Streetmix

Sep 23, 2023

Contents

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

Hi there, and welcome to Streetmix documentation! This documentation contains contributor guidelines for code, illustrations, translations, data, and more. It also contains helpful tips and reference information for street planning and civic engagement.

Attention: This is a work in progress as we migrate miscellaneous documents into this space. If you'd like to pitch in, check out our guidelines for contributing to documentation (*Documentation*)!

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

CHAPTER 1

Contributing

Important: The Streetmix team has adopted a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

In the spirit of open source, **everyone** is welcome to help! **You do not have to be a programmer to participate.** This section will help you make high quality contributions to Streetmix. Choose the activity you'd like to help with to learn more:

- I want to create illustrations.
  - *Illustrations*
  - *Reference library*
- I want to design the user interface and experience.
  - *User interface*
- I want to write or improve documentation.
  - *Documentation*
  - *Content styleguide*
- I want to translate.
  - *Translations*
  - *Content styleguide*
- I want to write code.
  - *Code contributions*
  - *Code styleguide*
- I want to give money to Streetmix.
  - Support us on OpenCollective.

  - For partnerships and business inquiries, email hello@streetmix.net.
- I want to be a part of the community, or find other ways to help!
  - Join us on Discord or the forums.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.1 Content styleguide

Streetmix is a tool for communication and collaboration, so the way we write and present content should facilitate those goals. This is a reference that will continue to grow as needed. When you run into a situation that isn't covered here, refer to the very excellent 18F Content Guide.

1.1.1 Basic guidelines

- Use plain language and simple sentence structure.
- Be funny or friendly, but:
- Choose clarity over cleverness.
- Avoid technical or industry-specific jargon.
- Content isn't frozen in time. Always be refining. As we learn, we change and adapt.

1.1.2 User interface

Sentence case

When writing standalone text such as titles, labels, or headings, use sentence case. With sentence case, capitalize the first letter of a phrase, but leave all other words in lowercase, unless they would normally require capitalization, such as proper nouns.

Avoid title case (Where Each Word Is Capitalized, Like This), or uppercase (SUCH AS THIS). These approaches are harder to read than sentence case.

1.1.3 Technical writing

People have different levels of expertise

Avoid using the words "simply" or "just" when writing instructions.

Don't write this:

> Simply install your packages by typing `npm install`.

What may be second nature for you may be foreign to another person. These instructions are missing additional information for a less experienced developer: Where would you type this? What is `npm`? What happens if the command doesn't work?

At the same time, instructions should be short and to-the-point, so we would not strive to add context to every instruction. We should state the instruction but not create a value judgment about its relative ease.

Instead, write this:

> Install your packages by typing `npm install`.

Those are not the only two words to avoid. For more, read Words To Avoid in Educational Writing from CSS Tricks.

1.1.4 Specific words and phrases

- **Streetmix** is one word, and the middle **m** is always lowercase.
- **Code for America** can be abbreviated **CfA**, after the full name has been mentioned at least once.
- **GitHub** always capitalizes its middle **H**.
- **NEW INC** is always in uppercase.
- **open source** is never hyphenated.
- **email** is one word. Do not hyphenate or capitalize (unless at the start of a sentence).
- The URL to our website is **https://streetmix.net/**, which uses the more secure **https** protocol, and does not have the `www.` prefix before the domain name.

> **Attention:** Additional guidance for text used in the Streetmix interface that are not likely to occur elsewhere in our written content will be provided in our translation system.

1.1.5 Typography

Consistent typography, based on standard practices in the typography industry, is another aspect of how we pay close attention to detail within the Streetmix project.

- Use curly quotes instead of straight quotes.
- Use the ellipses character `…` rather than a series of periods.
- Use the en dash and em dash characters appropriately.
- There's only one space after a period, never two.

Typography practices may differ between languages and cultures. For instance, in Chinese, periods use a small circle 。 instead of a small dot, and ellipses are written as six dots (two ellipses characters side by side) `……`, but in computer UI, the ellipses in menu items remain the single ellipses character. When working on translations of Streetmix, cultural and contextual differences in typography should be considered by translators and should be consistent throughout the application.

1.1.6 Units of measurement

By default, use the metric system. We support the imperial system only in the United States for users who are more familiar with it.

> **Note:** When writing distances in the imperial system, feet and inches should use the prime ′ and double prime ″ marks respectively, instead of the straight quotes `'` and `"`. When processing input, both quote and prime marks should be accepted as valid.

Distance

For distance measurements, include a space between the quantity and unit. For example, fifty meters should be expressed as `50 m`, not `50m`.

Vernacular exceptions

- In **Russian**, the Cyrillic is more commonly used instead of `m`.
- In **Arabic**, the Arabic letter is more commonly used instead of `m`, and measurements remain in the right-to-left content direction.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.2 Code contributions

Warning: This section is a work in progress.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.2.1 Code styleguide

You've heard it all before: code style should be consistent across a project, which helps readability and maintainability. We've done our best to automate formatting or linting of code to keep things consistent. However, we still need to set some guidelines and document some of our tooling and decision-making.

The first thing you should know is: *two-space indents everywhere; no tabs.*

Code style

HTML

We don't have anything to lint HTML, so here are some basic guidelines:

- All elements and attributes should be lowercase.
- All quotation marks should be double quotes.
- We use Handlebars templating. In the past, we've tried using templating language that didn't look like HTML, and, long story short, we won't be doing that again.
- Comments can also be written in Handlebars so they don't get sent to the client.
- Indent nested elements.

CSS

We use Sass in the SCSS flavor, which makes it easier to interoperate with standard CSS syntax. Code style is enforced with Stylelint, using a combination of stylelint-config-standard and Prettier rules.

SCSS files generally live adjacent to the components that they apply to, and will be imported by each component. SCSS files are not global, so variables and mixins need to be imported from other SCSS files. However, the compiled CSS will be global, not scoped to the component. *Don't be afraid of the cascade!*

Avoid styling elements using `id` attribute selectors. Instead, use class names (alongside pseudo-selectors and attribute selectors, if necessary). Namespace all classnames. If there's a component called `palette`, a good class name could be `palette-container`. Try to avoid writing generic class names like `large`. Because styles are not scoped, allowing generic class names without namespaces will inevitably cause collissions.

In the future…

We may make a stronger move toward Tailwind CSS-style composition of utility classes instead of relying on Sass mixins, extends, or copy-pasting duplicate CSS code across class names. Generic class names will then be used for utility classes.

Although it's popular, we do not use BEM framework/naming convention. Some good resources which have generally informed our approach to CSS organization (but have not dictated it) include Scalable and Modular Architecture for CSS (SMACSS) and parts of the Reasonable System for CSS Stylesheet Structure (rscss).

JavaScript

Our JavaScript code style is Standard JS. We enforce code style with a combination of Prettier and ESLint.

Import order

Generally, external dependencies are imported first, then relative imports local to the application. Developers familiar with Python's PEP8 guidelines will have seen this concept before.

When importing relative files, files in the local directory ("closest" to the current module) are usually imported first. Files that come from the same subdirectories elsewhere are grouped together. When in doubt, alphabetical ordering within groups is better than no order at all.

Here's a rough order of imports:

- import `react`
- import `prop-types`
- import other React packages (usually related packages are grouped)
- import other dependencies (usually related packages are grouped)
- import constants
- import components
- import functions from non-component modules not covered elsewhere
- import Redux-related modules (e.g. action creators) last
- import CSS

There's no technical reason to order imports. (In the past, some modules may have created side-effects when they load, meaning that import order was significant. This should no longer be the case. If you come across one, consider this a bug which should be fixed ASAP.) We order imports to make it easier for a human to understand the dependency graph of a module.

React

We extend the Standard JS ESLint rules with the eslint-config-standard-react package.

> **Attention:** We have only overridden one rule, `jsx-quotes` (link), to prefer double quotation marks in JSX attributes. This is because, unlike regular JavaScript, double quotation marks are *more* typical in HTML, and this convention has carried over to JSX. You can see single quotes in JavaScript and double quotes for JSX attributes coexisting in React documentation, and we believe developers expect this to be typical across the React ecosystem. This is a rare instance where we disagree with Standard JS's rule.

Additional guidelines

- **Prefer functional components.** We'll let React developer Dan Abramov do the talking (and the writing). Refactor existing class components to functional components only when it's cheap to do so.
- **Lifecycle ordering.** Refer to the Airbnb JSX guide for guidance.
- **State variable naming.** A variable that stores UI state as a boolean value should be named with the pattern `is[State]`. For instance, that's `state.isVisible === true`, and not `state.visible === true` Or `state.isEditing`, not `state.editing`.

Code comments

TODO: JSDoc

Commit style

We like the Conventional Commits specification. This commit style helps us organize our changes into discrete commits by documenting them in a standard way, which helps us understand project history over time. (Previously, this is also known as semantic commit messages or the Angular commit style.)

While the Conventional Commits specification only defines the `feat` and `fix` types, we also use the following **types**:

- **chore**: Changes to packages, configuration, external services such as CI (continuous integration) that do not affect the Streetmix application itself
- **docs**: Changes to documentation contents
- **style**: Changes to code style (white-space, formatting, etc)
- **test**: Changes to tests
- **perf**: Improvement to existing code that improves performance
- **revert**: Reverts a previous change
- **refactor**: All other improvements to existing code (and not `perf`, `fix`, or `feat`)

A **role** can also be added optionally. For React components, the role is often the name of the component:

```
refactor(Avatar): stop using internal state
```

This is flexible and new types may be adopted over time. Sometimes a commit may seem to fall into one or more category. The first thing to consider is whether the commit is too large and should be split into smaller commits. If not, then pick which category seems most relevant. Make your own best judgment call here. Pull request reviews should not be held up on semantic debates of commit types, but a review should note if commit types are completely missing or very clearly used improperly.

Note: We use commitlint to automatically check your commit messages for validity. If they're not valid, the check will fail. This check is hooked into each commit, and our CI test will also check your commit messages.

Tip: If your development style is to make quick, small commits whenever you've made progress, there's no need to change your development workflow right away to adopt the Conventional Commit style. You can always clean up the commit history on your branch when you're ready to make a pull request. You can use git rebase to do this (tutorial).

In the future...

...we may use commit messages to help with automated releases.

GitHub issues

Whenever possible, use commit messages or commit comments to close automatically close GitHub issues. (This may also be done in pull requests.)

```
refactor(Avatar): stop using internal state, resolves #1337
```

Other stuff

- Keep code concise, but consider readability. Resist the urge to play code golf.
- Avoid abbreviating variables. They should be readable.
  - It is acceptable to use single-letter variables as counters in loops (e.g. `i`).
  - For event handlers, the variable name `event` is preferred, but sometimes you will see it abbreviated `e`, which is common in JavaScript. However, never use the abbreviated form if the event handling function is not in the context of the event listener. For instance, `window.addEventListener('click', (e) => {})` can be acceptable, but `export function doSomething (e) {}` is not immediately clear that the function should only be used as an event handler.

If you have the capacity to write code, and wish to contribute to Streetmix, we'd love your help! **No patch is too small:** we welcome patches to fix typos, add comments, or refactor code.

We use GitHub Flow to make changes and proposals to our codebase.

Tip: Before writing any new feature or code, it's best to have a brief conversation with the Streetmix team to make sure we're all on the same page. Please check GitHub issues to see if a feature is already being discussed or work on, and comment there to register your interest. If it hasn't been discussed, please feel free to open a new issue for it. You

can also join our Discord server and talk with us there. A little bit of communication will go a long way, making it more likely that we're able to accept your work when it's ready!

1.2.2 Submitting a pull request

1. **Fork the project**, if you do not already have write access to the repository. Individuals making significant and valuable contributions will be given write access.
2. **Create a new branch.** Changes should always be made in a new feature branch. The branch should be named in the format `username/feature-name`.
3. **Implement your feature or bug fix.** Writing code is the fun part! Before you start any work, it's a good idea to talk to the team first, so that we can be sure you're on the right track.
4. **Commit your changes.** Commit messages should follow semantic commit message format (See *Commit style*). When committing, we use hooks to run code style linting. If it fails, please correct (or override) the code and commit again.
5. **Push your changes.** After pushing, we run continuous integration tests in the cloud to make sure commits pass. We recommend manually running tests locally as well. See *Running tests locally*.
6. **Submit a pull request.** Ideally, pull requests contain small, self-contained changes with a few commits, which are easier to review. You can simplify a review and merge process by making sure your branch contains no conflicts with the `main` branch and is up-to-date (either by rebasing on `main` or merging it in) when the pull request is created. If the pull request addresses an open issue, be sure to reference it in your request's title or description.
7. **Wait for a review.** A project maintainer will review your pull request and either approve, reject, or request changes on it. A well-written, small pull request that fixes an open issue is most likely to be approved and merged quickly. Once merged, a branch is deleted.

Stale branches, especially ones that cannot be cleanly merged anymore, are likely to be deleted after some amount of time has passed.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.3 Documentation

You're looking at it!

Our documentation lives in the `docs` folder of the Streetmix repository. It's built and hosted by Read the Docs.

We use the reStructuredText (reST) syntax. While Markdown is a very common alternative, it's harder to format extensive technical documentation with it.

Important: Not all technical documentation lives here! Documentation of specific functions or components should be written in the source code itself. Documentation relating to a directory of related modules should live in a dedicated Markdown `README.md` file coexisting with those files. This helps keep narrowly-focused documentation up-to-date and easier to find.

Technical documentation pertaining to cross-cutting concerns or high-level architecture do belong here!

For guidance on writing documentation in source code, see *Code styleguide*.

1.3.1 Local development setup

You don't need to build documentation locally when writing code for Streetmix. However, it *is* a good idea to document what you're working on, so we do recommend writing and updating documentation. Here's how to set up a local development instance of the documentation so you can preview any changes.

1. Install dependencies

First, make sure you have a working Python development environment on your system. Installing Python is outside of the scope of this guide.

Install Sphinx, the Read the Docs Sphinx theme, and extensions.

```
$ pip install sphinx sphinx-prompt sphinx_rtd_theme
```

2. Build documentation

Documentation must be built from the `./docs` working directory.

```
$ cd docs
$ make dirhtml
```

Note: The directory HTML renderer will create URLs that match the path structure that we use on Read the Docs.

Alternatively, we've provided an `npm` package script that can build documentation from the root directory.

```
$ npm run docs:build
```

In the future…

We would like to develop a watch and livereload system that automatically rebuilds documentation locally when contents change. For now, you must manually run a local build whenever you make changes.

3. Preview

Run a static file server, such as `http-server`, to preview the built documentation. Built files are located in `./docs/_build`, and the example command below assumes you are in the `./docs` working directory.

```
$ http-server _build/dirhtml
```

You may use any static file server solution you wish. We've also provided an `npm` package script that assumes `http-server` is available on the system's global packages and will automatically serve the documentation locally at `http://localhost:8080/`.

```
$ npm run docs:serve
```

Attention: A static file server does not automatically watch and rebuild changed files. You must manually rebuild files and then reload your browser to see the changes.

4. Upload

Commit your changes and push to the upstream repository.

```
$ git add .
$ git commit -m 'docs: short commit message [skip ci]'
$ git push origin
```

5. Deploy

Once documentation have been committed to the Streetmix `main` branch, Read the Docs will automatically build and deploy the revised documentation to https://streetmix.readthedocs.io/. Read the Docs does not wait for continuous integration to pass, and a production build will be triggered on each commit.

> **Attention: (August 2021) Streetmix documentation has now moved!** Please update your bookmarks to https://docs.streetmix.net/.

1.4 Illustrations

If you would like to contribute illustrations, or have feedback on existing illustrations, please talk to us on Discord or on the forums!

We collect illustration inspiration and reference material on Pinterest. Give it a look!

> **Attention: (August 2021) Streetmix documentation has now moved!** Please update your bookmarks to https://docs.streetmix.net/.

1.4.1 Reference library

Here is an overview of many (though not all) of the illustrations created for Streetmix so far, for reference.

> **Warning:** This section is a work in progress.

bikes

buildings

construction

dividers

furniture

lamps

parklet

people

plants

transit

trees

vehicles

wayfinding

1.4.2 Design principles

Streetmix illustrations use the following principles to achieve its simple, unique style.

