
Tabbycat Documentation

Versión 2.3.3

Philip Belesky, Chuan-Zheng Lee

13 de junio de 2020

Installation

1. Installing on Heroku	3
2. Local Installations	11
3. Installing Locally using Docker	15
4. Installing Locally on Linux	19
5. Installing Locally on macOS	25
6. Installing Locally on Linux on Windows (WSL)	31
7. Installing Locally on Windows	33
8. Importing Initial Data	41
9. Starting a Tournament	45
10. Running a Tournament	49
11. Finishing a Tournament	55
12. Tournament Logistics	57
13. Tab Software Comparisons	79
14. Scaling & Performance on Heroku	87
15. Upgrading Tabbycat	95
16. Adjudicator Feedback	97
17. Adjudicator Allocation	101
18. Backups	107
19. Breaks and Break Rounds	109
20. Check-Ins	113

21. Entering Ballots and Feedback	119
22. Draw Generation	125
23. Draw Generation (BP)	131
24. Preformed Panels	137
25. Printing Ballots and Feedback	141
26. Sending Notifications	143
27. Standings Rules	147
28. Team Code Names	151
29. Tournament Data Importers	153
30. User Accounts	159
31. Venue Constraints	161
32. Support	165
33. Authors & Acknowledgements	167
34. Change Log	169
35. Contributing	189
36. Licence Information	195
37. Tournament History	197

Tabbycat is a draw tabulation system for parliamentary debate tournaments. It supports two-team formats such as Australs, World Schools, Asians, Australian Easterns and all New Zealand formats, as well as British Parliamentary (WUDC). It is also highly configurable, supporting many custom variations of the above formats too. If you're looking for a general overview of the software, check out our [README on Github](#).

Installing on Heroku

When running Tabbycat on the internet, we set it up on [Heroku](#). The project is set up to be good to go on Heroku, and it works well for us, so if you'd like to run it online, we recommend that you do the same. Naturally, this requires you to have a Heroku account.

There are two ways to do this: a **short way** and a **long way**. Most people should use the short way. The long way requires some familiarity with command-line interfaces and Git, and requires a *local installation* as a prerequisite, but makes it easier to *upgrade versions* later on and (unlike the short way) allows you to import data from CSV files.

1.1 The short way

Click this button:

This is the easiest way to deploy an instance of Tabbycat online. It requires no technical background.

If you don't already have a Heroku account, it'll prompt you to create one. Once you're logged in to Heroku, choose a name for your installation, then scroll down and click **Deploy**. Once it's finished, click **View** and follow the prompts. Once finished, open the site and from there you can easily set up a demo data set (if you just want to learn Tabbycat) or use the data importer to set up a real tournament.

Nota: During the setup process, Heroku will ask you to verify your account by adding a credit card. A standard Tabbycat site *will not charge* your card — charges only accrue if you deliberately add a paid service in the Heroku dashboard.

If you can't access a credit card, you can instead install a limited version, which we call «Tabbykitten». However, Tabbykitten cannot send any e-mails or handle as much public traffic. We therefore strongly recommend it only as a last resort, and even then only for small tournaments. [Use this link to set up a Tabbykitten site.](#)

1.2 The long way

The long way sets you up with more control over your environment. Because you'll clone [our GitHub repository](#), it'll be easier for you to *upgrade your app* when a new version is released. You'll also have the flexibility to make and contribute updates to the source code. We recommend it if you have experience with Git. It's also easier with this method to import CSV files using the command-line importer, so if you have a very large tournament, this might make importing initial data easier.

We've tested these instructions successfully on Windows, Linux and macOS.

1.2.1 Requisite technical background

You need to have at least a passing familiarity with command-line interfaces to get through the longer traditional method. We'll talk you through the rest.

When we say «command shell», on Windows we mean **Command Prompt**, and on Linux and macOS we mean **Terminal** (or your favourite command shell).

Advanced users

Tabbycat is a [Django](#) project. As such, it can be installed on any web platform that supports Django, using any SQL system that Django supports. Just be aware that we haven't tried any other platform.

1.2.2 Short version of the long way

Advertencia: We provide a «short version» for experienced users. Don't just copy and paste these commands before you understand what they do! If things aren't set up perfectly they can fail, so it's important to supervise them the first time you do them. If this is all new to you, read the long version of the instructions below.

```
git clone https://github.com/TabbycatDebate/tabbycat.git
cd tabbycat
git checkout master
python deploy_heroku.py yourappname
```

If you want to *import tournament data* from CSV files, *install Tabbycat locally*, put your CSV files in `data/yourtournamentname`, then:

```
createdb yourlocaldatabasename      # Your settings_local.py file must point here from ↵
↵ DATABASES
dj migrate
dj createsuperuser
dj importtournament yourtournamentname --name "Your Tournament Name" --short-name
↵ "Tournament"
heroku maintenance:on
heroku pg:reset
heroku pg:push yourlocaldatabasename DATABASE
heroku maintenance:off
```


1.2.3 1. Install dependencies

- a. Install the [Heroku Command Line Interface \(CLI\)](#), then log in with the command `heroku login`.
- b. If you don't already have **Git**, follow the [instructions on the GitHub website](#) to set up Git.

1.2.4 2. Set up a local installation

If you don't already have a local installation, follow the instructions on the page for your operating system, listed below, to set up a local installation.

Atención: When downloading the source code, you **must** take the option involving cloning the GitHub repository using Git. In the macOS and Windows instructions, this means the option described in the «Advanced users» box. To do so, use these commands:

```
$ git clone https://github.com/TabbycatDebate/tabbycat.git
$ git checkout master
```

Do not download the .tar.gz or .zip file and extract it.

- [Installing Locally on Linux](#)
- [Installing Locally on macOS](#)
- [Installing Locally on Linux on Windows \(WSL\)](#)
- [Installing Locally on Windows](#)

If you do already have a local installation, update to the latest version using:

```
$ git checkout master
$ git pull
```

Advanced users

It's not *strictly* necessary to have a fully functional local installation if you don't want to import data from CSV files. But it certainly helps.

1.2.5 3. Deploy to Heroku

- a. Navigate to your Tabbycat directory:

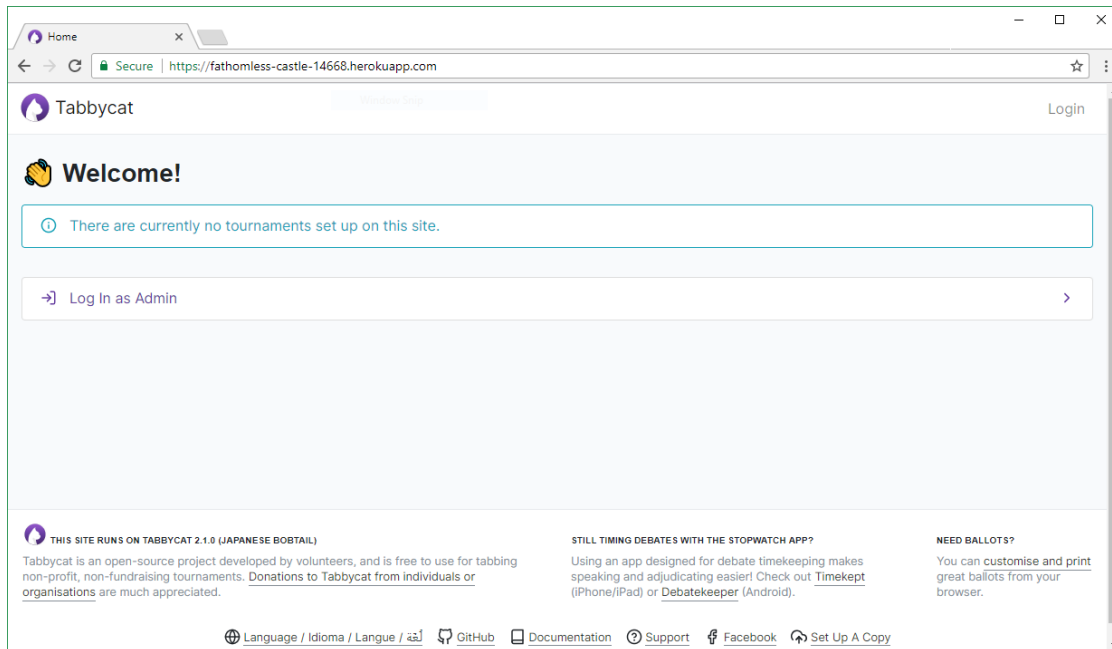
```
cd path/to/my/tabbycat/directory
```

- b. Run the script to deploy the app to Heroku. Replace `yourappname` with your preferred URL. Your website will be at `yourappname.herokuapp.com`.

```
python deploy_heroku.py yourappname
```

This script has other options that you might find useful. Run `python deploy_heroku.py --help` for details.

