logos.

4.2.2 Directory structure

The logos repository uses the following directory structure:

```
SMRC-logos/
├─ .github/
├─ .reuse/
├─ LICENSES/
├─ rendered/
```

The first three files are used for repo administration. They are the same as in all of the other official SMRC projects. The `.github/` directory contains configuration files for functions such as workflows. The `.reuse/` directory contains a dep5 file that covers all of the logos and other images without having to add a `.licence` file for each image. `LICENSES/` contains all of the licences used in the repository.

All of the SVG files should be placed in the root of the project. The `rendered/` directory contains all rendered versions of the SVG files in the document root. The files should be named in the following format:

```
<name>-<size>.<format>
```

Here is an example for a logo with the filename `track-bw-square.svg` rendered at a size of 192 pixels by 192 pixels in the portable network graphics (PNG) format.

```
track-bw-square-192x192.png
```

There are a few exceptions to this rule, especially regarding favicons. Favicon files should not follow the rule defined previously and should instead be named in the form `favicon.<format>`.

4.3 Desktop Client

The desktop client for the Sidings Media Railway Controller project is build using the electron framework with react for frontend functionality. It uses typescript where possible and aims to use an object oriented style where feasible.

4.3.1 Directory structure

The project uses the following directory structure:

```
SMRC-Desktop-Client/
    ├─ .github/
    ├─ .reuse/
    ├─ config/
    ├─ LICENSES/
    ├─ main-process/
    ├─ public/
    ├─ scripts/
```

(continues on next page)

(continued from previous page)

```
    ├─ src/
    │   ├─ styles/
    │   ├─ ui/
```

The first two directories are for project administration. They contain config files for managing the repository. The `config/` directory contains configuration ejected from create-react-app. This config controls things like building the REACT files and running the development server. The `main-process/` directory contains the source files for the electron app and the main process. Files like the preloader and main entry point are located here. `LICENSES/` contains all the licences used in this repository. They have been downloaded using the reuse tool. `public/` contains all of the files you want to be accessible to REACT such as images. They will be copied into `build/` when the react files are built. This directory contains all of the files created by create-react-app. `scripts/` contains all of the scripts referenced by package.json. This includes the scripts that were ejected by react-scripts. `src/` contains two sub-directories, `styles/` and `ui/`. `styles/` contains all of the styles for the react components. `ui/` contains the react components themselves.

4.3.2 Building

To build app from source run `yarn build`. You can also build the electron code and REACT code separately by running `yarn react:build` and `yarn electron:build` respectively. The built files are output to the `build/` directory. REACT files are in the root of the `build/` directory with the electron files being located in the `build/electron/` sub directory. Note: if you are building react and electron separately, react must be build first as it clears the directory. If electron is built first any files that were created will be deleted when react is built.

4.3.3 Running

To run the app in development mode use the command `yarn electron:dev`. This will start the REACT development server on `localhost:3000` and start the electron app. It will also monitor the electron files for changes and reload electron if any are detected. There is no need to build the app to run it in development mode as the electron files are built automatically. The REACT files are not built as they are served by the development server.

4.3.4 Packaging

Packaging the app can be done using the scripts available in package.json. Simply use the command `yarn electron:package:<platform>` where `<platform>` is one of `mac`, `win` or `linux`. There is no need to build the project before running these commands as this is done during packaging. The compiled binaries will then be available in the `dist/` directory. The unpacked files are located in `dist/<platform>-unpacked`. An installer is also provided in the `dist/` directory.

4.3.5 Desktop Client Preload API

A number of API endpoints are made available to the render process by the preloader script. These endpoints can be called by any script running in the render process. They expose various methods and data normally unavailable to the render process. To find out more about preloaders and the render process as well as context isolation in electron then see the electron Context Isolation[30] tutorial. All of the API endpoints are created under the window context and are defined in `src/global.d.ts`.

[30] https://www.electronjs.org/docs/latest/tutorial/context-isolation

4.3.5.1 window.control

This API handles all of the window control functions such as closing the windows and alerting when the window comes into or out of focus. It is primarily used on Windows based devices by the titlebar although the API is still active on other operating systems.

Methods

The `window.control` API has the following methods.

window.control.registerIPC(channel, callback)

- **channel - A string that represents the IPC channel. This should be the** same as the ctrlChannel the Window class was initialized with.
- **callback - A function that is called when an IPC message is received** from the main process.