Simplified forms and reduced details

The complexity of form is reduced to a simple silhouette and each object contains minimal details. Good rule of thumb: if you don't need a particular detail to identify the item, don't add it. This reduction helps each illustration scale to small sizes.

Nope. The extra details aren't visible at a small scale and the more complicated silhouette makes the car look clunky.	**Yes!** This silhouette has less vertices, and extraneous details are removed or simplified.

Sharp angles

All of the illustrations are comprised of sharp angles, with a minimal use of perfect circles.

Nope. Don't draw rounded corners or bulby shapes.	**Yes!** Edges should meet at sharp angles.

Few orthogonal lines

There are very few entirely straight vertical lines. All of the objects slant in or out in a trapezoidal way, unless it makes the illustration look too funky.

Nope. Avoid rectangular shapes.	**Yes!** Shapes should be trapezoidal.

Nope.	**Yes!**

Lines that appear to be exactly vertical or horizontal will not actually be precisely level or plumb. This isn't an engineering drawing. Lines can be slightly "off," with some exceptions.

One example of such an exception is for building walls. Since buildings can be multi-story, lines that continue from one floor to the next must be vertically straight so that each floor aligns properly.

Recurring elements

Shapes that recur throughout illustrations helps everything read as part of the same unified family. However, these should be subtle—don't overdo these, or force them into existence. Use them when they make sense.

45-degree angles

Some angled elements can be drawn at exactly 45 degrees. These work well for window reflections and shadows.

The "brick"

This is a simple trapezoidal shape of similar size and proportion that can be used as a small object, detail, or addition to a larger shape.

The "platter"

Similar to the Brick, the Platter is a trapezoidal shape but much longer and flatter.

Letters and words

Avoid using actual words or letters in illustrations. Streetmix is used all over the world, and we want to avoid making illustrations that are English-only (or specific to your preferred language).

One notable exception to this is the use of numbers or letters as transportation route identifiers.

Brands, logos, and other recognizable features

Similarly, avoid specific brands and logos, whether that's commercial businesses (like transportation network companies or chain restaurants) or public agencies (like transportation agencies). However, some illustrations are clearly homages to actual elements that come from specific parts of the world, like New York City's wayfinding pylons, or San Francisco's deep-red "Yerba Buena" parklet. We like that these recognizable elements can be dropped anywhere around the world, but logos tie the elements to its regional source too strongly.

In the future…

…we may be able to relax these guidelines by localizing illustrations to specific regions. For now, prefer abstractions that make an illustration look generic, even if it's an homage to a real place.

1.4.3 Color

Palette

Colors are mostly soft, with hints of brightness. They should have sufficient contrast against a light–medium blue background (the sky).

Colors are not necessarily "true to life." (For instance, we're not using the "actual" color of a taillight.) Use colors from the following color palette whenever possible, instead of using custom colors, so that illustrations appear unified. But if the palette just doesn't work, you may choose a new color that does. Some illustrations may have already chosen a color not shown below, so it's best to sample colors that have already been used elsewhere, even if it's not part of the official palette.

Skin tones

R 85
G 63
B 55

#553f37

R 114
G 89
B 75

#72594b

R 170
G 139
B 126

#aa8b7e

R 206
G 172
B 152

#ceac98

R 225
G 197
B 181

#e1c5b5

To keep people recognizable, use skin tone colors only for people.

Beiges

R 206
G 173
B 128

#cead80

R 224
G 206
B 168

#e0cea8

R 236
G 219
B 177

#ecdbb1

R 246
G 241
B 225

#f6f1e1

Recommended for: walls, stucco, plaster, pavement, sand

Off-whites

R 216
G 211
B 203

#d8d3cb

R 231
G 230

B 222

#e7e6de

R 245
G 243
B 233

#f5f3e9

Recommended for: walls, metal, concrete

Greens

R 53
G 129
B 63

#35813f

R 103
G 154
B 69

#679a45

R 136
G 186
B 106

#89ba6a

Recommended for: foliage, trees, plants

Browns

R 53
G 45
B 39

#352d27

R 100
G 88
B 73

#645849

R 124
G 112
B 90

#7c705a

R 152
G 138
B 116

#988a74

Recommended for: wood, dirt

Grayscale

R 42
G 43
B 42

#2a2b2a

R 92
G 94
B 95

#5c5e5f

R 193
G 190
B 187

#c1bebb

R 228

G 226
B 220

#e4e2dc

R 244
G 242
B 237

#f4f2ed

Recommended for: asphalt, metal, road markings

Indigos

R 37
G 41
B 71

#252947

R 49
G 49
B 86

#313156

R 73
G 74
B 114

#494a72

Recommend for: windows, reflective surfaces

Blues

R 35
G 76
B 107

#234c6b

R 54
G 99
B 135

#366387

R 79
G 131
B 163

#4f83a3

Recommended for: water

Reds / Oranges

R 114
G 14
B 26

#720e1a

R 174
G 32
B 37

#ae2025

R 239
G 156
B 116

#ef9c74

R 255
G 238
B 204

#ffeecc

Recommended for: lights, flowers

Brights

R 231
G 103
B 60

#e7673c

R 248
G 187
B 28

#f8bb1c

R 252
G 209
B 87

#fcd157

R 16
G 155
B 168

#109ba8

Bright colors used sparingly when colors above won't do.

Tip: You can download a palette swatch for use in your favorite graphics editing program.

- Adobe Swatch Exchange (ASE) format can be used in Adobe Illustrator or Affinity Designer.
- GIMP GPL format can be used in open-source image editing applications such as GIMP, Krita, or Inkscape.

Shadows and highlights

Many elements have one solid color with a highlight and a low light. Exceptions include objects with more details, like the bus. Because the bus contains detail elements (lights, windshield, mirrors, route number, etc), adding shading to the metal would make the object too busy. Similarly, details don't need shadows or highlights—they wouldn't be visible at a small scale anyway. All of the shading should be done with one solid color, without feathered shadows or gradients. (The only exception is the sky.)

Nope. The image is over–complicated by the number of shadows and highlights.	**Yes!** The shading and highlighting is used only on the most prominent element, the windshield.

Nope. This image uses gradients to blend out the shadows and highlights.	**Yes!** No gradients. No drop shadows.

Contrast

Some illustrations are smaller in nature, so they require more contrast between adjacent colors to be clearer. For example, the pedestrian's pants need more contrast than the building.

Nope. This looks good as a larger image, but the shadow is not really visible when scaled down.	**Yes!** This actually looks like too much contrast close up, but is just right when scaled down.

1.4.4 Workflow

For instructions on how to work on illustrations, see the instructions in the illustrations repository: https://github.com/streetmix/illustrations

---

Tip: The illustrations used here in this documentation are also located in the `illustrations` repository! If you want to update the graphics used here, please make sure to process the exported SVG files through SVGOMG so that they're optimized, making them faster to load.

You can also see a collection of illustrations in our *Reference library*.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.5 Translations

Warning: This section is a work in progress.

We're translating Streetmix into different languages! As a civic engagement tool, we are constantly improving accessibility to users of different needs, and this includes communities that speak a language other than English. Some of these communities may be in predominantly English-speaking countries like the United States, but we also have users all over the world in predominantly non-English speaking countries, as well.

The bulk of our translation work is contributed by volunteers who are fluent in those languages. If you would like to join us, please review this translator guide, join our Discord, and send us an email.

We are always on the lookout for new translators!

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.5.1 New translator guide

Warning: This page requires a complete overhaul.

Welcome to the Streetmix translation team! Your contributions are invaluable in making Streetmix accessible to non-English-speaking countries around the world, and for communities that speak languages other than English. Let's help you get started!

Overview

Translators do not need programming expertise to translate Streetmix! We use a few tools and terms that translators will become familiar with:

- We use Transifex to manage different translations, or *locales*. Translators will use this interface to translate English text, or *strings*, into strings of another language. We call this translation process *localization*.
- We use Discord for real-time communication between translators at Streetmix team members. Whenever you have a question about how a string is used, or how it should be translated, this is the place to get answers.
- We use a *staging server* with a test instance of Streetmix in order to preview translated strings in real time. Whatever changes you make in Transifex should instantaneously appear on the staging server, allowing you to see your translation in context.

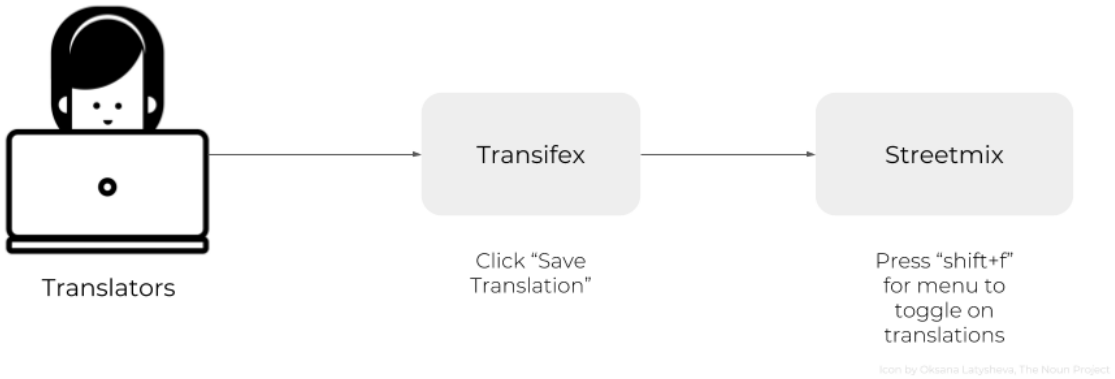
Localization is an iterative process. We may go through several rounds of iteration to ensure that localizations meet our high standards for quality. Once complete, we will turn on that language, or *ship* it, to our production server for all users of Streetmix.

We'll take a closer look at each tool below.

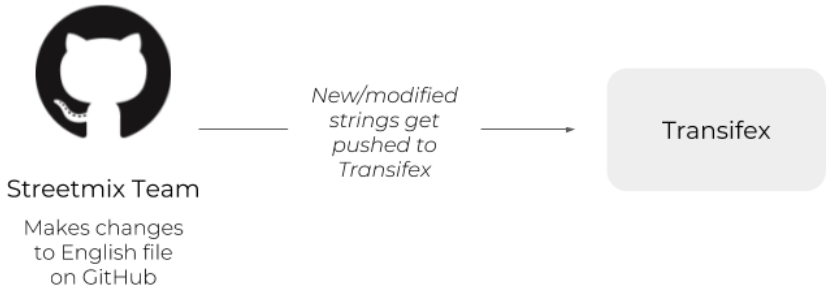
How it works

We manage translations through the Transifex platform.

The staging instance of Streetmix is constantly communicating with Transifex. When you finish a translation in Transifex, Streetmix fetches it and displays it.



Sometimes, there are updates or additions to the original, or *source*, English strings. When this occurs, the Streetmix team will make changes to the English files on GitHub. Transifex updates its source strings from GitHub twice a day.



Seeing translations live

The most important part about translating is seeing your hard work in Transifex show up live in the application! You can do this by visiting the staging server at http://streetmix-staging.herokuapp.com and toggling on the locales feature:

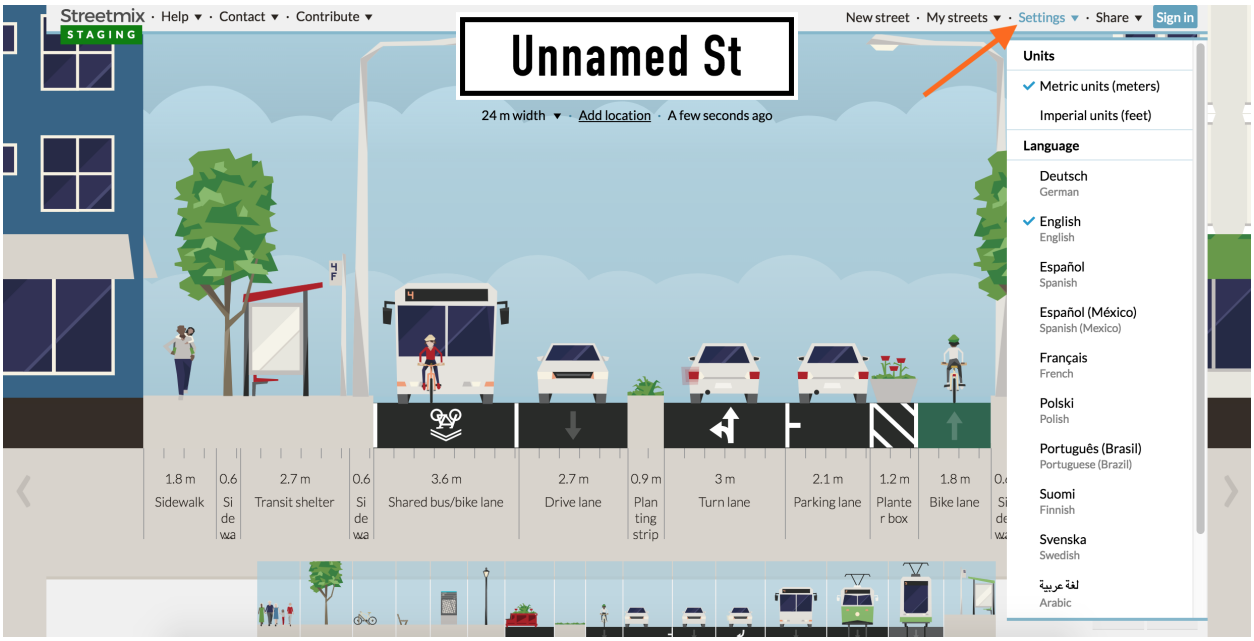
Once at the staging server, you can toggle on your language by doing the following:

- **Press "shift-f"** (make sure the app is focused and no text box or developer tool window is selected). This displays a dialog box with some checkboxes. *(Note that this is just intended for testing and viewing your work, so please don't share it publicly.)* Choose which box to check based on the locale descriptions below.

We have 3 levels of locales (languages). Levels are lower bounds and inclusive. In other words, level 1 means all locales 1 or higher are enabled. You cannot only turn on level 1 but not level 2 for example. Based on which languages you want to see, check one of the following boxes:

- **Level 1+ (in progress)**: Incomplete or in progress, available for translators to test, but not for end-users.
- **Level 2+ (complete, in testing)**: Complete or nearly complete, available for quality assurance & feedback with small group of users.
- **Level 3 (complete, final)**: Finished and shipped to production, for public use.

Once you've checked the appropriate locale box, you can change to your language via the **"Settings" dropdown menu** in the upper right.



This is a really good way of visualizing how your translated text works in the real application. Maybe in the editor it's not clear whether something should be capitalized or how long a text string can be, or maybe it's not clear what the proper translation should be. *Seeing it in Streetmix directly provides context that allows you to go back and clarify any translations.*

Also, if a translation is expected to show up, and it doesn't, please make a note of it (*see how to report problems below*). It might be that the application is not hooked up to the correct translation yet. Or it might be that the text showed up in another part of the application you didn't expect and is hard to get to (like an error page).

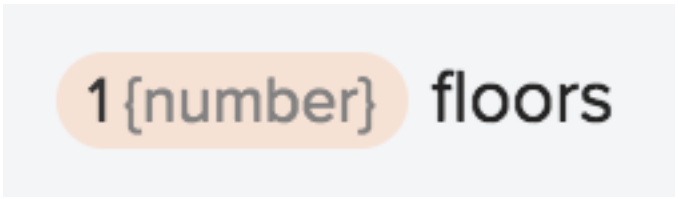
Glossary

Before you get started, here are some terms that you should get familiar with.

Issue: This is when there is a problem/error with one of the original English strings. When you add an issue, the project administrator will get a notification.

Key: Keys are unique identifiers for each string. An example is: buildings.wide.name.

Placeholders: These are what we use to store parts of a string that don't need to be translated. All you have to do for these is when you see one, click on it and it will be inserted into the translation in the appropriate place. Here is an example of one:



Resource: We have 2 main resources: the Main application UI (main.json file) and Segment info (segment-info.json file). Resource is just a fancy word used to indicate which of these files a string is in.