When this script finishes, it will open the app in your browser. It should look something like this:



1.2.6 4. Import tournament data locally

Nota: Steps 4 and 5 are optional; there are other methods of *importing data*. However the following method is most useful for large tournaments where manual entry would be tedious.

Nota: Step 4 is the same as the process described in *The importtournament command on local installations*.

- Place your CSV files in `data/yourtournamentname`, as described in *Importing Initial Data*.
- Create a new, blank local database:

```
createdb yourlocaldatabasename
```

It's normally easiest to name your local database after your app name, so that if you have multiple sites, you know which one relates to which.

Reconfigure `DATABASES` in your `settings_local.py` file to point to this new database.

- Activate your virtual environment:

```
source venv/bin/activate
```

- Run initial migrations on your blank local database:

```
dj migrate
dj createsuperuser
```

- Import your tournament data into your blank local database:

```
dj importtournament yourtournamentname --name "Your Tournament Name" --short-name
↪ "Tournament"
```

If your data's not clean, it might take a few attempts to get this right. We recommend either destroying and recreating the database (`dropdb`, `createdb`), or wiping it using `dj flush`, before retrying.

- f. Check it looks like how you expect it to look, by starting your local installation:

```
dj runserver
```

1.2.7 5. Push the local database to Heroku

Once you're happy with how your local import went, you can push the local database to Heroku.

Peligro: This step wipes the Heroku database clean, and replaces it with the contents of your local database. If you have any data on the Heroku site that isn't also in your local database, **that data will be lost** and will not be recoverable.

Truco: If you have multiple Heroku sites, you may find that the `heroku` commands refuse to run, prompting you to specify an app. If so, add `--app yourappname` to each `heroku` command.

- a. Enable maintenance mode. This takes the site offline, to ensure that no-one can possibly create or change any data on the site while you're pushing a new database up:

```
heroku maintenance:on
```

- b. Reset the database. (Caution: This permanently deletes all information on your Heroku database!)

```
heroku pg:reset
```

- c. Push your local database to Heroku:

```
heroku pg:push yourlocaldatabasename DATABASE
```

You might need to specify your local PostgreSQL credentials by adding `PGUSER=yourusername` `PGPASSWORD=*****` `PGHOST=localhost` to the *beginning* of that command. (This sets environment variables to those values for the duration of that one command.)

- d. Disable maintenance mode:

```
heroku maintenance:off
```

1.3 Heroku options you may want to change

If you have a large tournament, you may want to customize your Heroku app. This section provides some guidance on upgrades and settings you may wish to consider. Some of these configurations require you to have the [Heroku Command Line Interface \(CLI\)](#) installed.

1.3.1 Upgrading your database size

The free plan of [Heroku Postgres](#), «Hobby Dev», should work for most small tournaments. For large tournaments, however, you may find that you exceed the 10,000-row limit of this plan. It's difficult to give general guidance on how

many rows you're likely to use, because it depends on which features of Tabbycat you use (*e.g.*, if you use adjudicator feedback). But to give some idea:

- Australs 2016, which had 74 teams, 8 preliminary rounds and heavily used adjudicator feedback, ended up at around 30,000 rows.
- The Asia BP championships 2017 had 100 teams, 6 preliminary rounds, and mandatory feedback (i.e. 100 % return rates) used 15,000 rows.
- A 3 vs 3 tournament with 54 teams, 5 preliminary rounds, and which only lightly used adjudicator feedback ended up using around 4,500 rows

If you need more than 10,000 rows, you'll need to upgrade to a paid Heroku Postgres Plan. The 10,000,000 rows allowed in the lowest paid plan, «Hobby Basic», should certainly be more than sufficient.

If you're not sure, you can always start at Hobby Dev—just be prepared to [upgrade](#) during the tournament if you run close to capacity.

1.3.2 Custom domain names

Your Heroku app will be available at `yourappname.herokuapp.com`. You may want it to be a subdomain of your tournament's website, like `tab.australasians2015.org`. If so, you'll need to configure your custom domain and SSL. Instructions for both are in the Heroku Dev Center:

- [Custom Domain Names for Apps](#)
- [Heroku SSL](#)

The custom domain name basically requires two things: a DNS CNAME entry on your website targeting `yourappname.herokuapp.com`, and the custom domain configured on Heroku using `heroku domains:add tab.yourwebsite.com`. You'll also need to provide an SSL certificate for your custom domain and add it using the `heroku certs:add` command.

1.3.3 HTTPS

All Tabbycat sites deployed to Heroku redirect all traffic to HTTPS by default.

For a myriad of reasons, we strongly advise against disabling this. But if for some reason you need to run on plain HTTP, you can do this by setting the `DISABLE_HTTPS_REDIRECTS` config variable in Heroku to `disable` (see [Heroku documentation on config vars](#)). The value of the config var must be `disable`; if it's anything else, HTTPS redirects will remain in place.

Truco: Most modern browsers, after having been redirected by a site to HTTPS once, remember that that site requires HTTPS and go there for all subsequent visits even if the user typed in a plain `http://` address. It may do this because it cached the HTTP 301 permanent redirect, stored an HSTS entry and/or tagged its session cookie to require HTTPS. If, after disabling HTTPS on your Tabbycat site, you find that you're still being redirected to HTTPS, first try a browser or computer that *hasn't* visited the site before. If that works, then remove the relevant entry from your (original) browser's cache, HSTS set and cookies, and try again.

1.3.4 Time zone

If you want to change the time zone you nominated during deployment, you can do so by going to the [Heroku Dashboard](#), clicking on your app, going to the **Settings** tab, clicking **Reveal Config Vars** and changing the value of the

TIME_ZONE variable. This value must be one of the names in the IANA tz database, *e.g.* Pacific/Auckland, America/Mexico_City, Asia/Kuala_Lumpur. You can find a [list of these on Wikipedia](#) in the “TZ*” column.

1.3.5 SendGrid account details

By default, Heroku will automatically create a SendGrid account for you. For small tournaments, this should work fine. For larger ones, though, SendGrid typically doesn’t allow new accounts to send so many emails without additional vetting. This vetting is separate to the verification you did for your Heroku account, and as far as we’re aware, it can’t be done until you send your first email, by which time it’s probably too late.

If you’re running a large tournament, you may wish to use your own SendGrid account instead. The free tier probably won’t suffice after the trial period, but the Essentials tier should be more than adequate. If you’re a student and have the [GitHub Education Pack](#), you might find the SendGrid plan here useful.

If you set up and use your own SendGrid account, you can remove the SendGrid add-on from your Heroku app. The SendGrid add-on is only necessary if you wish to use Heroku’s auto-created SendGrid account.

To set up your app to use your own SendGrid account:

1. [Sign up for a SendGrid account](#), if you don’t already have one.
2. [Create an API key](#) in your SendGrid account.

There are [instructions for how to do this in the SendGrid documentation](#). The only permission that is needed is the «Mail Send» permission, so you can turn off all others if you want to be safe.

3. Set the following config vars in Heroku Dashboard (or using the Heroku CLI, if you have it):

- SENDGRID_USERNAME should be set to `apikey` (not your username).
- SENDGRID_PASSWORD should be set to your API key, which will start with `SG*****`.