This method registers the IPC handlers for window control features. It also sets the channel name that the API should use when sending requests to the main process. If this method is not called then features such as detecting when the window goes in and out of focus or when it is minimized or maximized will not work and requests sent to control the window will be sent on the `win-ctrl` channel which may or may not have any handlers registered and may cause errors in the main process as well as the render process. If the `window.control` API is to be used this method should be called as soon as possible after page load.

window.control.openDevTools()

This method opens the developer tools on the calling window. It is equivalent to running `this.win.openDevTools()` in the main process.

window.control.closeDevTools()

This method closes the developer tools on the calling window. It is equivalent to running `this.win.closeDevTools()` in the main process.

window.control.toggleDevTools()

This method toggles the developer tools on the calling window. It is equivalent to running `this.win.toggleDevTools()` in the main process.

window.control.close()

This method closes the calling window and if all other windows are closed will exit the application. Note: this behaviour does not occur on macOS based devices. It is equivalent to running `this.win.close()` in the main process.

window.control.minimize()

This method minimizes the calling window. It is equivalent to running `this.win.minimize()` in the main process.

window.control.maximize()

This method maximizes the calling window. It is equivalent to running `this.win.maximize()` in the main process.

window.control.unMaximize()

This method restores the calling window. It is equivalent to running `this.win.unmaximize()` in the main process.

window.control.toggleFullScreen()

This method toggles full screen the calling window. It is equivalent to running `this.win.setFullScreen(true)` or `this.win.setFullScreen(false)` in the main process.

Properties

The `window.control` API has the following properties.

window.control.platform

This is a string representation of the current platform that the application is running on. It is equivalent to `process.platform`. More information on `process.platform` can be found in the process[31] documentation available from NodeJS.

4.3.6 Main Process Classes

A number of classes are made available to the main process of the desktop client. These can all be imported into the main process file by using a standard typescript import.

Detailed information on each class along with a description of it's methods and parameters can be found below.

4.3.6.1 Window

This class is responsible for creating and controlling each window of the application. It is important to note that this class does not extend the electron BrowserWindow class. More information on why this shouldn't be done can be found in electron issues #23[32] and #8898[33]. It instead creates an instance of the BrowserWindow class it's self and controls that. There are a number of public methods made available by the class as well as a few public properties.

[31] https://nodejs.org/api/process.html#processplatform
[32] https://github.com/electron/electron/issues/23
[33] https://github.com/electron/electron/issues/8898

Constructor

new Window([windowSettings], ctrlChannel)

- windowSettings - An object that provides the settings for the BrowserWindow instance. The object may have any of the properties specified in the BrowserWindow options[34] documentation. Note that another property has been added to the windowSettings object. This is detailed below:
  - url - A string with the URL from which the window is to load it's initial content. Required
- ctrlChannel - A string that specifies the IPC channel to use to send and receive window control messages.

Instance Methods

openDevTools()

Opens the window's chrome developer tools.

closeDevTools()

Closes the window's developer tools.

toggleDevTools()

Toggles the window's developer tools. If they are not open this method will open them else if they are open they will be closed.

getFocus()

Returns `Boolean` - Whether the window is in focus

close()

Attempts to close the window. This process can be prevented by creating an event listener that prevents the default action. For more information on this see the BrowserWindow Event: Close[35] documentation available from the Electron documentation[36].

minimize()

Minimizes the window.

[34] https://www.electronjs.org/docs/latest/api/browser-window#new-browserwindowoptions
[35] https://www.electronjs.org/docs/latest/api/browser-window#event-close
[36] https://www.electronjs.org/docs/latest/

maximize()

Maximizes the window. This method also emits the `maximize` event as a work around for a bug that is present on linux systems. See Electron issue #28699[37] for more information on this bug.

unMaximize()

Unmaximizes the window. This method also emits the `unmaximize` event as a work around for #28699[38].

isMaximized()

Returns `Boolean` - Whether the window is maximized.

toggleFullScreen()

Toggles the window full screen.

Instance Properties

There are no publicly available properties for the *Window* class.

[37] https://github.com/electron/electron/issues/28699
[38] https://github.com/electron/electron/issues/28699