Strings: These are chunks of text in the user interface that we need to translate. "My streets" (a menu button in the top right corner of Streetmix) is one example of a string, and "Sidewalk with a lamp" (a segment) is another.

TM: TM stands for "translation memory". If you see "TRANSLATED BY TM" in the Transifex interface, that means the string was auto-completed by Transifex because it identified an identical string that was translated before.

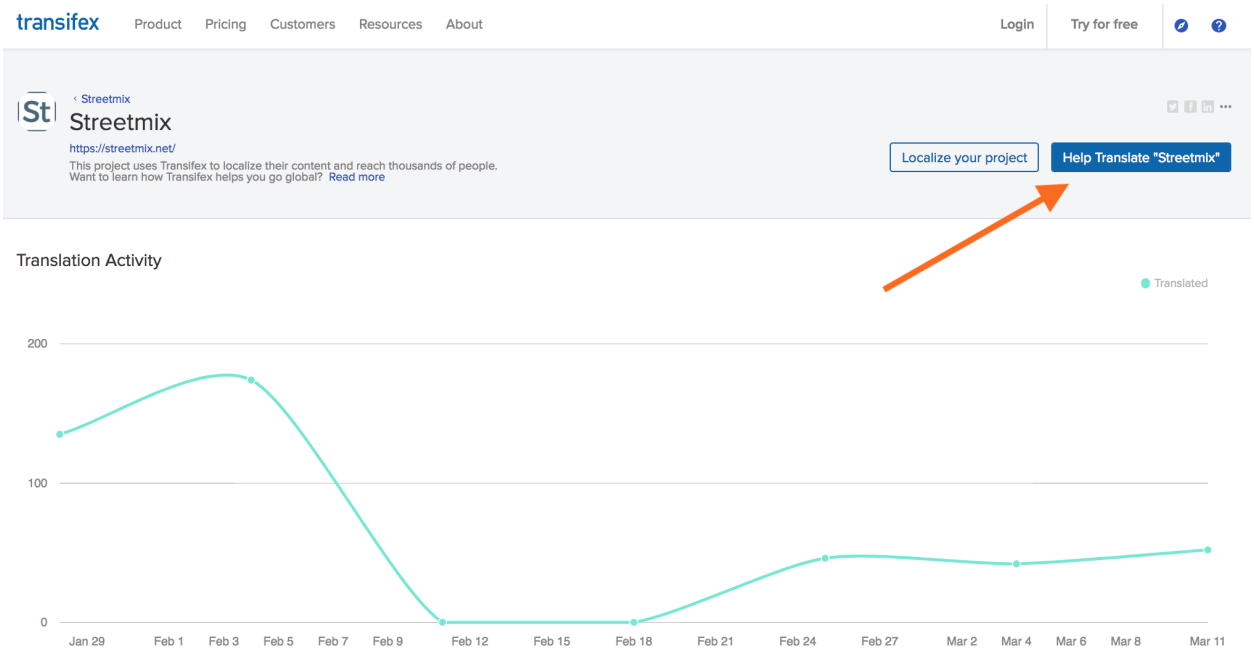
UI: UI means "user interface". This refers to what people see when they use an app. You will be working in the Transifex UI and your translations will show up as a finished product in the Streetmix UI.

Using Transifex

Join the team

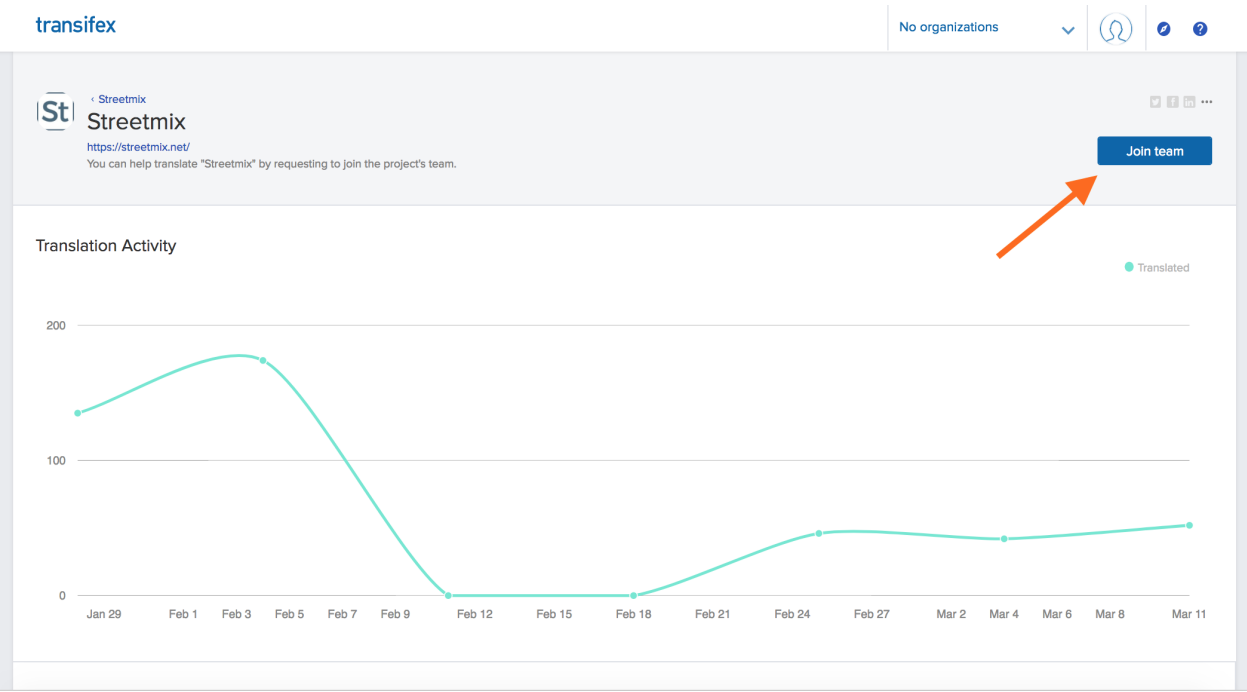
Start here: https://www.transifex.com/streetmix/streetmix/

If you are not logged in or don't have an account yet, you will need to click the "Help Translate Streetmix" button and you will be prompted to log in or create an account.

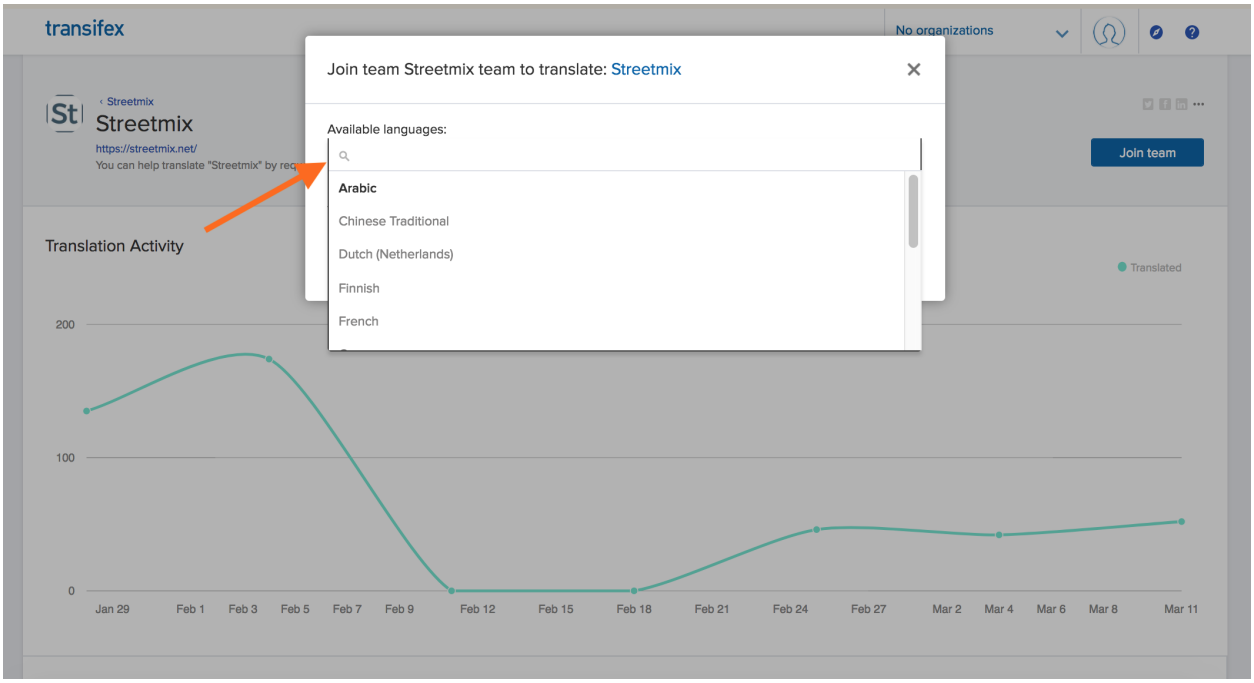


Caution: The Transifex interface can take some getting used to. If something doesn't match a screenshot, let us know. It's most likely that there's some permissions problems, like we did not give you the correct editing privileges for your account. If it's because something in their interface actually changed, we'll update this accordingly.

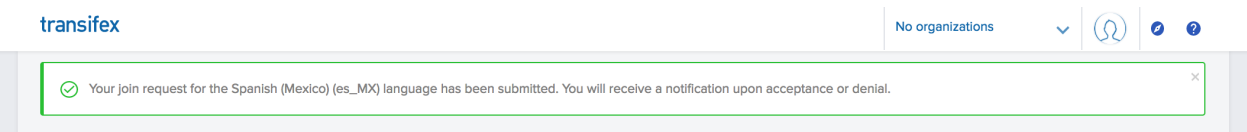
Once you create your account, you may have to confirm your email address. Then, navigate back to https://www.transifex.com/streetmix/streetmix/ and you should see a blue button that says *Join Team*.



Click that button and then select which language you would like to be added as a translator for. If you would like to translate multiple languages, you can join another language later on by going to the *Languages* page, selecting a language, and then clicking *Join Team*.

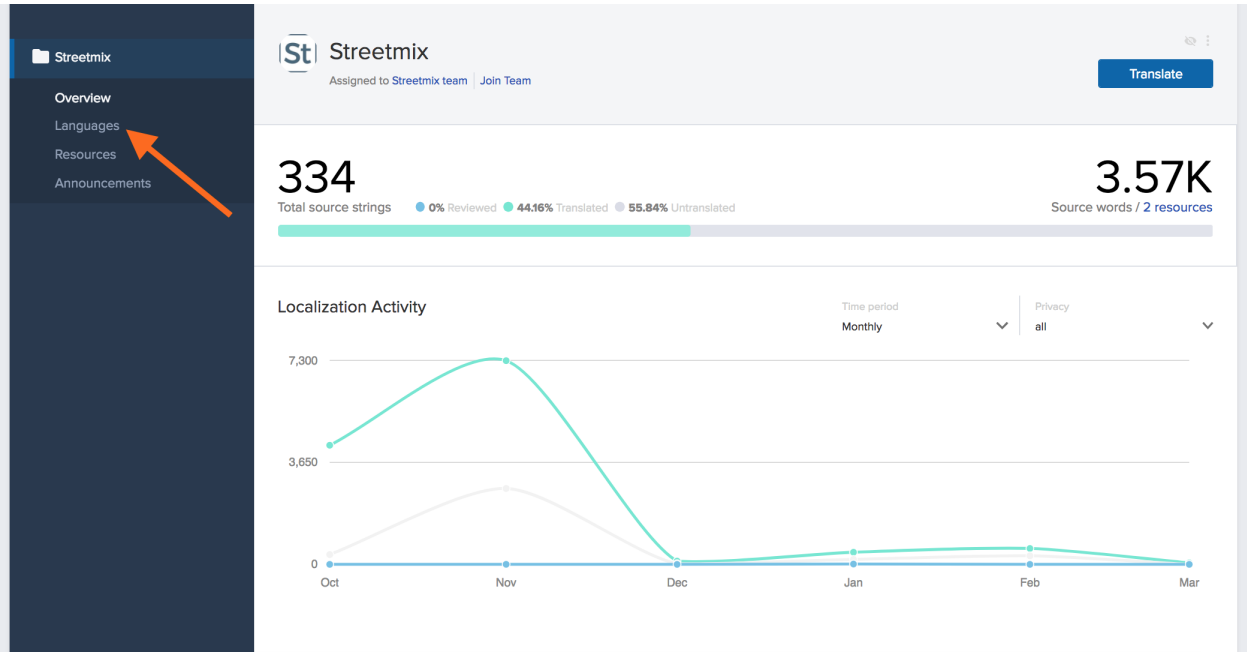


After requesting to join a language, you should receive a confirmation message. Hang tight and a Streetmix project administrator will approve your request.

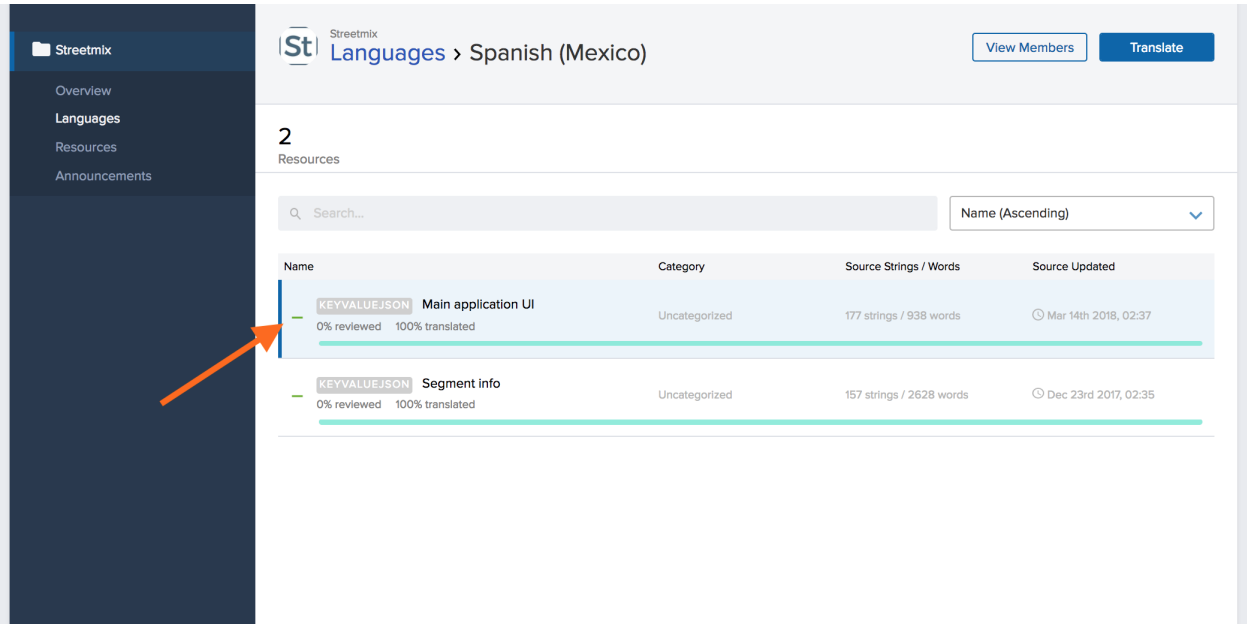


Getting started

Once your request has been approved, you should see the following when you go back to Streetmix's Transifex homepage. From here, select *Languages*.