Advertencia: The [Heroku SendGrid instructions](#) to do something with `SENDGRID_API_KEY` are **incorrect**. We figured this out by contacting SendGrid support staff. Use the above config vars instead.

1.4 Upgrading an existing Heroku app

Nota: For most users, we recommend starting a new site for every tournament, when you set up the tab for that tournament. There’s generally not a pressing need to upgrade Tabbycat after a tournament is concluded, and every time you deploy a new site, you’ll be using the latest version at the time of deployment.

To upgrade an existing Heroku-based Tabbycat app to the latest version, you need to *deploy* the current version of Tabbycat to your Heroku app. There are several ways to do this. We list one below, primarily targeted at users with some background in Git.

The essence of it is that you need to [create a Git remote](#) for your Heroku app (if you don’t already have one), then [push to it](#).

Atención: You should **always** *back up your database* before upgrading Tabbycat.

You’ll need both Git and the Heroku CLI, and you’ll need to be logged in to the Heroku CLI already.

1. Take a backup of your database:

```
$ heroku pg:backups:capture
```

2. If you haven't already, clone our Git repository and check out the master branch:

```
$ git clone https:\\\\github.com/TabbycatDebate/tabbycat.git
$ git checkout master
```

If you've already cloned our Git repository, don't forget to pull so you're up to date:

```
$ git checkout master
$ git pull
```

3. Check to see if you have a Git remote already in place:

```
$ git remote -v
heroku  https://git.heroku.com/mytournament2018.git (fetch)
heroku  https://git.heroku.com/mytournament2018.git (push)
```

If you do, the name of the remote will be on the left (*heroku* in the above example), and the URL of your Git repository will be on the right. In the example above, our Tabbycat site URL would be `mytournament2018.herokuapp.com`; the Git remote URL is then `https://git.heroku.com/mytournament2018.git`.

If a Git remote URL for your Tabbycat site *doesn't* appear, then create one:

```
$ heroku git:remote --app mytournament2018 --remote heroku
set git remote heroku to https://git.heroku.com/mytournament2018.git
```

Truco: If you tab many tournaments, it'll probably be helpful to use a name other than `heroku` (say, `mytournament2018`), so that you can manage multiple tournaments.

4. Push to Heroku:

```
$ git push heroku master
```

This will take a while to complete.

2.1 What is a local installation?

Tabbycat is a web-based system: it's designed to run as a web site. However, instead of installing it on a web server, you can install it on your computer, serving web pages to yourself. This is called a local installation.

Then, when you open your browser, you can use Tabbycat like any other web site. The only difference is that this one is on **your** computer, not some computer in a data centre in a far-away land. In effect, you are getting your computer to behave like a web server.

2.2 Should I use a local installation?

In most cases, you should make an online Tabbycat installation by *setting up an instance on Heroku*. This has a number of major advantages:

- The installation process is easier.
- You can enter ballots and manage your tournament from multiple computers.
- Participants can access the draw, motions, results and more online.
- Heroku's data centers are less likely to fail than your computer is.
- Heroku e-mails Tabbycat's developers error reports to help us fix bugs.

In some cases, you might have a good reason to use a local installation. We can think of just one such reason: If you won't have access to the internet at your tournament, or if internet access will be flaky, then you should use a local installation.

Atención: You'll need internet access to download dependencies during the local installation process. So if you're not expecting to have reliable internet access at your tournament, be sure to have Tabbycat installed *before* you get there!

Advanced users

Tabbycat is a [Django](#) project, so if you have your own preferred method of running Django projects, you can also do that. Just be aware that we haven't tried it.

2.3 Okay, so how do I do it?

The easiest option is to *install Tabbycat using Docker*. This method should work across all operating systems and is by far the easiest way to get a local copy running.

If installing using Docker does not work, or if you want to be able to modify Tabbycat's code we also have a number of instructions for manually setting up a copy of Tabbycat. There instructions are here:

- [Installing Locally on Linux](#)
- [Installing Locally on macOS](#)
- [Installing Locally on Linux on Windows \(WSL\)](#)
- [Installing Locally on Windows](#)

2.4 Advanced uses

2.4.1 Can others access my local install?

Local installations can also take advantage of multiple-computer site access, including data entry—it's just takes more work than a Heroku installation to set up.

Since a local installation is just having your computer run a web server, it can serve other computers too. You can make this work even if you don't have internet access: all you need is a router that you can use to connect a few computers together. Then other computers on your local network can access the Tabbycat site hosted on your computer. We did this at Victoria Australs 2012.

We don't provide detailed instructions for this; we leave it for advanced users to set up themselves. As a rough guide:

- You need to pass in your computer's IP address and port to the `runserver` command, for example, if your computer (the one acting as a server) is 196.168.0.2 and you want to run it on port 8000: `django runserver 192.168.0.2:8000`
- You need to configure your firewall settings to allow incoming connections on the IP address and port you specified in that command.
- Be aware that local installs use the Django development server, whose **security is not tested**. Therefore, it's a good idea to make sure your firewall **only lets in computers on your local network** (or, if you're really paranoid, isolate the network from the internet completely).

2.4.2 Can I run an internet-accessible website from a local installation?

Probably not. Even if you disable your firewall, chances are your home router (or university router) will block any connections from the outside world to you. Even if you can though, **you really shouldn't**. The local installation uses the *Django development server*, which is a lightweight server designed for developers. Specifically, Django **does not test the security of its development server** in the way that proper web servers do. That is: It's a security risk to run a local installation as an internet-accessible site. Don't do it. [Install Tabbycat on Heroku](#) instead.

Alternatively, if you have a background in web development, you might choose to install Tabbycat on your own production server. It's a Django project, so any means of supporting Django projects should work fine.

It's safe to run on a small, isolated network (see above) with your firewall correctly configured because you presumably trust everyone you let on the network!

Installing Locally using Docker

Is this the best install method for you?

In most cases, we recommend doing an *internet-based installation on Heroku* instead. If you decide to do a local installation, be sure to read our page on *local installations* to help you understand what's going on, particularly this section: *Should I use a local installation?*

Atención: If you need an offline copy of Tabbycat, installing using Docker should be simpler and quicker than using the «Install Locally on...» instructions for your operating system. However if a Docker installation doesn't work as outlined below, it's harder to address what isn't working. If you encounter any problems with Docker, we recommend using the «Install Locally on...» option as a fallback, but if you need to do so, [please also report the issue you're having on GitHub](#) or *contact the developers*.

Docker is an application that makes it very easy to load and run a specific collection of software. It allows us to bundle everything necessary to run Tabbycat into a single package rather than have users install everything needed step-by-step. Once set up, Docker will allow you to start and stop a webserver (that in turn runs Tabbycat) on your computer whenever you want and without the need for internet access.

3.1 1. Download Tabbycat

1. [Go to the page for our latest release.](#)
2. Download the zip or tar.gz file.
3. Extract all files in it to a folder of your choice.