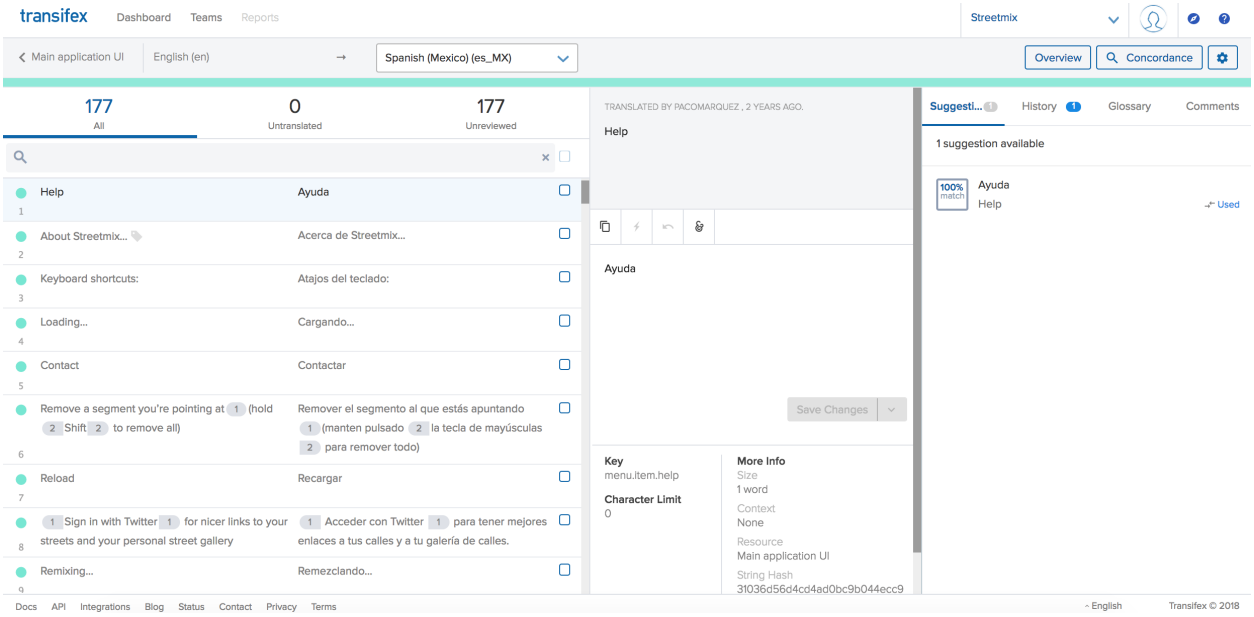
The next screen will show different "Resources". *Resources* are different categories of text that Streetmix uses. Right now there are two. "Main application" is anything that Streetmix uses in the UI, which is almost all the text. "Segment info" is specifically text that is used for each segment (like car lanes or sidewalks). Main application text is the highest priority text to translate, followed by names of segments in the segment info text. Descriptions in segment info is more complex, and might not be up to date or as relevant for other countries and languages right now, so that is the lowest priority.



If you click on a resource, such as "Main application," you will see a popup with more information. From here, click

the large *Translate* button on the right.

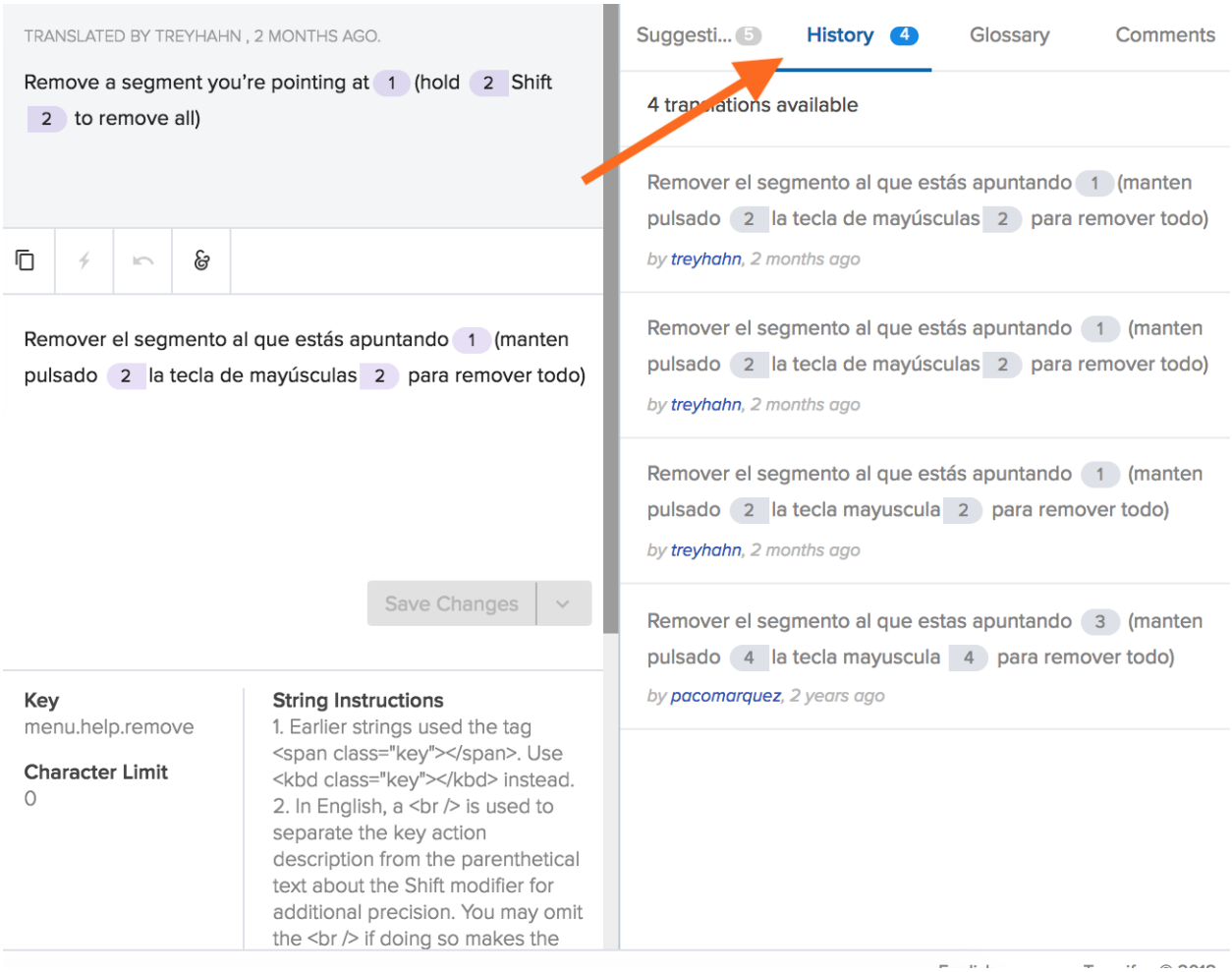
The next screen is the main translation UI where you will be spending most of your time.



On the left you will see a list of all the *strings* (phrases/sections of text to be translated) in English followed by its translation into your language, if already there. On the right is where you will make the translation. When you are looking for a specific string, filter them by using the search bar.

You will also see a series of tabs (*Suggestions, History, etc.*) that can help you create a translation depending on the context. The first is *suggestions* — here Transifex will display similar strings to the one you are translating. This can help you see language that other translators have used for similar strings. However, do not copy and paste this as it is probably for a different string.

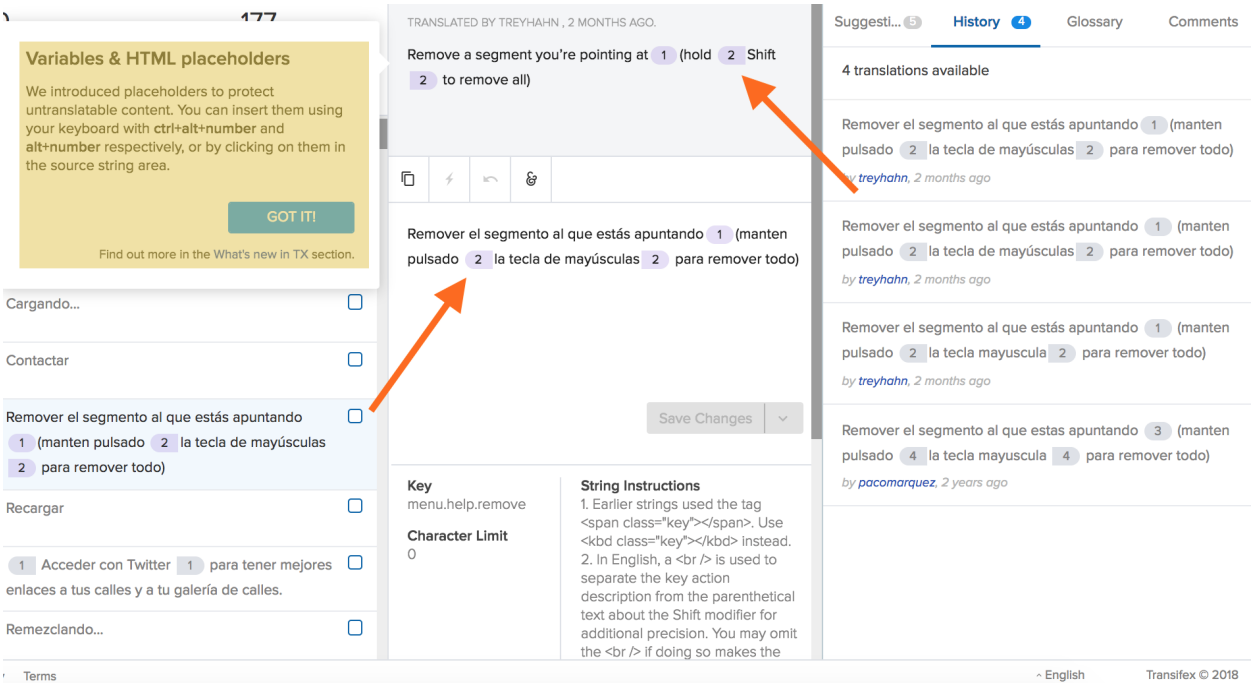
Next, there is the *history* tab. This lets you see prior translations for the string. It is really helpful when an English string is updated, as it lets you retrieve previous translations, and then tweak the string from there.



The last two tabs are *glossary* and *comments*. We are not utilizing the glossary yet, but comments are where you will tell the team anything that you think is important to note about the string. It is also where you can *report issues* with the string (in some cases there may be an error in the English string, so please tell us!).

Placeholders

Some of our strings have placeholders. These are what we use to represent elements in a string that do not need to be translated. This can be a link to a webpage, HTML markup, or text being inserted from another part of the app. Whatever the case, just know that you do not need to translate these. All you have to do is copy them over from the English version of the string. This is a simple process: the placeholders will show up as purple or orange buttons, and you just need to click on it in the English version to copy it over.



Plurals

There is a special way to deal with plural strings in Transifex. They can be inputted in Transifex's UI as shown in the animation below. Some languages have more plural forms than others, and Transifex takes this into account based on a Unicode standard.

Translation string changes

Sometimes there will be updates to the English strings. Transifex will detect those and require a new translation to be submitted. However, have no fear! There is a handy dandy "History" feature that lets you retrieve your previous translations of the string, and then tweak the string from there.

Need more help?

Refer to the Transifex documentation if you need more help. A good place to start is the Transifex web editor tutorial.

FAQ & Discord chat

As questions come in, we will add them to the *Frequently Asked Questions page*.

If your question isn't answered there, come ask the Streetmix community on Discord! Please ask translation-related questions in the #translations channel.

Reporting issues

If you ever see an error on Transifex, please mark it. (Instructions are here.)

If there is a difficult or confusing string, please add a comment on Transifex. You can also put any notes you think are important for the string there. This is the same place where you would mark something as an issue (just click "Add as issue" instead of "Add".

For issues that are larger than one specific string, please bring this up to the Streetmix community. There are two ways you can do this:

1. Tell us on the Streetmix Discord in the *#translations* channel. You can join our Discord server here: https://strt.mx/discord.
2. Create an issue on GitHub (learn more). Our issues are located here: https://github.com/streetmix/streetmix/issues/new

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.5.2 Technical guide

Where translations live

The locale files are located in the `assets/locales` folder, and each separated into sub-directories named by their locale code. These files are generated by an automated script. **Do not manually edit locale files!**

Adding or modifying translatable strings

There is one exception to the above rule, and that's the files in the `assets/locales/en` folder. The English (US) translations are the source language for all translatable strings in Streetmix. This file can be manually modified as necessary to add, edit, or remote translatable strings from the application. Changes here will be automatically mirrored in Transifex within one day.

When new strings are added, Transifex will make them available to be translated in all other languages.

When strings are modified, Transifex will reset all languages to be the blank string, but will remember the history of previous strings.

When strings are deleted, Transifex deletes that string from all locales, *including its history.* Deleting strings are extremely destructive and cannot be recovered. Therefore, be very careful when deleting strings. The danger of losing history usually means that even when a feature has been deprecated or removed, the translation strings will not be removed, out of an abundance of caution.

Previewing translations

Translators can preview translations from Transifex in instances of Streetmix where the *TRANSIFEX_API_TOKEN* environment variable is defined. When present, the strings for each language code are retrieved directly via the Transifex API. In this way, a translator can try different version of a translation in real time to see how it appears in Streetmix.

If you are an in instance where you do not want live translations from Transifex, unset the *TRANSIFEX_API_TOKEN* environment variable.

Updating locale files

Periodically, we export a snapshot of the translated strings from Transifex and store them in our git repository. These locale files are used in production (and offline) deployments, which requires a more stable source of translation.

You can update locale files by running this on the command line:

```
$ node ./bin/download_translations.js
```

Note: A valid `TRANSIFEX_API_TOKEN` must be set in *environment variables* in order to retrieve translations. This environment variable can be obtained from the Transifex platform and it must correspond to a user who is a member of the Streetmix team.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.5.3 Frequently asked questions

Note: This page is in progress and will be updated as questions come in.

For general language guidance, please refer to our *content styleguide*.

Q: Should we use formal or informal language in the translations?

A: Use a tone that is friendly and open. This generally tilts toward informal, but as the translator, you can decide what would work best in your language. Please be consistent across strings and add comments to explain your choices when necessary.

Q: When do I capitalize strings?

A: Please capitalize only the first letter of a phrase/sentence unless it's a proper noun. See *here* for more detail.

Q: When do I add a period at the end of a string?

A: The translated string's final punctuation mark (or lack thereof) should match the source English entry in Transifex. If your language has different rules, please bring it up in our Discord server so we can make a note of this in Transifex.

Q: In my language, action buttons can use either the infinitive form or the conjugated command form of a verb (e.g. "Guardar imagen" o "Guarda imagen" in Spanish). Which form should I use?

A: Please choose the form that you think communicates a friendly tone and then be consistent with your choice. If you disagree with prior decisions, please *report an issue* on the string.

Q: What if I find a link to an English website in a string?

A: Please add "(en)" in your translation so the user knows it's in English. If you have an equivalent link in your language, please create an issue so we can get it added in. Some links are handled through code — don't worry about those.

Q: What are these "strings" you keep mentioning?

A: Strings are chunks of text in the user interface that we need to translate. For definitions of other jargon you may encounter while translating, see the *glossary*.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.6 User interface

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.6.1 Colors

The palette includes the primary brand colors as well as two complimentary tones in a range of tints and shades for use on the website, packaging, and additional brand materials. When using with text, all color pairings should have adequate contrast in accordance with WCAG 2.0 guidelines on contrast ratio.

Palette

Emerald

700
R 12
G 62
B 22

#093f14

600
R 35
G 113
B 55

#1f7234

500
R 35
G 144
B 80

#1a914e

400
R 95

G 192
B 127

#5bc17d

300
R 149
G 218
B 170

#93dba9

200
R 201
G 236
B 210

#c8edd1

100
R 231
G 247
B 234

#e7f7ea

Turquoise

700
R 23
G 68
B 70

#154447

600
R 51
G 108
B 113

#306c72

500
R 86
G 162
B 174

#52a2af

400
R 139
G 193
B 205

#89c1ce

300
R 194
G 235
B 246

#c1ebf7

200
R 221
G 243
B 249

#dcf3f9

100
R 235
G 249
B 252

#ebf9fc

Copper

700
R 85
G 34
B 19

#562211

600
R 136
G 69
B 44

#894528

500
R 180
G 109
B 76

#b56d48

400
R 214
G 141
B 104

#d88d64

300
R 247
G 184
B 147

#f8b890

200
R 248
G 213
B 193

#f9d5c0

100
R 249
G 238

B 232

#f9eee8

Midnight

700
R 24
G 38
B 72

#182549

600
R 54
G 68
B 117

#354377

500
R 84
G 101
B 143

#536491

400
R 124
G 140
B 172

#7b8bad

300
R 166
G 178
B 200

#a6b2c9

200
R 212
G 219
B 231

#d4dbe8

100
R 230
G 235
B 244

#e6ebf4

Other resources

- Building Your Color Palette [Refactoring UI]

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

1.6.2 Icons

General user interface icons

Most user interface icons are stock icons from Font Awesome. We do not have a license to Font Awesome, so we stay within their "free" icon selection.

Icons generally should not be used on their own. They should have a text label for readability. Icons with much less chance of being misinterpreted have more leeway to be used without a text label.

Streetmix segment icons

We've created custom icons to select between different variants of street segments (for instance, inbound or outbound lanes). The source files for these icons are in SVG and are located in `/assets/images/icons`.

Working with icons

These are our guidelines for icons:

- SVGs are made individually. Do not use layers or symbols to store multiple icons within the same SVG file.
- Use a viewbox of 48x48.
- Do not use strokes, only fills.
- Outline (or expand) all text. Do not use font rendering.

- Use black as fill color unless the icon should default to another color. If an icon might exist in Streetmix as different colored versions (see the asphalt icon for example) we can use the CSS `fill` property to re-use a single icon with different colors.
- Clip paths break in Internet Explorer. This is a known bug with no apparent fix in sight. To work around this, crop and subtract paths instead of clipping them.

Build pipeline

All the current SVGs are exported from Affinity Designer. They have a very easy to use exporter that doesn't insert a lot of the extra meta-gunk that Adobe Illustrator does. This file is located at `/assets/images/icons/icons.afdesign`.

If you don't have Affinity Designer or would rather use Adobe Illustrator or another vector file editor, the actual software used to create or export the SVGs are no longer as important, since we also minify these icons during Streetmix application's start-up process. Icons are compiled into a single, compressed `icons.svg` file. Streetmix does not use the raw, source SVG files.

Here are some tips when exporting from Affinity Designer (which can also be relevant to your application of choice):

- Use the *SVG (for web)* preset, which flattens transforms for even cleaner output
- Use the *Whole document* area (turning off other layers), which preserves the square viewbox

Other resources

- SVG 'symbol' a Good Choice for Icons [CSS Tricks]

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

CHAPTER 2

Technical manual

Warning: This page is a work in progress.

This section describes the architecture of Streetmix.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.1 Installing Streetmix

These instructions are for first-time setup on a "local" or "development" environment.

Contents

2.1.1 On Mac OS X

Prerequisites

You may already have some of these prerequisites installed. Skip or update the packages that already exist.

1. Install XCode Developer Tools.
2. Optionally, install the Homebrew package manager. This makes installing other software packages easier, but you can use any other package installation method you wish. In this example and in the following steps, we will use Homebrew on the command line to set up your development environment.

   ```
   $ /usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/insta
   ```

3. Install Node.js. In production, Streetmix uses the latest "Active LTS" release. Locally, you should be able to use any version of Node.js in the "Current" or "Active" state. While you can download installers and binaries from the website, Homebrew makes it easier to keep Node.js up to date in the future. Let's install Node.js with Homebrew:

   ```
   $ brew install nodejs
   ```

4. Install PostgreSQL. You can download MacOSX packages here or use the Postgres app, but the easiest method would be to use Homebrew, again:

   ```
   $ brew install postgres
   ```

5. Install PostGIS. You can download MacOSX packages here but just like above the easiest method would be to use Homebrew:

   ```
   $ brew install postgis
   ```

Install and run Streetmix

See *Instructions for all systems*, below.

2.1.2 On Windows

These instructions below will assume that the user has basic familiarity with Git, GitHub, and the Git Bash or Windows Powershell command line interface, and has administrative permissions to install software on the machine.

> **Warning:** Streetmix was not developed on a Windows platform, and testing is limited. Although our users have successfully stood up Streetmix on Windows machines in the past, these instructions may be out of date.

Prerequisites

You may already have some of these prerequisites installed. Skip or update the packages that already exist.

1. Install a modern browser. We recommend Firefox or Chrome. Internet Explorer is not supported. (See *Does Streetmix support Internet Explorer?*).
2. Install Git. You may want to consider following Microsoft's guide to Windows Subsystem for Linux, which helps us provide similar command line instructions to Mac OSX and Linux environments.
3. Install Node.js. The site should detect your Windows system and provide you with the correct install executable, but you may download a specific package at http://nodejs.org/download/ (e.g. Windows 64-bit installer). In production, Streetmix uses the latest "Active LTS" release. Locally, you should be able to use any version of Node.js in the "Current" or "Active" state.
4. Install PostgreSQL. You can download Windows packages here.

5. Install PostGIS. You can download Windows packages here.

Install and run Streetmix

See *Instructions for all systems*, below.

2.1.3 On Linux

The Linux ecosystem can vary greatly depending on distribution (or "distro") and developer preferences, which makes it challenging to maintain accurate and up-to-date installation instructions that will be perfect for every instance. Consequently, our instructions must assume that the user has basic familiarity with their own system, that common developer tools such as `git` are already installed or that the user knows how to obtain them on their own, and that the user has the necessary permissions to install software on the machine.

Here, our goal is to provide quick-start instructions that should work in most cases, with minimal configuration. However, experienced developers should feel free to modify any of the instructions as necessary according to their own preferences (e.g. database usernames, etc.).

Prerequisites

The primary requirements for this project are Node.js, PostgreSQL and PostGIS. You will need those installed if you do not have them already.

1. Install Node.js. There are different methods for installing Node.js, and you will need to choose a method depending on your own preference. In production, Streetmix uses the latest "Active LTS" release. Locally, you should be able to use any version of Node.js in the "Current" or "Active" state.

   - **Method 1: install with nvm (recommended).** `nvm` is a command-line tool to manage multiple Node.js versions on a machine. This is effective for development and troubleshooting with multiple Node.js versions, and also makes it easy to install and upgrade Node.js versions over time. Learn how to install and use nvm from the repository: https://github.com/nvm-sh/nvm.
   - **Method 2: package manager installation.** Many Linux distributions have package managers from where you can install Node.js. The Node.js documentation provides a list of package managers and installation instructions, but please note the disclaimer that package manager versions are maintained by the Linux community, not by the Node.js team. As a result, not all package managers have the latest Node.js versions at all times.
   - **Method 3: source download.** Download the latest source tarball from the Node.js homepage or choose a specific package from the Node.js download page. Instructions for installing from source are sparse and is only recommended for expert Node.js users.

2. Install PostgreSQL. From the PostgreSQL download page select Linux, then your Linux distribution, for installation instructions.
3. Install PostGIS. Linux instructions (per distribution) are available here.
4. Configure PostgreSQL. If this is your first time installing PostgreSQL, you may need to set up some initial configuration, in the next section.

PostgreSQL configuration

ArchLinux

Here is additional setup instructions for ArchLinux.

Debian/Ubuntu

In our experience, the Debian or Ubuntu package restricts the authentication methods such that one must set up a username with a password in order for Streetmix to be able to access the database. If you experience login problems, check the pg_hba.conf file (see documentation) to see if the `trust` authentication method isn't present for the user. You can either modify that configuration file, or follow these basic instructions for setting up a new user.

```
$ # Switch to the PostgreSQL administrator user
$ sudo -iu postgres
$
$ # Create a new username
$ # Tip: if you create a user with the same name as your Linux
$ # username, you won't need to set the username in Streetmix
$ createuser streetmix_user
$
$ # Enter the PostgreSQL console
$ psql
$
$ # Give your user permission to create and migrate the database
$ ALTER USER streetmix_user WITH CREATEDB SUPERUSER;
$
$ # Set a user password
$ ALTER USER streetmix_user WITH PASSWORD 'password';
$
$ # Leave the database
$ # Note: if prompted, type
$ exit
$
$ # Switch back to original user
$ exit
```

The user created here only needs the `SUPERUSER` role during migration. After a successful initial migration, you may remove the `SUPERUSER` role.

Other

You may need to look for instructions more specific to your distro for setting up PostgreSQL.

We also welcome contributions to our documentation, so if you get Streetmix up and running on a different distro and would like to share how, please feel free!

Install and run Streetmix

See *Instructions for all systems*, below.

2.1.4 Instructions for all systems

After installing all prerequisites (depending on your platform, see above), you can now install and run Streetmix. Assuming all the prerequisites are installed correctly, the instructions below should work on most platforms in the command line terminal. (Note: for Windows platforms, we recommend using Git Bash or Windows Powershell to use these commands.)

Clone and install Streetmix

1. In the command line terminal, clone the Streetmix repository to a folder on your computer.

   ```
   $ git clone https://github.com/streetmix/streetmix.git
   ```

2. Change the directory to Streetmix's root directory, and install project dependencies.

   ```
   $ cd streetmix
   $ npm install
   ```

   Caution: We are not using the **Yarn** package manager. Installing with Yarn may cause unpredictable errors.

3. (Optional) If necessary, create a `.env` file and set PosgreSQL credentials. **Note: By default, most environments will not require PostgreSQL credentials.** For more information, including instructions for setting third-party tokens to run services like authentication or localization, see *Setting environment variables*.

   ```
   $ # Make a copy of the environment variables file
   $ cp .env.example .env
   $
   $ # Using vim, but replace with your editor
   $ vim .env
   ```

   You may use any editor you wish in place of vim. Set your username and password, as well as other database credentials, as necessary, by uncommenting the appropriate lines and setting the environment variables. The following is an example:

   ```
   PGUSER=streetmix_user
   PGPASSWORD=streetmix
   ```

   Tip: If the PostgreSQL username is the same as your operating system's current username, `PGUSER` is assumed to be that username by default, and you won't need to set it explicitly.

4. Initialize the PostgreSQL database.

   ```
   $ npx sequelize db:create
   $ npx sequelize db:migrate
   $ NODE_ENV=test npx sequelize db:create
   $ NODE_ENV=test npx sequelize db:migrate
   ```

   Tip: If you run into issues creating or migrating the database, you can access Sequelize's "verbose" debug output with the command `DEBUG=sequelize* npx sequelize db:migrate`. (This feature is not well-documented by Sequelize.)

   Debug a migration on a Heroku application instance like so: `heroku run 'DEBUG=sequelize* npx sequelize db:migrate' --app <heroku app id>` (Note the quotation marks surrounding the command.)

   In general, Sequelize will print a confirmation or an error after completing each command. If creating the database is successful, you should be able to see the database using psql, PgAdmin, or other tools. A modern, open source, and cross-platform database GUI tool is Beekeeper Studio. The database needs to successfully exist before migrations can occur.

   Notice that the database create and migrate commands are run a second time prepended with `NODE_ENV=test`. This is because your local environment and a test environment use different database instances, and both of them

need to be set up.

You cannot run Streetmix without successfully creating a database, so this is an important step!

Setting environment variables

Environment variables store secret values (like authentication keys and passwords) used to connect to third-party services. Just like regular passwords, secrets should never be revealed to the public, so we store them in a `.env` file that isn't committed to the repository.

You can create a `.env` by copying the starter `.env.example` in the Streetmix root directory.

To obtain keys for local development, you should be able to create your own free-tier accounts at each service and refer to their documentation for more information. To obtain keys to production resources, you will need to ask the project maintainers.

Required environment variables

The only required environment variables are the keys used for the Auth0 authentication service. Streetmix will run without this, but a lot of functionality is only available to signed-in users, and you will need these keys to sign in.

Variable name	Description	Required
`AUTH0_DOMAIN`	Authentication service (Auth0) domain	Yes
`AUTH0_CLIENT_ID`	Authentication service (Auth0) client ID	Yes
`AUTH0_CLIENT_SECRET`	Authentication service (Auth0) client secret	Yes

Optional environment variables

Streetmix will run without these keys. Some functionality will be limited, but they are not critical.

Variable name	Description	Required
`PELIAS_API_KEY`	Geocoding (Pelias) API key	No
`TRANSIFEX_API_TOKEN`	Translations (Transifex) API token	No
`CLOUDINARY_API_KEY`	Image cloud storage (Cloudinary) key	No
`CLOUDINARY_API_SECRET`	Image cloud storage (Cloudinary) secret	No

Optional database configuration (PostgreSQL)

Environment variables are the preferred way for PostgreSQL to access the database. If you have a local database that are not using default values, you can set these here as well. Usually, you won't need to specify these at all.

Variable name	Description	Default value
`PGUSER`	PostgreSQL username	(your system username)
`PGPASSWORD`	PostgreSQL password	(none)
`PGDATABASE`	PostgreSQL database name	`streetmix_dev`
`PGHOST`	PostgreSQL server host IP	`127.0.0.1`
`PGPORT`	PostgreSQL server post	`5432`

Sample .env

A sample `.env` file looks like this:

```
AUTH0_CLIENT_ID=1234567890
AUTH0_CLIENT_SECRET=abcdefghij
PELIAS_API_KEY=a2c4e6g8i
```

Starting the application

1. Start the PostgreSQL service. (Note: the method for doing this may differ depending on your operating system and how you installed PostgreSQL.)
2. Start the web server. In the Streetmix project directory, run:

   ```
   $ npm start
   ```

3. Load the application in your web browser by navigating to `http://localhost:8000` or by running in your terminal:

   ```
   $ open http://localhost:8000
   ```

Stopping the application

To stop running Streetmix, press `Ctrl-C`.

Updating the application

Every so often, you will need to update the project.

1. Pull the latest code from the repository.

   ```
   $ git pull
   ```

2. Install the latest version of all dependencies.

   ```
   $ npm install
   ```

3. Update the database schema.

   ```
   $ npx sequelize db:migrate
   ```

2.1.5 Setup in an offline environment

This is for a special case where you may need to deploy Streetmix onto machines that are going to be running in an environment without Internet access, such as a public space without Wi-Fi, or a conference center with very limited Wi-Fi. To put Streetmix into "offline mode", set your `NODE_ENV` environment variable to `demo`.

You may do this by editing the `.env` file (see *Setting environment variables* for more information about this file).

You can also do it one time by starting the server like this:

```
$ NODE_ENV=demo npm start
```

Caution: "Offline mode" is not a well-supported feature of Streetmix. Use it with care.

Tip: When you are running Streetmix on a device without Internet access, you do not need to provide environment variables for to authenticate third-party services such as Auth0.

2.1.6 Troubleshooting

If you run into problems, please see the *Development issues* section.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.2 Project organization

Warning: This page is a work in progress.

Streetmix is a Node.js and JavaScript project. We use the following frameworks:

- Express, a web application framework. This is basically the server side of Streetmix. It handles HTTP requests and serves files and data requested by the front end.
- PostgreSQL, a relational database. All the data on the server is stored in PostgreSQL.
- PostGIS, PostGIS is a spatial database extender for PostgreSQL, adding support for geographic objects.
- Parcel, a web application bundler. When Streetmix starts, it uses Parcel to bundle all the front end JavaScript and CSS.
- Babel, a compiler which allows us to use modern JavaScript in browsers that do not yet support it.
- React, a front-end user interface framework. Most UI is rendered with React.
- Redux (with Redux Toolkit), a state management framework that usually works alongside React. We maintain most application state in Redux, using Redux Toolkit to help make it easier to write code for Redux.
- SCSS, an extension of CSS that allows us to use variables and calculate values.
- PostCSS, a CSS processor. We primarily use it to autoprefix certain CSS properties.

2.2.1 Dependency pinning

We **pin** our dependencies, which means that we specify exact dependency versions, not version ranges, in `package.json`. This allows Streetmix to run with the greatest reliability and consistency across different computing environments.

Because Streetmix is an application, and it's not intended to be imported by other applications, we don't need the flexibility that comes from using version ranges. As a result, all developers, and any deployment environments, are running the same code for any given commit. This consistency makes obscure bugs easier to track down and resolve.

The tradeoff is that this introduces "upgrade noise". We are currently using Greenkeeper, a third-party automated service to create pull requests whenever a dependency has updated. Because we have pinned dependencies, Greenkeeper creates a new branch and opens a new pull request for *every* depedency update, no matter how minor. We're willing to accept this tradeoff for the time being, although we're open to considering any strategy to limit the noise.