3.2 2. Install Docker

3.2.1 If using macOS

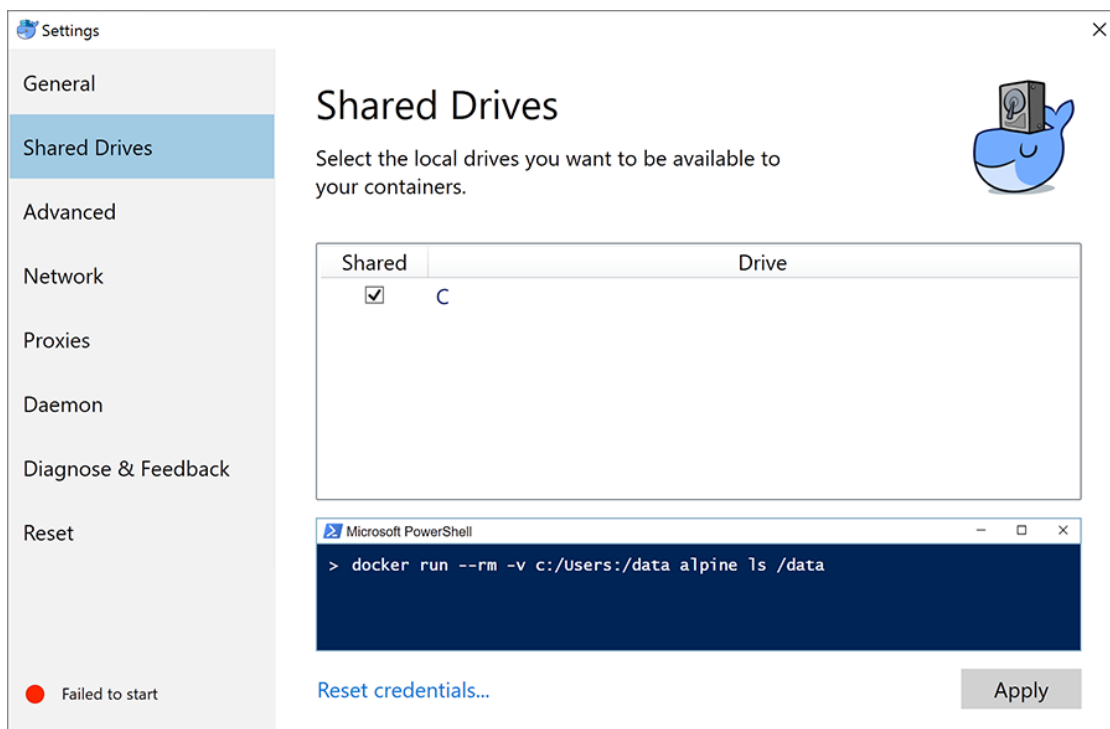
- Install the **Docker Desktop for Mac** from the [Docker store page](#).

3.2.2 If using Linux

- Install the **Docker Engine** for your OS from the [Docker store page](#).

3.2.3 If using Windows 10 Pro, Enterprise, or Education Edition

1. Install the ****Docker Desktop for Windows*** from the [Docker store page](#).
2. Before or shortly after installing it, Docker may ask you to enable hypervisor and restart your PC. If it asks you this follow the prompts and restart as asked.
3. Once Docker has finished installing, open up the newly-installed Docker application, then right-click the app's icon (the whale) in the Taskbar.
4. From there, hit *Settings* in the menu and *Shared Drives* in the sidebar. Tick the checkbox next to your harddrive and then click *Apply*. After that has applied exit and reopen the docker app (ie right-click the taskbar icon and hit exit) and verify that the checkbox is still there.



3.2.4 If using Windows 7, Windows 8, or Windows 10 Home Edition

- Install the **Docker Toolbox** from the [Docker Toolbox downloads page](#).

Truco: Not sure which edition of Windows you have? Click Start, search for «System», and open the Control Panel item «System».

3.3 3. Run Tabbycat in Docker

3.3.1 If using the Docker App

These instructions apply if you installed the Docker App, i.e., if you are using macOS, Linux or Windows Pro/Enterprise/Education.

1. Ensure that Docker application is open (there should be a whale icon in your menu/task bar) and that it says that Docker is running.
2. Browse to the location where you extracted Tabbycat to. Open up the **bin** folder there. Within that folder:
 - If you're on macOS, press the Control key, click the icon for **osx_docker_start.command**, then choose Open from the shortcut menu.
 - If you're on Windows, open **windows_docker_start.bat**.
 - If you're on Linux, open up a terminal in the Tabbycat folder (*i.e.* the folder containing `README.md`) and run `docker-compose up`.
3. A terminal window should popup and bunch of text scroll by. If this is your first time running Docker it may take a while (30 minutes or more) to download the virtual machine. When the text has stopped scrolling by you should see a *Finished building Tabbycat!* message.
4. Open up <http://localhost:8000/> (Windows) or <http://0.0.0.0:8000> (OSX/Linux) in a browser of your choice!

Nota: If you want to reopen Tabbycat at a later time (say after restarting) repeat steps 1 through 4 here.

3.3.2 If using the Docker Toolbox

These instructions apply if you installed the Docker Toolbox, i.e., if you are using Windows 7, Windows 8 or Windows 10 Home.

1. Start the **Docker Quickstart Terminal**.
2. Run the command `docker-machine ip`. Take note of the IP address it shows, for example:

```
$ docker-machine ip
192.168.99.100
```

3. Navigate to the Tabbycat folder (*i.e.* the folder containing `README.md`) and run `docker-compose up`.
4. Open a browser and go to <http://192.168.99.100:8000/>, replacing «192.168.99.100» with whatever IP address was shown in step 2.
5. Once you're done and want to stop the Tabbycat server, press Ctrl+C, wait until the next prompt appears, and then run `docker-machine stop`.

Nota: If you want to reopen Tabbycat at a later time (say after restarting) repeat steps 1 through 4 here.

Installing Locally on Linux

Is this the best installation method for you?

In most cases, we recommend doing an *internet-based installation on Heroku* instead. If you decide to do a local installation, be sure to read our page on *local installations* to help you understand what's going on, particularly this section: *Should I use a local installation?*

If you just want to quickly set up a copy of Tabbycat to run locally on Linux, consider *installing using Docker*, which is a shorter process than the one below.

The instructions apply to both Linux, and *Linux on Windows*.

4.1 Requisite technical background

You need to be familiar with command-line interfaces to get through this comfortably. While a background in the specific tools Tabbycat uses (Python, PostgreSQL, *etc.*) will make things easier, it's not necessary: we'll talk you through the rest.

Advanced users

Tabbycat is a *Django* project, so can be installed in any manner that Django projects can normally be installed. For example, if you prefer some SQL system other than PostgreSQL, you can use it so long as it's Django-compatible. Just be aware that we haven't tried it.

4.2 Short version

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -      # add Node.  
→js source repository
```

```
sudo apt install python3-dev python3-venv postgresql-11 libpq-dev nodejs gcc_
↳g++ make
git clone https://github.com/TabbycatDebate/tabbycat.git
cd tabbycat
git checkout master
sudo -u postgres createuser myusername --createdb --pwprompt      # skip if not_
↳first time
createdb mydatabasename
```

Then create **settings/local.py** as described *below*, then:

```
python3 -m venv venv
source venv/bin/activate
pip install --upgrade pip
pip install -r ./config/requirements_core.txt
npm install
cd tabbycat
dj migrate
npm run build
dj collectstatic
dj createsuperuser
dj runserver
```

4.3 1. Install dependencies

First, you need to install all of the software on which Tabbycat depends, if you don't already have it installed.

Advanced users

These instructions are for Ubuntu, and are targeted at Ubuntu 18.04. If you have another distribution of Linux, we trust you'll know how to navigate the package manager for your distribution to install the dependencies.

4.3.1 1(a). Python

Tabbycat requires Python 3.6 or later. You probably already have Python 3.6, but you'll also need the development package in order to install Pycopg2 later. The `venv` module will come in handy too. Install:

```
$ sudo apt install python3-dev python3-venv
```

Check the version:

```
$ python3 --version
Python 3.6.2
```

Advertencia: Tabbycat does not support Python 2. You must use Python 3.6 or later.

4.3.2 1(b). PostgreSQL

PostgreSQL is a database management system.

Install PostgreSQL using the [PostgreSQL installation instructions here](#).

Normally, installing the latest stable version should be best, but if you're having issues, install the same version as the current [default version on Heroku](#), as that will be what is currently most commonly used with Tabbycat. If you're planning on pushing data between your local installation and a Heroku site, it's best to match the Heroku's current default version.

You'll also need the `libpq-dev` package in order to install Psycopg2 later:

```
$ sudo apt install libpq-dev
```

4.3.3 1(c). Node.js/NPM

Node.js is a JavaScript runtime.

Tabbycat requires Node and its package manager to compile front-end dependencies. Install using:

```
$ sudo apt install curl
$ curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
$ sudo apt install -y nodejs
$ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

4.3.4 1(d). Other development tools

Some of the Python packages require GCC, G++ and Make in order to install:

```
$ sudo apt install gcc g++ make
```

4.4 2. Get the source code

Choose either of the following two methods.