References

- Should you Pin your Javascript Dependencies? *[Renovate Docs]*

2.2.2 Browser support

Because of limited resources, browser support is any of the evergreen desktop browsers (e.g. Firefox, Chrome, Safari, and Edge). We do not support Internet Explorer. (See *Does Streetmix support Internet Explorer?*)

Mobile support is also limited, because the application was not initially designed for mobile. However, we should be supporting tablet devices such as iPads and laptops with touchscreens.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.3 Content Security Policy

Streetmix adopts a Content Security Policy (CSP), which permits only approved content to appear or run on the application. The intent is to mitigate certain types of malicious attacks, like cross-site scripting (XSS) and data injection attacks. As we become a bigger platform, with user accounts and access to user data, it's important we adopt good security practices, and CSP is a web standard that is within our reach.

However, CSP can also be quite limiting. Third-party service integrations, cloud-hosted assets like fonts and images, and browser plugins can break, if they're not explicitly whitelisted. Currently, our CSP directives are very strict.

This section details more information about the way we implement CSP.

- **CSP directives are sent in HTTP headers.** This is the most secure way to set a CSP directive. We use helmet to help write the directive string. This is set up when the Express server is starting.
- **We write the most restrictive directive we can.** Currently we avoid allowing "any" content of any type, if possible.
- **Inline scripts require a nonce value.** Since arbitrary inline scripts are not allowed, we "approve" inline scripts by giving them a unique ID. A nonce value can be anything, but our recommendation is to generate a `uuid`, and make it available to both the CSP definition, and the HTML templates which can then inject the nonce as a variable.
  - Note: we avoid using SHA hashes, which is an alternative way to allow inline scripts. We don't have a way to automatically generate them, so if they're manually generated and added to a CSP header, any change to the inline script, including whitespace formatting, can break the allow directive. They're brittle, so we don't use them.
- **CSP violations are blocked in production, but allowed in development.** While blocking unknown scripts and assets help secure the production deployment, this can be annoying in development environments, where new scripts and assets may be implemented or experimented with. So resources that trigger a CSP violation is allowed when the `NODE_ENV` environment variable is set to `development`. CSP violations are logged to console, so please keep an eye on the console output, which can tell you what directives you will need to update to make new or updated code work in the production environment.
  - Note: Another reason why CSP is relaxed in development (violations are allowed, but reported) is because developer extensions, such as the React or Redux inspectors, are disabled in Firefox if CSP is too strict.
- **CSP violations are reported to the /services/csp-report endpoint.** All reports are logged, in both production and development environments. Some CSP violations are expected, and can be safely ignored. (We don't list the expected CSP violations here because this documentation inevitbably lags behind changes in the codebase.

You will begin to recognize expected violations as you become familiar with local development.) Remember, in development mode, resources that are allowed can still trigger a CSP violation report. Please remember to update the CSP directive if they are intended for production.

  - Note: we do not have automated testing for CSP violations. In other words, continuous integration cannot catch if new or changed code will trigger a violation or issue a report. Do not depend on automated testing to catch CSP violations for you.
  - We log all reports in production so that we can see if users are doing anything that we should be adding to the CSP directive. (For instance: user profile images are hosted on a wide variety of cloud-based cache servers, and we still need to decide whether to allow each of these servers manually, or allow all image resources more liberally.)

- **Avoid writing separate CSP headers for development versus production environments.** We have tried this in the past, and have seen instances where something that "just worked" in development mysteriously stopped working in production, because the CSP headers were different. This "ease of development" strategy became a footgun that made CSP feel worse. This is why we use the strategy of sending CSP headers even in development, and reporting all violations, so that we can catch problems earlier.
  - The one exception we make for development is to allow websockets for Parcel's hot reloader. This is considered acceptable because the hot reloader will never be present in production.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.4 Tests

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.4.1 Frontend Tests

The frontend tests are in the `__tests__` folder of each directory. The naming convention is `filename.test.js`.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

Component Tests

When it comes to testing React components, we need a way to test components in isolation without needing to mount the entire application. We use React Testing Library (more information) to help us write tests.

Tip: Many of our React components use Redux and react-intl, which are required in the component's context to render properly. If a component has either (or both) in its context, use the helper functions in `./test/helpers/` which wrap React Testing Library's *render()* with mock `<Provider />` and `<IntlProvider />` components.

How to test components

React Testing Library works directly with the rendered DOM. It provides utilities for querying the DOM in the same way the user would. Finding for elements by their label text (just like a user would), finding links and buttons from their text (like a user would).

Test the interaction rather than the way it looks. For example use `fireAction` from React Testing Library to simulate Mouse and Keyboard events.

Use the render method, be aware that you need to `wait` for changes when the store is used. Due to the asynchronous nature of the store. The helper method `render` provides you with a wrapper around `Provider` and you can provide `initialState`.

Snapshot testing

Snapshots should be used with caution. They tend to break, and developers tend to update them without examining why a snapshot might have changed unexpectedly. Please be careful when snapshots fail and when to add new ones.

Snapshots are good when you test different results, like error messages. Take a look at `./assets/scripts/app/__tests__/StreetEditable.test.js` for an example how snapshots can be used with error messages.

Mocks

Be aware of mocks. A few files and functions are mocked globally. For example `load_resources` is mocked globally and if you need to use that file in your tests/components be sure to check the mock. Otherwise use `jest.mock` to mock modules, classes, etc.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

API testing

Currently MongoDB needs to be running for front end tests. To remove this barrier and potential bottleneck we're moving away from using `fetch`, to use *Axios <https://github.com/axios/axios>*. It provides us with convenient methods and a single place for API call. Axios Mock Adapter offers the possibility to mock all API requests and eventually remove mongodb as a dependency for front end related tests.

If you work on API-related features, consider refactoring the `fetch` calls to use the API in `.assets/scripts/util/api.js`

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

Integration testing

Integration tests help to test a specifix flow of information.

What integration tests to write?

Whenever you're writing boilerplate tests consider writing an integration test rather than an unit test. Especially Redux-related actions and reducers are a good opportunity to use integration tests. Another good point to introduce an integration test is for a test where you need to mock a lot of modules to test in isolation. This is typicaly a sign of a lot of side effects that should be tested with an integration test.

Redux

Redux actions and reducers are mostly boilerplate code and it is tedious to write unit tests for them. They do not serve a specific purpose since you're testing the framework more than you're testing your code. There are instances where you have more business logic in your actions, typically for asynchronous actions. The best way we found to test those actions is to use a Integration test approach. Instead of testing the action in isolation, we test the end result of the action. Since actions are the way to change the store, we can use that to make assumptions on the end result.

Take a look in `./test/helpers/` where you find a helper for creating the store.

Unit tests

We use the Jest testing framework.

End-to-end testing

We have adopted Cypress.io, a modern framework for end-to-end testing, to make writing and running our integration and end-to-end tests easier.

Tests are extremely important to the health and stability of Streetmix. We have established some systems and processes to help ensure the ongoing reliability of our platform.

We do not have a strict test-driven development (TDD) methodology, although individual engineers may use this approach if that's the development pattern they are most comfortable with. Also, while we do measure code coverage, our goal is not necessarily to reach 100%. We're looking for "enough" coverage to have confidence that new features or refactoring will not create new bugs, which can be more of a subjective approach. As Guillermo Rauch says, "Write tests. Not too many. Mostly integration."

For context...

We did not have *any* test infrastructure in the early phases of Streetmix. Tests have been added over time and are constantly improving. This document reflects our current thoughts about *how* we should test, but you'll find lots of moments in the codebase where tests are incomplete or non-existent. We could always use some help with writing tests!

2.4.2 Running tests locally

When testing in a local development environment, only *linting* and *unit tests* are run.

```
$ npm test
```

Full integration tests happen in our continuous integration infrastructure. You're not required to run this locally, but if you'd like, you can do so with this command.

```
$ npm cypress:run
```

2.4.3 Frameworks

Unit and integration tests

> **Warning:** This section is incomplete and should be expanded.

Our primary test framework is the Jest test runner with React Testing Library (RTL). (These do not do the same thing and are not interchangeable; these two systems work closely together to provide a full unit and integration test environment.) A number of resources already exist that fully document why and how we use these, see the resources list just below.

Our goal is to be as close as possible to a community-designed "best practice" in order to simplify our understanding and comprehension of tests; *do not do anything exotic in these tests if you can avoid it.*

Resources

- Introducing the react-testing-library [Kent C. Dodds] - why RTL instead of Enzyme?
- How to use React Testing Library Tutorial [Robin Wieruch] - start here for the basics
- Common mistakes with React Testing Library [Kent C. Dodds]

End-to-end tests

> **Warning:** This section is incomplete and should be expanded.

We use Cypress sparingly. We do eventually want more tests to exist in Cypress, when appropriate, and can replace the unit or integration tests that end-to-end tests can cover.

Cypress only runs in *our automated continuous integration test environment* by default, but can also be run locally:

```
$ npm run cypress:run
```

Linting

We use ESLint and Stylelint to lint JavaScript and CSS, respectively. There is a commit hook that automatically runs the linter on each commit. If the lint fails, you will need to fix your code and try your commit again, or force it to ignore the lint errors. For more on code style, see *Code styleguide*.

In the future…

…we may adopt Prettier (or prettier-standard) to automatically format code. We have not introduced it yet because doing so across the entire codebase would be disruptive to existing work. We currently use Prettier to lint and format JSON. If someone wants to champion adoption of Prettier, please get in touch.

Type safety

JavaScript is notoriously not type safe: you may pass any type of object or JavaScript primitive to any function or method, which may not be able to handle them. Or you may write a function that returns values of different types, and the calling script wasn't expecting that return value. Various attempts to introduce type safety on top of JavaScript have entered the ecosystem, and here's how we use these tools.

PropTypes (React)

PropTypes is a runtime typechecking library used for React development. Because it is a runtime checker, PropTypes will only throw errors in the console when running in the browser or in test suites. (The PropTypes library is not compiled into production code.)

We currently enforce using PropTypes for React components in development. This means that React components must declare all of its props and what types of values that prop should be. The benefit of this approach is that React components self-document what props it accepts. Sometimes, a prop can be overloaded with multiple types, but this is generally discouraged if you can avoid it.

TypeScript

TypeScript is an extension of the JavaScript language that allows types to be checked statically (that is, reason about whether the right types are being passed around, without having to run the code itself). It's been growing steadily in popularity (based the State of JS 2018 report) over the past few years.

We have experimented with TypeScript in auxiliary codebases, but we've not incorporated it into Streetmix itself. Because we already compile code with Babel, adopting TypeScript piecemeal is doable. However, we have not yet run into a situation where the solution is specifically to adopt TypeScript. That being said, if and when a good case can be made for adopting it, we will likely jump on board. If a migration to TypeScript occurs in React components, it will supercede using PropTypes.

Device testing

We do not currently implement device testing, but this is on our to-do list. We have a Browserstack account for this purpose.

2.4.4 Continuous integration (CI)

We use Travis CI to automatically run tests for every commit and pull request to our repository.

Troubleshooting CI

Continuous integration testing is, unfortunately, not deterministic. Because there are various moving parts and third party services involved, CI can sometimes fail, despite the code running perfectly locally (or even in production)! When CI fails, we need to examine why. **Passing CI is almost always required to maintain confidence that a deploy will not break the production site.**

Even after determining that CI is failing not because of a bug or linting problem, here are some common tips for addressing issues with the CI infrastructure.

1. **Try running the build again.** Because CI isn't deterministic, sometimes running it a second time with no changes will cause it to pass. This is commonly the issue when the Selenium smoke test fails.

2. **Check the status of third-party services.** Sometimes, TravisCI itself has issues, so also be sure to check TravisCI status.

Skipping CI

CI can be skipped by appending `[skip ci]` to a commit message.

Automatic deployment

Every commit or merged pull request to the `main` branch that passes CI is automatically deployed to the staging server.

Currently, there is no automatic deployment to the production server. We've noticed that each deploy introduces a small amount of lag while the server software restarts. As a result, we now manually trigger deployments to the production server.

2.4.5 GitHub checks

In addition to continuous integration, we use some third-party services to keep an eye on code quality and test coverage. These services should be considered "code smell" detectors, but treat them with a grain of salt. They are not required to pass before merging pull requests.

CodeClimate

CodeClimate measures **technical debt**, or the long-term maintainability and readability of code. It applies some heuristics to detect and track "code smells," which are opportunities to refactor code or fix potential bugs. A CodeClimate review is triggered automatically on every pull request, but some of the thresholds it uses are quite arbitrary. Here's some of the issues are raised, and how we'd address them, in order of increasing severity (as it applies to Streetmix):

- **Lines of code**. CodeClimate triggers a warning when functions and modules exceed an arbitrary line limit. This means there is a potential opportunity to separate concerns, but we will never enforce this, since we don't want to encourage "code golf" or quick workarounds instead of actually taking the time to separate logic. If something can be refactored into smaller pieces, but can't be prioritized immediately, add a `TODO` comment instead. If something doesn't make sense to shorten, mark the issue as *Wontfix*.
- **Duplicate code.** CodeClimate triggers a warning when it detects code that look the same as other code elsewhere. This can be an opportunity to refactor code, but more often than not, CodeClimate is seeing similar-looking boilerplate code or patterns. In this case, mark the issue as *Invalid*.
- **Cognitive complexity.** CodeClimate triggers a warning when a function contains too many conditional statements, resulting in complex branching or looping code. Not all code can be made simpler, but you may want to consider whether it can be written diffferently. However, use your best judgment here. If you don't agree with CodeClimate's assessment, mark the issue as *Wontfix*.
- **TODOs**. CodeClimate tracks when a `TODO` or a `FIXME` comment is written in the code. Because this is a developer's own judgment call, this takes priority above other issues and should be addressed in the future. Never mark this as *Wontfix* or *Invalid*. If it's no longer valid, instead remove the `TODO` or `FIXME` comment from the code.

Issues that should be addressed in the future, but can't or won't be addressed immediately, should be marked with *Confirmed*.

In spite of CodeClimate's warnings, reviewers may approve its review even if the issues it raises are not addressed right away.

Codecov

Codecov measures code coverage, which is the percentage of code that is covered by at least one test suite. This percentage is a commonly used metric that software projects use to show how complete its test suites are. However, the percentage itself is not necessarily a measurement of test *quality*. As a result, while we strive for higher coverage, 100% is not the goal.

A Codecov review is triggered automatically on every pull request, which allows a reviewer to see at a glance whether a pull request increases or decreases overall code coverage. It fails if a large amount of new code is added without increasing a corresponding amount of test coverage.

Because our test suite coverage is quite low at the moment, it is preferred that all new code and refactored code come with test suite coverage.

2.4.6 Resources

These additional resources from the developer community help guide our approach to testing. This is not an exhaustive list, and we'll keep updating this over time.

- Write tests. Not too many. Mostly integration. [Kent C. Dodds], on Guillermo Rauch's tweet.
- JavaScript & Node.js Testing Best Practices [Yoni Goldberg]

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.5 Segment definitions

Attention: This page is a work in progress. This is a draft specification of Streetmix's forthcoming segment definition schema. For documentation related to the legacy specification, see this file.

The schema and structure described here is subject to change and evolve.

2.5.1 Glossary

segment **Segments** are the building blocks of a street. A street cross section is composed of a sequence of non-overlapping segments. Note that a *segment* in this context is not the same as a *street segment*, which is a length of a street with an arbitrary start and end point (but usually at intersections).

Segments are defined as an assembly of various *segment components*.

segment component Segments are made up of **components**. We will call them *segment components* to differentiate them from the generic concept of a *component*, which might be used in other contexts (for instance, a *React component*).

sprite A **sprite** is the illustration used to represent a *segment component* visually.

2.5.2 Types of segment components

There are four types of segment components. Segment components have **profiles**, based on its **type**, that describe its general characteristics.

1. Lanes

Lanes are ground-level zones of street activity. Common examples of lanes include *vehicle travel lanes*, which are typically allocated to automobiles and bicycles, paved with asphalt, clear of obstructions, and allow for high speeds of travel, and *sidewalks*, which are typically allocated to pedestrian use, paved with concrete or other decorative paving material, and allows for lower speeds of travel.