4.4.1 Method 1 (Git clone)

If you have Git, life will be easier if you clone our [GitHub repository](#):

```
$ git clone https://github.com/TabbycatDebate/tabbycat.git
$ git checkout master
```

(You can find out if you have Git using `git --version`. If you don't, you can install it using `sudo apt install git`.)

Nota: The default branch is `develop`, so you need to explicitly change the branch to `master`, which is what the `git checkout master` line does.

Advanced users

You might like to fork the repository first, to give yourself a little more freedom to make code changes on the fly (and potentially [contribute](#) them to the project).

4.4.2 Method 2 (tarball)

If you don't want to use Git, simply download and extract:

```
$ wget https://github.com/TabbycatDebate/tabbycat/archive/v2.3.3.tar.gz
$ tar xf v2.3.3.tar.gz
$ cd tabbycat-2.3.3
```

4.5 3. Set up a new database

Consejo: You can skip step 1 if this is not your first installation. Every Tabbycat installation requires its own database, but they can use the same login role if you like.

- a. Create a new user account with a password, replacing `myusername` with whatever name you prefer. If you don't know what username to pick, use `tabbycat`. Grant this user the ability to create databases, since this'll make it easier to spin up new instances of Tabbycat in the future.

```
$ sudo -u postgres createuser myusername --createdb --pwprompt
```

Truco: If you'll be running multiple instances of Tabbycat, developing, or diving into the database yourself, you might find it convenient to set up client authentication so that you don't need to do all manual operations from `sudo -u postgres`. See the [PostgreSQL documentation on client authentication](#) for more information. For example, you could add a `local all myusername md5` line to the `pg_hba.conf` file, or you could define a mapping in `pg_ident.conf` and append the `map=` option to the `local all all peer` line in `pg_hba.conf`.

- b. Create a new database, replacing `mydatabasename` with whatever name you prefer, probably the name of the tournament you're running:

```
$ createdb mydatabasename
```

4.6 4. Install Tabbycat

Almost there!

- a. Navigate to your Tabbycat directory:

```
$ cd path/to/my/tabbycat/directory
```

- b. Start a new virtual environment. We suggest the name `venv`, though it can be any name you like:

```
$ python3 -m venv venv
```

- c. Run the `activate` script. This puts you «into» the virtual environment:

```
$ source venv/bin/activate
```

- d. Install Tabbycat's requirements into your virtual environment:

```
$ pip install --upgrade pip
$ pip install -r ./config/requirements_core.txt
$ npm install
```

- e. Navigate to the **tabbycat/settings** sub folder and copy **local.example** to **local.py**. Find this part in your new **local.py**, and fill in the blanks as indicated:

```
DATABASES = {
    'default': {
        'ENGINE' : 'django.db.backends.postgresql',
        'NAME'    : '', # put your PostgreSQL database's name in here
        'USER'    : '', # put your PostgreSQL login role's user name in here
        'PASSWORD': '', # put your PostgreSQL login role's password in here
        'HOST':   'localhost',
        'PORT':   '5432',
    }
}
```

Optionally, replace the value in this line in the same file with your own time zone, as defined in the [IANA time zone database](#) (e.g., Pacific/Auckland, America/Mexico_City, Asia/Kuala_Lumpur):

```
TIME_ZONE = 'Australia/Melbourne'
```

- f. Navigate to the **tabbycat** sub-directory, initialize the database, compile the assets, and create a user account for yourself:

```
$ cd tabbycat
$ dj migrate
$ npm run build
$ dj collectstatic
$ dj createsuperuser
```

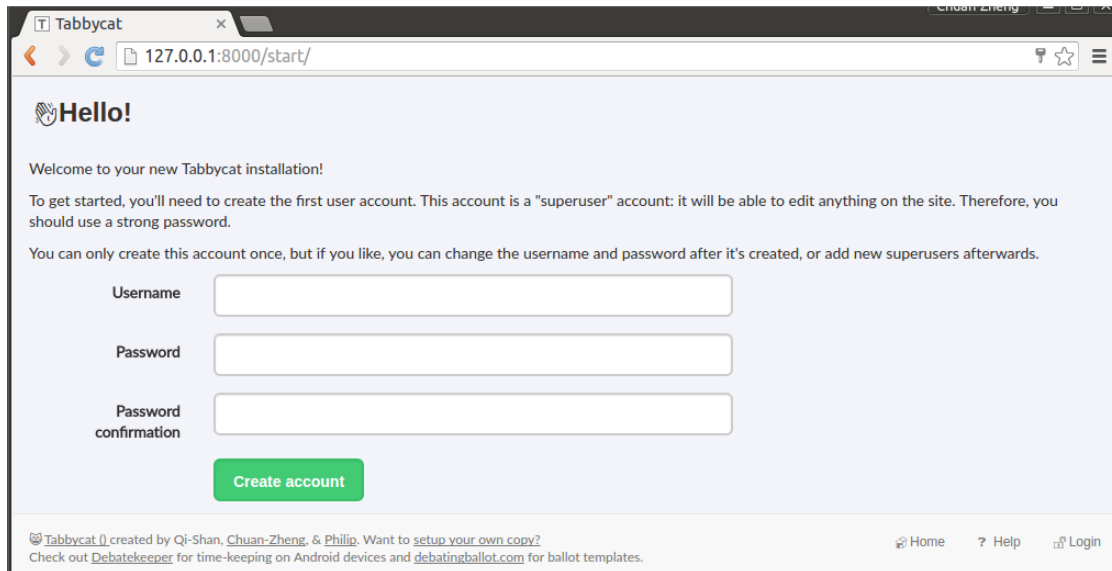
- g. Start Tabbycat!

```
$ dj runserver
```

It should show something like this:

```
serving on http://127.0.0.1:8000
```

- h. Open your browser and go to the URL printed above. (In the above example, it's <http://127.0.0.1:8000>.) It should look something like the screenshot below. If it does, great! You've successfully installed Tabbycat.



Naturally, your database is currently empty, so proceed to *importing initial data*.

4.7 Starting up an existing Tabbycat instance

To start your Tabbycat instance up again next time you use your computer:

```
$ cd path/to/my/tabbycat/directory
$ source venv/bin/activate
$ dj runserver
```

Installing Locally on macOS

Is this the best installation method for you?

In most cases, we recommend doing an *internet-based installation on Heroku* instead. If you decide to do a local installation, be sure to read our page on *local installations* to help you understand what's going on, particularly this section: *Should I use a local installation?*

If you just want to quickly set up a copy of Tabbycat to run locally on macOS, consider *installing using Docker*, which is a shorter process than the one below.

5.1 Requisite technical knowledge

You need to be familiar with command-line interfaces to get through this comfortably. While a background in the specific tools Tabbycat uses (Python, PostgreSQL, *etc.*) will make things easier, it's not necessary: we'll talk you through the rest. You just need to be prepared to bear with us. It'll take a while the first time, but it gets easier after that.

Every line in the instructions that begins with \$ is a command that you need to run in a **Terminal**, but without the \$: that sign is a convention used in instructions to make it clear that it is a command you need to run.

Advanced users

Tabbycat is a [Django](#) project, so can be installed in any manner that Django projects can normally be installed. For example, if you prefer some SQL system other than PostgreSQL, you can use it so long as it's Django-compatible. Just be aware that we haven't tried it.

5.2 1. Install dependencies

First, you need to install all of the software on which Tabbycat depends, if you don't already have it installed.

5.2.1 1(a). Python

Tabbycat requires Python 3.6 or later. macOS only comes with Python 2.7, so you'll need to install this. You can download the latest version from the [Python website](#).

The executable will probably be called `python3`, rather than `python`. Check:

```
$ python3 --version
Python 3.6.8
```

Advertencia: Tabbycat does not support Python 2. You must use Python 3.6 or later.

Advanced users

These instructions will use the `venv` module. If you prefer, you can use [Virtualenv](#) instead.

5.2.2 1(b). Postgres.app

Download [Postgres.app](#), move it to your Applications folder, and open it. This should place an icon in your menu bar, showing that the postgres database is running. Whenever you are running Tabbycat you'll need to have this app running.

You'll need to use the PostgreSQL command-line tools, so run the command that the Postgres.app suggests in its [installation instructions](#) for adding them to your `$PATH`. As of February 2018, it was:

```
sudo mkdir -p /etc/paths.d && echo /Applications/Postgres.app/Contents/Versions/  
↪latest/bin | sudo tee /etc/paths.d/postgresapp
```

5.2.3 1(c). Node.js/NPM

Download and run the [node.js 8 macOS Installer \(.pkg\)](#)

5.3 2. Get the source code

- a. [Go to the page for our latest release](#).
- b. Download the zip or tar.gz file.
- c. Extract all files in it to a folder of your choice.

Advanced users

If you've used Git before, you might prefer to clone [our GitHub repository](#) instead. Don't forget to check out the v2.3.3 tag or the master branch.

Even better, you might like to fork the repository first, to give yourself a little more freedom to make code changes on the fly (and potentially *contribute* them to the project).

5.4 3. Set up a new database

Consejo: You can skip steps 1–3 if this is not your first installation. Every Tabbycat installation requires its own database, but they can use the same login role if you like.

- a. Open up a copy of the Terminal app, then copy/paste or type in:

```
$ sudo mkdir -p /etc/paths.d && echo /Applications/Postgres.app/Contents/Versions/  
→latest/bin | sudo tee /etc/paths.d/postgresapp
```

- b. Hit enter then quit and reopen the Terminal app.
- c. Create a new user account with a password, replacing myusername with whatever name you prefer. If you don't know what username to pick, use tabbycat.

```
$ createuser myusername --pwprompt
```

- d. Create a new database, replacing mydatabasename with whatever name you prefer, probably the name of the tournament you're running:

```
$ createdb mydatabasename --owner myusername
```

- e. In terminal type in:

```
$ PATH="/Applications/Postgres.app/Contents/Versions/9.6/bin:$PATH"
```

5.5 4. Install Tabbycat

Almost there!

- a. Navigate to your Tabbycat directory:

```
$ cd path/to/my/tabbycat/directory
```

- b. Copy **settings/local.example** to **settings/local.py**. Find this part in your new **local.py**, and fill in the blanks as indicated:

```
DATABASES = {  
    'default': {  
        'ENGINE' : 'django.db.backends.postgresql',  
        'NAME'   : '', # put your PostgreSQL database's name in here  
        'USER'   : '', # put your PostgreSQL login role's user name in here  
        'PASSWORD': '', # put your PostgreSQL login role's password in here  
        'HOST'   : 'localhost',  
        'PORT'   : '5432',  
    }  
}
```

Optionally, replace the value in this line in the same file with your own time zone, as defined in the [IANA time zone database](#) (e.g., Pacific/Auckland, America/Mexico_City, Asia/Kuala_Lumpur):

```
TIME_ZONE = 'Australia/Melbourne'
```

- c. Ensure you are in the main Tabbycat directory (not the config folder where `settings_local.py` is and start a new virtual environment. We suggest the name `venv`, though it can be any name you like:

```
$ python3 -m venv venv
```

- d. Run the activate script. This puts you «into» the virtual environment:

```
$ source venv/bin/activate
```

- e. Install Tabbycat's requirements into your virtual environment:

```
$ pip install --upgrade pip
$ pip install -r ./config/requirements_core.txt
$ npm install
```

- f. Navigate to the **tabbycat** sub folder, initialize the database, compile the assets, and create a user account for yourself:

```
$ cd tabbycat
$ dj migrate
$ npm run build
$ dj collectstatic
$ dj createsuperuser
```

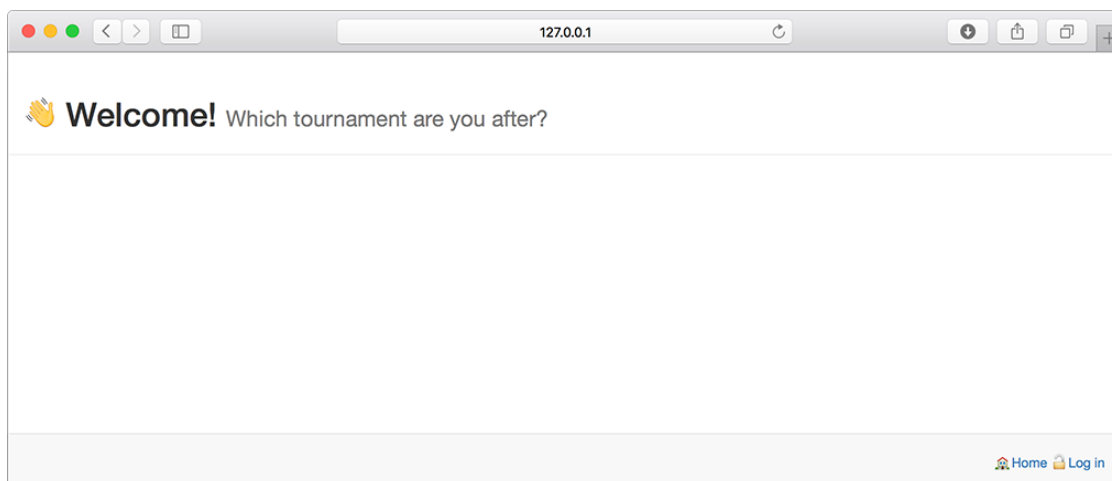
- g. Start Tabbycat!

```
$ dj runserver
```

It should show something like this:

```
serving on http://127.0.0.1:8000
```

- h. Open your browser and go to the URL printed above. (In the above example, it's <http://127.0.0.1:8000>.) It should look something like the screenshot below. If it does, great! You've successfully installed Tabbycat.



Naturally, your database is currently empty, so proceed to *importing initial data*.

5.6 Starting up an existing Tabbycat instance

To start your Tabbycat instance up again next time you use your computer:

```
$ cd path/to/my/tabbycat/directory
$ source venv/bin/activate
$ dj runserver
```

Installing Locally on Linux on Windows (WSL)

Is this the best installation method for you?

In most cases, we recommend doing an *internet-based installation on Heroku* instead. If you decide to do a local installation, be sure to read our page on *local installations* to help you understand what's going on, particularly this section: *Should I use a local installation?*

If you just want to quickly set up a copy of Tabbycat to run locally on Windows, consider *installing using Docker*, which is a shorter process than the one below.

Windows Subsystem for Linux is only available on Windows 10. If you have an older version of Windows, *install Tabbycat locally on Windows* instead.

Nota: Windows Subsystem for Linux (WSL) was taken out of beta in the *Windows 10 Fall Creators Update*, which was released in October 2017. On Windows 10 computers, we now recommend this local installation method over *installing it directly on Windows*.

6.1 Requisite technical background

It will help a lot if you have some experience with Linux, but mainly you need to be familiar with command-line interfaces, and you should be willing to install and work with the *Windows Subsystem for Linux*. You might need to be prepared to familiarise yourself with aspects of WSL not covered in these instructions. While a background in the specific tools Tabbycat uses (Python, PostgreSQL, *etc.*) will make things easier, it's not necessary: we'll talk you through the rest.