Lanes have a consistent width, but a variable length, aligned along the street's centerline axis. Lanes do not have to be consistently parallel with the street axis, nor does it need to be continuous along the entire length of a street segment. While lanes are assumed to have a consistent width, it is possible for lanes to deviate along its length.

Lanes are not required to permit vehicular travel. Some may be designated for parked vehicles (*parking lane*), or be clear of vehicles altogether (*sidewalk furnishing zone*, *planting strip*).

Lanes are not required to have the same function at all times. Some lanes change function throughout the day, such as parking lanes that turn into travel lanes during rush hour. Some lanes may allow a mix of functions throughout, such as a parking lane where some spaces are occupied by cafe seating.

Lanes have the following defined characteristics:

- **Surface material**, such as concrete, asphalt, brick, permeous paving, etc. This informs
- **Surface height**, a +/- delta value from "grade" level, which is defined as zero. The "grade" level is the top of the surface used for vehicular travel. A height value of +1 is the height of a standard sidewalk curb lip. Note that this value is _not_ a real-world engineering measurement, and visual rendering of surface height in Streetmix is illustrative.
- **Intent**, which specifies allowed functionality. Example values may include *pedestrian*, *fast vehicular*, *slow vehicular*, *parked vehicular*, *recreational*, *unknown*, or *any*. This may be overridden at the *segment component* level.
- **Minimum desired speed**, for *intents* that allow vehicular travel, set the lowest minimum desired speed. This may be overridden at the *segment component* level.
- **Maximum desired speed**, for *intents* that allow vehicular travel, set the highest maximum desired speed. This may be overridden at the *segment component* level.

Segments must have one lane component.

2. Vehicles

Vehicles are modes of transportation.

- Vehicles. This defines things that can travel in a direction of motion (e.g. pedestrians, bikes, scooters, cars, trucks, magic flying carpet)
- **Label**
- **Weight**
- **Footprint** (square meters)
- **Speed** (maximum)
- **Emissions**
- **Capacity**
- **Level of automation**

3. Objects

Objects, also known as furniture, are stationary items. Examples include *benches*, *parklets*, *trees*, and *signage*.

Objects can be a single-instance item, like signage, which usually occur once in a specific spot. Objects can also repeat along a lane's length, such as street trees, which might be planted every 10 m or so. Note that repeating object spacing are rough guidelines for illustrative purposes and may not reflect actual spacing in reality, because of real-world physical constraints.

One handy rule of thumb for determining if something should be an *object* is to consider whether a single instance of it can be mapped with a point (that is, a latitude/longitude coordinate pair) and still have meaning. Ob

Objects have the following defined characteristics:

- **Label**

4. Markings

Markings a fourth "special" category that defines various ways lanes can be marked. An example of a *marking* are lane striping. Markings are conceptually similar to objects, in that they are stationary, and can possibly repeat along the length of the lane. However, they're different in that markings are really symbols. Their physical expressions are a side effect.

Note: Markings are a transitional category which emerged because it was necessary to find a bridge between the original segment specification and the new one. This may go away in favor of splitting the different types of markings between the *lane* segment component type and a new property, *segment transitions*.

Components definitions

Segment components are defined in https://github.com/streetmix/streetmix/blob/main/assets/scripts/segments/components.json

2.5.3 Segment definitions

Background

Currently every object type (what we call "segments" internally) is defined here: https://github.com/streetmix/streetmix/blob/main/assets/scripts/segments/info.json

Its basic structure is very minimal. For every segment we have some properties attached to it (e.g. its display name, minimum / maximum recommended width, if any, and so on), and we also specify which image sprites represent the segment graphically. Each segment may also have variants, which come with its own properties.

The problem with this format is that each segment hard-codes its own definitions for everything. For example, both drive lanes and parking lanes define what a car is, redundantly. This makes it very hard for segments to represent real-world flexibility. If we wanted all segments capable of supporting a bicycle to have a bicycle option, then each segment needs to duplicate the details of the bicycle.

Finally, there is the problem variants. Each segment variant combines with other varients so the total number of variants are multiplied together. This means each time a new variant is added, the data multiplies exponentially. Most segments have two variants, which is manageable. Once you have three or more, it's unworkable. It's one of the main reasons why we resisted putting in raised bike lanes despite it being one of the most requested segment features over

the last few years. Doing so raised the complexity of our data, and so we need to consider how to implement it in a better way.

Segments are now defined as an assemblage of components, with additional properties. See here: https://github.com/streetmix/streetmix/blob/main/assets/scripts/segments/segment-lookup.json

Additional segment properties

Once a segment has been constructed out of component parts, it can define additional information.

Properties

- **Label**. Name of the segment.

Properties may be editable by the user.

Variants

Variants are toggles that slightly change the appearance or behavior of the segment.

Segment transitions

Adjacent segment transitions define what happens when the edge of one segment abuts the edge of another segment.

Connecting segment transitions define what happens when the end of one segment's *lane* is connected to another segment 's *lane*.

Note that segments cannot overlap in this specification, so if a segment overlap exists, it is redefined to adjacent, and the adjacent transition applies.

> **Attention: (August 2021) Streetmix documentation has now moved!** Please update your bookmarks to https://docs.streetmix.net/.

2.6 Environs definitions

> **Attention:** This page is a work in progress. This is a draft specification of Streetmix's forthcoming environs feature. The schema and structure described here is subject to change.

Environs is a setting that allows uers to change the "environment" that a street is in, which can include elements like time of day, weather, and background. In the UI, we prefer to use the term *environment*. Internally, we use *environs* to prevent confusion with the computing term *developer environment*.

Environs definitions are contained in `environs.json`. You can browse the current environs definitions here.

2.6.1 environs.json schema

The definitions file exports a single JavaScript object, containing a keyed collection of environs definitions. The key is the identifier for the environs, and the value is an object with the following properties:

`name` Required. A string that is the English-locale display name for the environs. *Note: we have not yet determined a method for retrieving translated names in other locales.*

```
"name": "Night"
```

`enabled` Optional. A boolean value, which determines whether the environs is available for the user to choose. If this property is not set, its default value is `true`. If set to `false`, the environs will not be accessible in the UI. However, streets that have already been set to a disabled environs may still render that environs.

```
"enabled": false
```

Note: We use `enabled` as the property name instead of `disabled` because code such as `if (environs.enabled)` is easier to read than `if (!environs.disabled)`, which uses a double negative. It is also easier to use read and use filters in this way.

`iconImage` Optional. A string that refers to a custom icon image to be used when displaying a graphic icon for the environs. The image is retrieved from the application's image cache. If this property is not defined, no custom image will be displayed.

```
"iconImage": "environs--icon-night"
```

`backgroundImage` Optional. A string that refers to an image that is displayed in the background of the street. The image is retrieved from the application's image cache. If this property is not defined, no background image will be displayed. Background images display at 1:1 pixel scale and will be tiled to repeat in all directions.

```
"backgroundImage": "sky--stars"
```

`backgroundGradient` Optional. An array of CSS colors that create a linear gradient corresponding to the CSS gradients specification. Linear gradients specified in this way will always be rendered top-to-bottom.

Each item in the array may be a string that is a CSS color:

```
"backgroundGradient": [
  "#01051600", "#01081777", "#031641", "#061c45"
]
```

. . . or, each item may be an array, where the first item is the CSS color string and the second item is a stop value:

```
"backgroundGradient": [
  ["#01051600", 0],
  ["#01081777", 0.33],
  ["#031641", 0.66],
  ["#061c45", 1]
]
```

Mixing and matching item types is permitted. Array items can optionally omit the second item (the stop value). If a stop value is not provided, the transition between each step will be interpolated by the browser.

```
"backgroundGradient": [
  ["#01051600", 0],
```

(continues on next page)

(continued from previous page)

```
  ["#01081777"],
  "#031641",
  ["#061c45", 1]
]
```

Any valid CSS value is permitted, including RGBA definitions and keywords.

```
"backgroundGradient": [
  ["rgba(255,255,255,0.85)", 0],
  ["rgba(192,192,192,0.5)", 0.85],
  ["transparent", 1]
]
```

If the gradient has any transparency, the `backgroundColor` property, if defined, will be underneath the transparent colors. If `backgroundColor` is not defined, the underlying color will be the page background. We do not recommend relying on the page background as the underlying color.

`backgroundColor` Optional. A string of a CSS color which is a solid background color. We always recommend setting a background color, even if a non-transparent background gradient or a background image is specified, which allows it to be used as a fallback color, just in case.

```
"backgroundColor": "#000928"
```

`backgroundObjects` Optional. An array of objects, defining images to be displayed in the background. Each object has five required properties:

`image` A string, referring to an image in the application image cache.

`width` A number, specifying in CSS pixels the width to display the image.

`height` A number, specifying in CSS pixels the height to display the image.

`top` A number between 0 and 1, specifying the relative vertical position of the image in the background. A value of 0 is at the top of the background, and a value of 1 is at the bottom.

`left` A number between 0 and 1, specifying the relative horizontal position of the image in the background. A value of 0 is at the left side of the background, and a value of 1 is at the right.

Images that cannot be retrieved from the cache will not be displayed.

```
"backgroundObjects": [
  {
    "image": "sky--moon",
    "width": 116,
    "height": 116,
    "top": 0.2,
    "left": 0.8
  }
]
```

`foregroundGradient` Optional. This works exactly the same as `backgroundGradient` (see above), but creates a gradient that overlays street elements, such as buildings and vehicles. We recommend setting `transparent` as the final value so that the overlay fades away smoothly.

```
"foregroundGradient": [
  ["rgba(255,255,255,0.85)", 0],
  ["rgba(192,192,192,0.5)", 0.85],
  ["transparent", 1]
]
```

`cloudOpacity` Optional. A number between 0 and 1, specifying the opacity of the clouds that are normally displayed in the background. The default value is 1. A value of 0 will make the clouds invisible.

```
"cloudOpacity": 0.2
```

`invertUITextColor` Optional. A boolean value, which if `true`, will invert the color of any text that exists on top of the background. The exact implementation of color inversion may differ between UI components. Please set this to true if you have a dark-colored background, and choose the value that creates the best contrast according to WCAG 2.0 guidelines on contrast ratio. Also, please be aware the viewport heights differ between devices, and that backgrounds will be "compressed" to fit the vertical height of the viewport. Backgrounds that change from light to dark (so that contrast ratio varies greatly depending on the viewport size) is not recommended.

```
"invertUITextColor": true
```

All other properties are ignored. However, additional properties may be defined in the future. We recommend using undefined properties as comments and prefixing comment properties with `_comment`, for example:

```
"_commentBackgroundColor": "This is a comment about the background color."
```

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

2.7 Helpers and utilities

Warning: This page is a work in progress.

We use the following utilities to solve common programming problems.

2.7.1 Pseudo-random number generator

If you need a one-time use random number (which will never be stored), there's nothing wrong with JavaScript's built-in `Math.random()`. However, there are times where we need a **seed** for a pseudo-random number generator (PRNG). Seeds generate a random-*looking* sequence of numbers, but they have the benefit of remaining consistent across renders. For example, this is useful on our sidewalk segment, where we want to generate a random set of pedestrians but we want those pedestrians to remain the same between page refreshes or React component updates.

We can generate seeds by calling `generateRandSeed()`, then by feeding those seeds into the `seedrandom` library:

```
import seedrandom from 'seedrandom'
import { generateRandSeed } from './util/random'

function doThingsWithThePRNG () {
  const seed = generateRandSeed()
  const prng1 = seedrandom(seed)

  // Generate pseudo-random numbers. Each time rng() is called, it returns
  // the next pseudo-random number in the sequence.
  console.log(prng1()) // -> first pseudo-random number
```

(continues on next page)

(continued from previous page)

```
  console.log(prng1()) // -> second pseudo-random number
  // .. etc

  // Create another generator with the same seed
  const prng2 = seedrandom(seed)
  console.log(prng2()) // -> same value as the first call to prng1()
}
```

When `seedrandom()` is called, it returns a function that will return a consistent sequence of numbers each time it is called, for the given `seed`. The sequence returned is local to that function.

Note: Although the `seed` can be any type of value, like a string, `generateRandSeed()` returns only integer values so that when we store seeds in a data model we can check for type consistency.

In the above example, we call `seedrandom()` twice, with the same `seed`, so that we can create the same sequence of pseudo-random numbers in two different scenarios. This example is contrived because the pseudo-random number generators are used in the same function, and in reality the `seedrandom()` call may happen in separate parts of the application.

References

- Random Seeds, Coded Hints, and Quintillions, *David Bau, 30 January 2010*
- seedrandom.js *[GitHub]*

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

CHAPTER 3

Support

Warning: This section is a work in progress.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

3.1 Frequently asked questions

3.1.1 Can I use Streetmix for professional presentations or publications?

Yes, absolutely! Use Streetmix however you like, but *please provide attribution*.

3.1.2 What is the end user license?

Streetmix has adopted two licenses: the **software** uses the GNU Affero General Public License v3.0 (AGPL), while the **content** uses a Creative Commons CC-BY-SA license. One or both of these licenses might apply to your situation. These licenses give you some essential rights for using Streetmix without having to explicitly ask us for permission.

The **AGPL license** is an open-source software license. This means you may copy, modify, and reuse the original code base, as long as the copyright remains with Streetmix LLC, and that you make your modified source code open if you host a publicly accessible service.

The **Creative Commons CC-BY-SA license** governs the content in Streetmix, such as text or graphics, as well as any derived graphics or content. You may modify and republish anything you make in Streetmix as long as you *provide attribution to Streetmix*. You must also allow others to reuse or republish your Streetmix-created content.

This is a basic overview of what these licenses mean. For more details, including how these licenses may apply to your specific use case, please refer to the license text or consult with a qualified legal advisor.

Note: Previous versions of the Streetmix codebase was licensed under a **BSD 3-Clause license**. We are not lawyers and cannot offer legal advice, and if you believe your usage of Streetmix is affected by the change in license, please consult with your legal counsel.

3.1.3 How do I provide attribution to Streetmix?

When you reuse or republish Streetmix content, you should mention the following:

- That you created this content with Streetmix
- That the content is licensed under Creative Commons (or the AGPL license, if the reproduction includes code)
- Include a link to Streetmix (https://streetmix.net/)

We are flexible with attribution. You may be in a situation where you're not able to include all or some of those things. That's okay! But please use your best judgment about what form of attribution is viable for you.

3.1.4 Can I copy the Streetmix code or set up my own instance?

You sure can. If you are technically inclined, please refer to the project repository README for installation steps.

If you are looking for someone to set up a whitelabeled instance of Streetmix, we can help! Shoot us an email to hello@streetmix.net and let's chat.

3.1.5 Does Streetmix support Internet Explorer?

Our unique interface means Streetmix is designed with the most recent web technology in mind. We do not support the Internet Explorer browser, and we are deprecating support for Internet Explorer 11 in favor of the newer Microsoft Edge browser.

We also support the common "evergreen" browsers Chrome, Firefox, and Safari.

3.1.6 How do you add setback lines or curb-to-curb widths?

This is not currently possible in Streetmix. We designed Streetmix for a lay audience, who may not be fully aware of all the fixed constraints of street design. As a result, we decided not complicate the user interface with setbacks or curb-to-curb dimensions, focusing instead on space between buildings, which is what people experience every day. We plan to add these features in the future for a professional version of Streetmix.

3.1.7 How do you lock down parts of a street to prevent users from editing them?

We believe that civic engagement is most empowering when users are not artificially constrained from expressing their creativity or their opinions. Instead, we believe users should be educated about constraints and be given the right to accept or reject them. Furthermore, a digital sandbox is the perfect environment to experiment with concepts that might be impossible or unrealistic in real life.

3.1.8 Can you add X feature?

We'll see! Please *contact us*. We can't respond to all requests, but we do read and consider all user suggestions to help prioritize new features in Streetmix.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

3.2 Troubleshooting

3.2.1 Loading or authentication issues

When loading Streetmix, the screen is stuck on the loading spinner indefinitely, or I only see the "We're having trouble loading Streetmix. (Error RM1)" error message.

If Streetmix is stuck on the initial loading spinner screen, try to open a "private mode" window in your browser. Normally, this mode is used to prevent browsers from saving history, and it also means you can start with a "clean slate" when browsing the web. Here are resources for the two most common browsers.