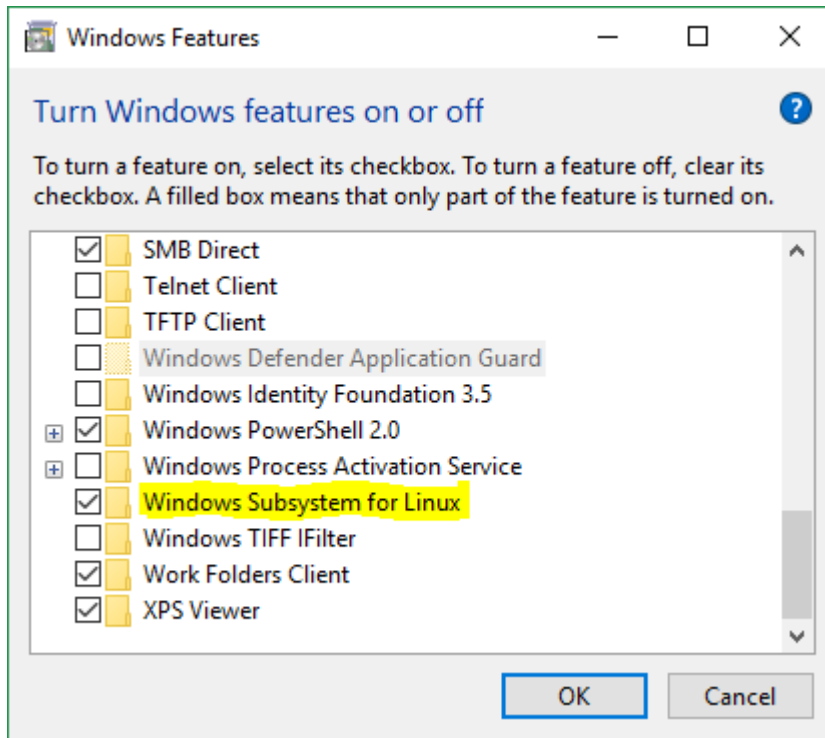
6.2 A. Install Ubuntu on Windows

If you already have a Linux distribution installed on your PC, skip to [part B](#).

First, check that you have the Fall Creators Update (build 1709). If you don't, update Windows.

Then, install the Windows Subsystem for Linux. For most people, this involves the following:

1. Enable the Windows Subsystem for Linux feature, by finding **Turn Windows features on or off** on the Start Menu, then checking the box for **Windows Subsystem for Linux** and clicking **OK**. You'll be prompted to restart your computer to make the changes take effect.



2. Install Ubuntu by finding it on the Microsoft Store. For your convenience, [here's a direct link to Ubuntu on the Microsoft Store](#).

3. Launch Ubuntu and follow the instructions. You'll be prompted to create a user account for your Ubuntu system.

Some more detailed instructions, including some troubleshooting, are [available on Microsoft's website](#).

Advanced users

You can, of course, use any Linux distribution that Windows supports. We just suggest Ubuntu because it's the most well-known (and the one that we use).

6.3 B. Install Tabbycat

You now have a Linux subsystem running on your computer, so head over to the [instructions to install Tabbycat locally on Linux](#) and follow those (in full).

Installing Locally on Windows

Is this the best installation method for you?

In most cases, we recommend doing an *internet-based installation on Heroku* instead. If you decide to do a local installation, be sure to read our page on *local installations* to help you understand what's going on, particularly this section: *Should I use a local installation?*

If you just want to quickly set up a copy of Tabbycat to run locally on Windows, consider *installing using Docker*, which is a shorter process than the one below.

If you have Windows 10 and any experience with Linux, we recommend installing it on *Linux on Windows (WSL)* instead, which is much easier than the process below.

7.1 Requisite technical background

You need to be familiar with command-line interfaces to get through this comfortably. While a background in the specific tools Tabbycat uses (Python, PostgreSQL, *etc.*) will make things easier, it's not necessary: we'll talk you through the rest. You just need to be prepared to bear with us. It'll take a while the first time, but it gets easier after that.

In these instructions, we'll use **Windows PowerShell**, a command-line interface that comes with every installation of Windows (since XP). The easiest way to find it (on Windows 7 and later) is to search for it in your Start Menu. Every line in the instructions that begins with `>` is a command that you need to run in PowerShell, but without the `>`: that sign is a convention used in instructions to make it clear that it is a command you need to run.

Advanced users

Tabbycat is a *Django* project, so can be installed in any manner that Django projects can normally be installed. For example, if you prefer some SQL system other than PostgreSQL, you can use it so long as it's Django-compatible. Just be aware that we haven't tried it.

7.2 1. Install dependencies

First, you need to install all of the software on which Tabbycat depends, if you don't already have it installed.

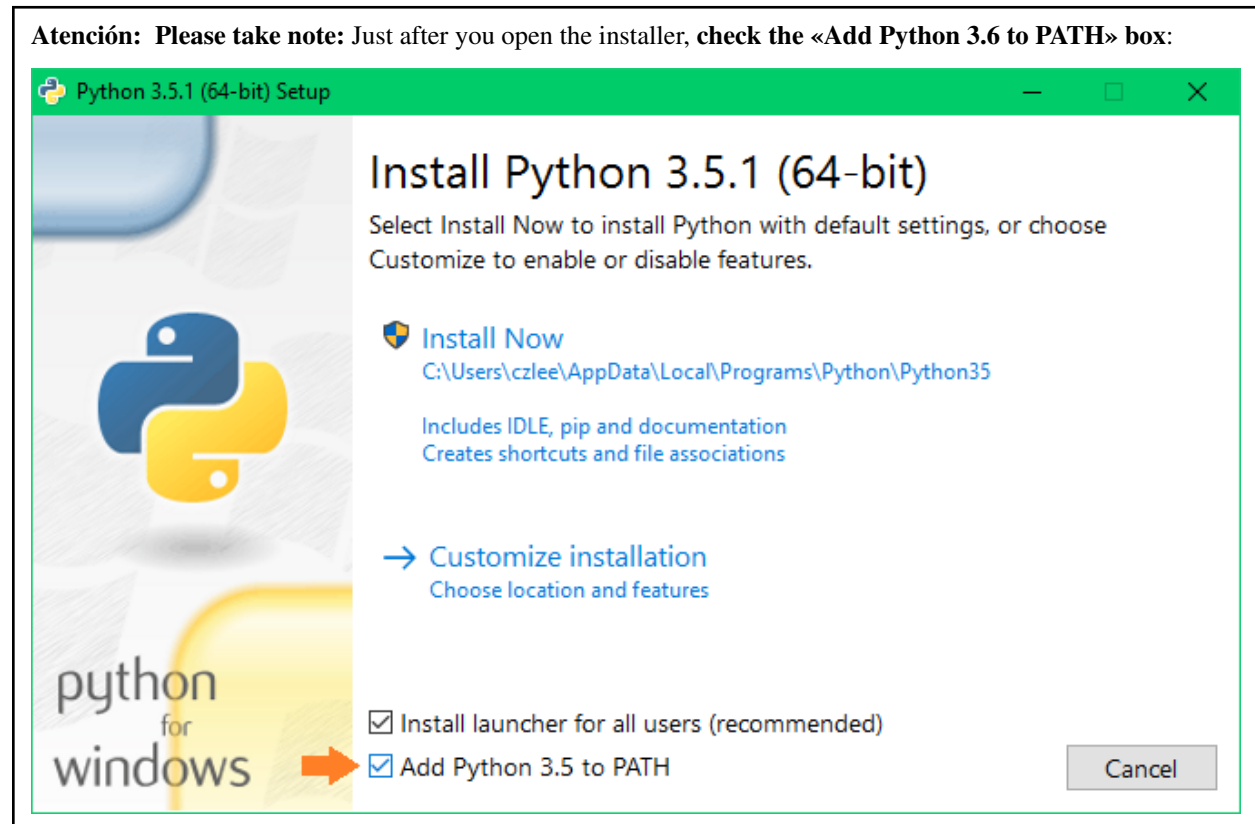
7.2.1 1(a). Python

Python is a popular programming language, and the language in which the core of Tabbycat is coded.

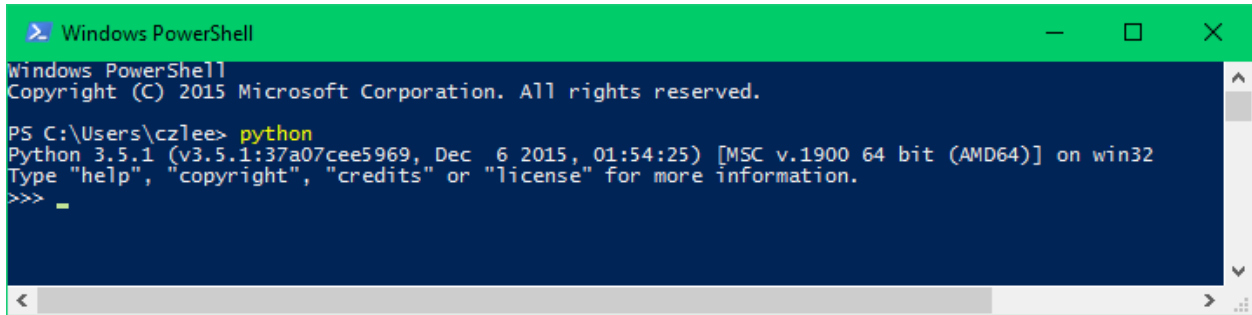
Download and install the latest version of Python 3.6 from the [Python website](#). In the installer, check the box to add Python to your PATH (see box below).