- How to browse in Incognito mode in Chrome
- How to enter Private Browsing in Firefox

If Streetmix works in private mode, this means something in your browser cache has become corrupted. You will want to reset your browser back to a "clean slate."

- How to clear browsing data in Chrome
- How to clear cookies and site data in Firefox

If Streetmix does not work in private mode, or still does not work after clearing your browser's cache, please ensure that you are using the latest version of Chrome or Firefox. If you still need help, please *contact us*.

When following an e-mail sign-in link, I see the "You cancelled the sign-in process" message.

This can happen when an organization-managed e-mail service blocks or tampers with the sign-in link while doing routine security checks. We are currently investigating solutions to this problem but do not yet have a timeline for a fix. The following workarounds can help:

- Sign in with a personal e-mail address (e.g. Gmail) or with a Twitter, Facebook or Google account method.
- Ask your organization's IT administrators to allow sign-in emails from Streetmix.

3.2.2 Error codes

Warning: This section is a work in progress.

These are error codes that may occur while using Streetmix.

Error code	Reason	Suggested fix
9B	**Data error**: The server sent street data that had no data in it.	Load a different street.
RM1	**Authentication error**: The user attempted to load Streetmix with remembered credentials which have become corrupted.	Reset cache and site data for the site.

3.2.3 Development issues

While developing Streetmix, here are solutions to some problems that may arise.

Strategies for resolving most common issues

Check which version of Node.js you are using and make sure it matches the versions listed in `package.json`.

If you're still running into problems, **try this first!**:

- Update your global `node` and `npm` versions to the latest versions.
- Remove the `node_modules` and `.cache` folders, if present, and reinstall (*Installing Streetmix*).
- Ensure that you are installing and running Streetmix with `npm`, not `yarn`.

Specific issues

This section is for troubleshooting specific issues.

- **The server keeps restarting in a loop** with the `EADDRINUSE` error code. We have documented a solution in GitHub issue #983.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

3.3 Report a bug

If you notice any bugs while using Streetmix, let us know by opening a new issue on our GitHub repository. Please check to make sure it hasn't already been reported. If there is already an existing issue, feel free to add a comment or indicate your support with an emoji reaction.

For serious bugs on our production server, you might be able to get a faster response by notifying us on Discord first.

When submitting a bug report, please include the following information:

- What **web browser** you're using (i.e. MS Internet Explorer, Google Chrome, Firefox), what version (i.e. 9.0, 27), and what operating system you're using (i.e. Mac OS X 10.8, Windows 7, Android 4.1).
- If you are running a local copy, include **details about your enviroment** that's necessary to reproduce the bug, such as your node version, npm version, and operating system.
- The **URL** in your browser's location bar.
- **What you were doing** just before the bug occurred. The easier it is for us to reproduce the bug, the better.
- **What you expected to happen.**

- **What actually happened.**
- Any **other information** you think might be at all helpful. If you are technically inclined, console output can be very valuable.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

CHAPTER 4

Guidebook

Warning: This section is a work in progress.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

4.1 Case studies

Warning: This page is a work in progress.

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

4.2 Segments

Warning: This section is a work in progress. It has been ported from an older document and is very many years out of date.

Note: We may rename "segments" to something else, e.g. "slices."

Notes on street segments. See Issue #7 for discussion around more street segments.

4.2.1 Automobile / Drive lanes

Typical drive lane

- **Subtypes:** Inbound, outbound
- **Default width:** 10 ft (3.0m)
- **Minimum width:** 9 ft (2.7m)
- **Maximum width:** 12 ft (3.6m)
- **References:**
  - AASHTO 2001 Green Book - Chapter 4 - Lane Widths (pp 315-316)
  - NCHRP Report 500, Volume 10: A Guide for Reducing Collisions Involving Pedestrians - section V-48
  - ITE Designing Walkable Urban Thoroughfares (2010)
  - Walkable City by Jeff Speck (2012) - Step 5: "Protect the Pedestrian"

Lane widths typically fall between 2.7m (9ft) and 3.6m (12ft). For urban contexts where Streetmix is most useful, the 3.0m (10ft) lane width is most ideal. Some guides allow for 3.3m (11ft) lanes, but we will follow Jeff Speck's lead (and other progressive planners) in setting 10ft widths as typical.

There may be outlying scenarios where wider lanes are needed, but these should never be considered the default: a 12-ft lane may be desirable to provide passing clearances for large commercial vehicles on two-lane highways, and widths less than 10-ft can be more desirable for urban and residential areas where pedestrian crossings exist and lower traffic speeds are needed.

When there is more than one lane in a direction, sometimes the outside drive lane can be wider than inside drive lanes, and even increased beyond the 12-ft maximum range, to accommodate larger vehicles (such as trucks) as well as bicycles (see also sharrow). Note that when space allows, a dedicated bike lane is always preferred and safer than a sharrow.

need source tying lane width to maximum car speed and safety

Design considerations

- Generally, any lane improvement that increases traffic flow (and speed) will correlate to an increase in pedestrian injuries and deaths, and reduce walkability. For instance:
  - One way streets
  - "Fat" lanes (see *Walkable City*, "Fat Lanes")
  - More lanes than needed (see *Walkable City*, "Protect the Pedestrian")

Typical turn lanes

- **Subtypes:** Inbound, outbound, combined with through lane
- **Default width:** 10 ft (3.0m) (same as typical lane)
- **Minimum width:** 10 ft (3.0m) (per AASHTO 2001)
- **Maximum width:** 12 ft (4.8m) (same as typical lane)
- **References:** AASHTO 2001 Green Book - Chapter 4 - Lane Widths (pp 315-316)

Turn lanes are typically provided on major streets to provide a refuge for cars turning into cross-streets, side streets, or sometimes even into a parking lot. At intersections, space for turn lanes are often created by removing a parking lane, or by converting a through lane to a turn lane (and dual-purpose through/turn lanes exist as well). If the street is particularly wide, a median can be provided between the directional lanes, and the turn lane occupies the space where the median was when needed. On streets with a lot of possible turns, using a continuous center turn lane is common.

Center turn lane

- **Full name:** Continuous two-way left-hand turn lane
- **Default width:** 12 ft (3.6m) (Used as a default for an 80' collector road by the Las Vegas Unified Development Code 19.04.190)
- **Minimum width:** 10 ft (3.0m) (per AASHTO 2001)
- **Maximum width:** 16 ft (4.8m) (per AASHTO 2001)
- **Markings:** yellow solid line with yellow dashed lines on the inside
- **References:** AASHTO 2001 Green Book - Chapter 4 - Lane Widths (pp 315-316)

A center turn lane is a special type of turn lane that is continuous along the street, allowing refuge for turns along the entire street and not just where a specific turn lane area is provided.

Adding a center turn lane is a good "road diet" tool for turning dangerous four-lane roads into a three-lane road (two lanes in either direction, with the center turn lane as the third lane). However, when street widths are fixed, it's usually not a good idea to take away some other kind of amenity (like parking lanes or bike lanes) to provide the turn lane. (for more information, see *Walkable City* "A Road Too Far")

Parallel parking lanes

- **Subtypes:** inbound, outbound
- **Default width:** 8 ft (Source: Complete Streets Chicago 3.2.1 "Cross Section Assemblage" (2013))
- **Minimum width:** 7 ft (Source: Complete Streets Chicago 3.2.1 "Cross Section Assemblage" (2013))
- **Maximum width:** 10 ft
- **Markings:** unmarked, marked with "T"

On-street automobile parking lanes are common and have been generally included wherever there is room. In residential areas, the minimum width is 7 ft and are usually unmarked. On commercial streets the width is usually minimum of 8 ft and may be marked so that the number of parking spaces fixed to a specific amount, and can be tied to parking meters.

Some municipal guidelines look at drive lanes and parking lanes together, especially when parking lanes are unmarked. For instance, the Las Vegas Unified Development Code designs residential streets to be 17'-18' (so that it reasonably breaks down into 10' drive, + 7'-8' parking). Note that the Chicago Complete Streets Plan, cited above, requires an 18' minimum - this creates a 11' drive lane with a 7' parking lane in a residential zone, which - given that 10' is plenty particularly in residential streets where speeds should not be encouraged to go above 20-25mph - we should not accept as ideal.

Other types of parking lanes

Guidelines and helpful plan diagrams for these exist in certain municipalities, but I have to remember which ones and then track them down.

Perpendicular

- **Default width:** 18-20 ft
- **Minimum width:** 14-18 ft
  - A car's length is usually more than 14 ft but some municipalities allow the front of cars to overhang curbs.
- **Maximum width:** 22 ft

A situation where cars are allowed to park perpendicular to the curb. Mostly seen on extremely wide commercial corridors.

Angled

Like perpendicular, but angled parking takes up less width. The angle of the parking space will influence its width.

4.2.2 Bike facilities

Generally, for Streetmix, bike lane and bike facilities will adopt the NACTO Urban Bikeway Design Guide, which provides in-depth discussion on the different types of bike paths that are available, based on guidelines set out in the **AASHTO Guide for the Development of Bicycle Facilities (1999)**. Note that our print copy is the April 2011, but we should be using the second edition of the NACTO guide released in 2012.

Typical bike lane

- **Subtypes:** Inbound, outbound
- **Default width:** 6 ft (Source: NACTO text, diagram)
  - Note that many practitioners (such as *Walkable City*'s Jeff Speck) follow AASHTO guidelines for default bike lane width, which is 5 ft. NACTO allows for it but prefers to advocate for 6 ft wherever possible.
- **Minimum width:** 3 ft (Source: see above)
- **Maximum width:** 8 ft (Source: ?)

Bike lanes designate an exclusive space for bicyclists through the use of pavement markings and signage, and are usually located adjacent to motor vehicle travel lanes and flows in the same direction as motor vehicle traffic. Bike lanes are typically on the right side of the street, between the adjacent travel lane and curb or parking lane, although in some countries the preferred location is between the parking lane and the curb. For other buffer types, see below.

Bicycles are a desirable alternate form of transportation to the automobile because it is a form of personal vehicle that has a lot of benefits, which we won't describe here. Providing bike lanes allows planners to be equitable to different forms of transportation while reducing a carbon footprint and also allows for more passengers to occupy a road, and separating the lanes helps bicyclists be safe and comfortable riding on the street. It also reminds car drivers that bicyclists are present. Bicyclists may leave the bike lane to pass other bicyclists, make left turns, avoid obstacles or debris, and avoid other conflicts with other users of the street.

Shared lane markings, or "sharrows" (bike lane + drive lane)

- **Subtypes:** Inbound, outbound
- **Default width:** 14 ft (4.2m) (Source: AASHTO Green Book 2011, page 100; Complete Streets Chicago 3.2.1 "Cross Section Assemblage" (2013))

- **Minimum width:** 12 ft (3.6m) (Source: AASHTO Green Book 2011, page 316)
  - AASHTO's actual suggestion of 12-13 ft (3.6-3.9m) describes a situation where an outside drive lane is slightly wider than an inside drive lane only to allow for bicycles. Presumably, drivers are allowed or expected to pass bicyclists on the inside lane.
- **Maximum width:** 14 ft (at 15 ft, you may as well put in a 10' lane and a 5' bike lane.)

Sharrows are a strategy where additional road space is available, and you want to get a bike lane in there somehow, but you can't really stripe it without giving both the drive lane and bike lane less than the standard width, so it is combined. This is a "shared lane" scenario.

In a California study, researchers found that drivers tend to prefer real bike lanes to keep bikers separate from the cars, and dislike sharrows. Bikers, on the other hand, prefer sharrows to nothing at all. (Source: Do All Roadway Users Want the Same Things? Results from a Roadway Design Survey of Pedestrians, Drivers, Bicyclists, and Transit Users in the Bay Area by Rebecca L. Sanders and Jill Cooper, 2013)

Cycletrack (bike lane + median or buffer)

- **Subtypes:** One-way, two-way
- Painted buffer. 2 solid white lines with diagonal hatching. 2 feet minimum because it would be impractical to paint a buffer area less than this. 3-feet buffer seems to be a pretty appropriate starting default. Buffer can be included with the bike lane as the total "bike lane width" (so a 2-ft buffer + 5-ft bike lane, or 3-ft buffer + 4-ft bike lane = 7-ft bike lane).
-
  - Source: NACTO Urban Bikeway Design Guide (April 2011) pp 20-22

Other types of bike lanes

For discussion around other types of bike lane placements, such as left-hand side bike lanes, or contra-flow bike lanes (both of which are permissible in Streetmix) see the NACTO Urban Bikeway Design Guide.

Bike parking

4.2.3 Other road infrastructure

Medians (divider only)

- **Subtypes:** Curb only, planted

1.2m - 24m (4ft - 80ft) or more - AASHTO Green Book p341. Does this apply only to highways? pp341-343 has information

Pedestrian median

A median that is at least 6' wide is good for pedestrian crossings. When designed as "refuge islands" on two-way streets, it allows pedestrians to cross one direction of traffic at a time in a safer way. While 6' is the minimum needed to accommodate the length of a bicycle or a person pushing a stroller, the preferred width of a refuge would be between 8 to 10 feet, and even wider where there are more pedestrians than usual. Trees, landscaping, signage, lights, or other "street furniture" could also be installed on the median to make the refuge obvious to drivers.

Source: Complete Streets Chicago pg 106.

Shoulders and gutters

Not presently planned for Streetmix.

4.2.4 Public Transportation

Trolleys, streetcars, light rail

- **Subtypes:** Different car types; inbound, outbound; separated infrastructure or in drive lane
- **Default width:** 10 ft (same as drive lane)
- **Minimum width:** 10 ft
  - According to streetcar.org, the widths of most streetcars are between 8'-0" to 9'-0". In the absence (so far) of guidelines defining streetcar width, we can assume that 10' should be the minimum and can remain the default.
- **Maximum width:** none

Buses

Trains

4.2.5 Sidewalks (pedestrian zones)

Sidewalk

Sidewalks are essential for pedestrian activity, and when they are made as safe and welcoming as possible for people, it helps to create lively, vibrant streets.

- 2.4m (8-ft) minimum (but total including buffer)
- residential areas 4ft-8ft (1.2-2.4m)
- buffers: 2ft

AASHTO Green Book pp361-362

Sidewalk border

SIDEWALK "BORDER" (aka planting strip, buffer, setc)
minimum 2ft (0.6m)

4.2.6 Sidewalk infrastructure (furnishing zones)

Street Trees

Generic types

Probably no need to be specific about species, which is something landscape architects do all the time.

- Palm tree

- Cherry tree (flowering)

Here's some really good examples for San Francisco.

4.2.7 Miscellaneous

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

4.3 Vehicles

This page will categorize the types of vehicles present or planned in Streetmix, and notes resources used for research or for future reference.

Warning: This section is a work in progress.

4.3.1 List of vehicles

- Automobiles (cars)
  - Personal automobile
  - Taxi
  - Rideshare
  - Autonomous vehicle
- Bicycle
- Motorcycle (with sidecar)
- Truck
- Food truck
- Streetcar
- Light rail
- Bus
- Electric scooter
- Freight train
- Magic carpet

Note: Although pedestrians have a definition in NUMO's vehicle attribute framework, individual persons — including those using wheeled mobility aids such as wheelchairs — are not classified in Streetmix as vehicles. Instead, we use the placeholder concept of "the pedestrian mode of travel" rather than individual persons representing a pedestrian "vehicle."

Attention: (August 2021) Streetmix documentation has now moved! Please update your bookmarks to https://docs.streetmix.net/.

CHAPTER 5

Contact

Whether you need help, have feedback, or want to work with us, we have different methods of getting in touch.

5.1 GitHub

The primary Streetmix code repository is on GitHub. Most code-related conversations happen here. You can file bug reports or feature requests on the issues page.

5.2 Discord

We have a public chat room on Discord. Contributors to Streetmix coordinate here, and it's a great place to hang out with others in the Streetmix community. If you have seen a serious bug on Streetmix, the best place to get a quick response is to notify us on Discord first.

5.3 Twitter

You can reach us on Twitter at @streetmix.

5.4 E-mail

For business inquiries, or to report code of conduct violations, email us at hello@streetmix.net.

Index