Consejo: Which file should I download?

- Regardless of if you have 64-bit or 32-bit Windows, choose the «Windows x86 executable installer».
-



To check that Python is installed correctly, open Windows PowerShell, type `python` and press Enter. It should look something like this. If you installed the 32-bit version, it will say 32 bit instead of 64 bit.



(To exit Python, type `exit()` then press Enter.)

Nota: If you already have Python, great! Some things to double-check:

- You must have at least Python 3.6 (Python 2 is not supported.)
- Your installation path must not have any spaces in it.
- If that doesn't work, note that the following must be part of your PATH environment variable: `C:\Python36;`
`C:\Python36\Scripts` (or as appropriate for your installation directory). Follow [the instructions here](#) to add this to your path.

7.2.2 1(b). PostgreSQL

PostgreSQL is a database management system.

Go to the [PostgreSQL downloads](#) page, then follow the link through to EnterpriseDB to download and install the latest version of PostgreSQL.

Truco: Once PostgreSQL is installed, the PostgreSQL service will run on your computer whenever you are using it. You might prefer to configure it so that it only runs when you want to run Tabbycat. To do this, open «Services» in your Control Panel on Windows, find the PostgreSQL service, and change its startup type to «Manual». This will tell it not to start whenever you log in. Then, if you want to run the server (so you can use Tabbycat), you can do so from «Services» by selecting the PostgreSQL service and clicking «Start the service».

7.2.3 1(c). Git

Git is a version control system.

We won't use Git directly, but Node.js (which we install in the next step) requires Git to work. So, install the latest version for Windows from the [Git website](#).

Advanced users

If you already have [GitHub Desktop](#) installed, you might think that this would be good enough. Unfortunately, it's not—GitHub Desktop installs a portable version of Git. Node.js, on the other hand, requires the `git` to be in the PATH, so it can call it directly. The easiest (but not only) way to do this is just to install Git from the link above.

7.2.4 1(d). Node.js/NPM

Node.js is a JavaScript runtime.

Download and run the node.js 8 Installer (.msi) for either [x64 versions](#) of Windows or [x86 versions](#).

7.3 2. Get the source code

- Go to the page for our latest release.
- Download the zip file.
- Extract all files in it to a folder of your choice.

Advanced users

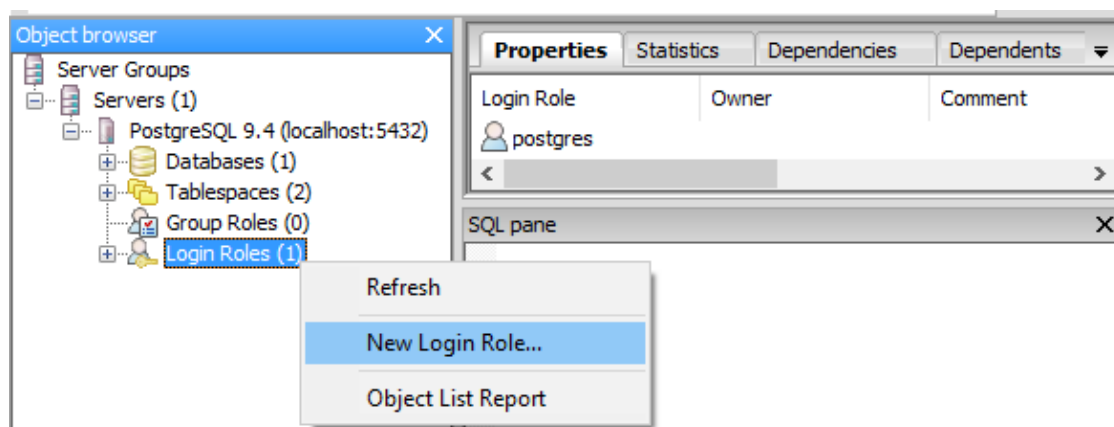
If you've used Git before, you might prefer to clone [our GitHub repository](#) instead. Don't forget to check out the v2.3.3 tag or the master branch.

Even better, you might like to fork the repository first, to give yourself a little more freedom to make code changes on the fly (and potentially [contribute](#) them to the project).

7.4 3. Set up a new database

Consejo: You can skip steps 2 and 3 if this is not your first installation. Every Tabbycat installation requires its own database, but they can use the same login role if you like.

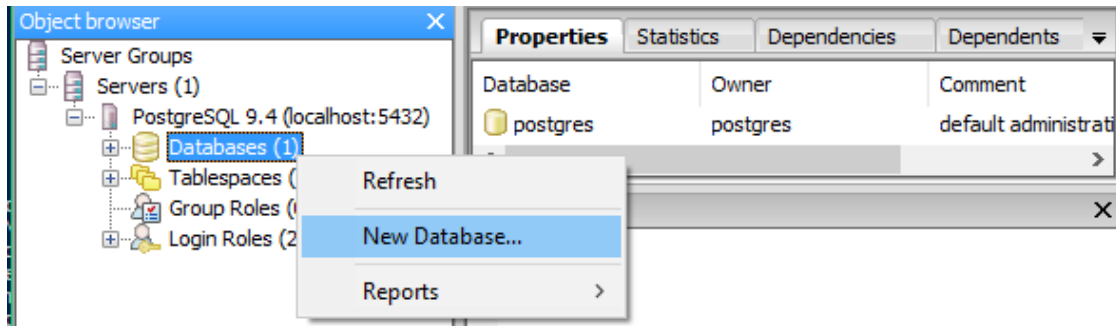
- Open the **pgAdmin** tool, which you installed as part of installing PostgreSQL. In the object browser on the left, double-click the server marked «(localhost:5432)». Log in using the password you set during installation.
- Right-click Login Roles, and click «New Login Role...»



- Fill in the New Login Role box as follows (everything not listed below can be left as-is):
 - In the **Properties** tab, in **Role Name**, choose a user account name. (If you don't know what to pick, we suggest «tabbycat».)
 - In the **Definition** tab, choose a **Password** and type it in **Password (again)**.

Then click OK. (Remember this user name and password, you'll need it later.)

- d. Right-click Databases, and click «New Database...»



- e. Fill in the New Database box as follows (everything not listed below can be left as-is):

- In the **Properties** tab, in **Name**, choose a database name (with no spaces in it).
- In the **Properties** tab, in **Owner**, type the name of the login role you just created.

Then click OK. (Remember the database name, you'll need it later.)

7.5 4. Install Tabbycat

Almost there!

- a. Open a Windows PowerShell. Navigate to the folder where you cloned/extracted Tabbycat. For example, if you installed it in `C:\Users\myusername\Documents\GitHub\tabbycat`, then run:

```
> Set-Location C:\Users\myusername\Documents\GitHub\tabbycat
```

- b. Make a copy of `settings\local.example` and rename it to `settings\local.py`. Open your new `local.py` file. Find this part, and fill in the blanks (the empty quotation marks) as indicated:

```
DATABASES = {
    'default': {
        'ENGINE' : 'django.db.backends.postgresql',
        'NAME'    : '', # put your PostgreSQL database's name in here
        'USER'    : '', # put your PostgreSQL login role's user name in here
        'PASSWORD': '', # put your PostgreSQL login role's password in here
        'HOST'    : 'localhost',
        'PORT'    : '5432',
    }
}
```

Optionally, replace the value in this line in the same file with your own time zone, as defined in the [IANA time zone database](#) (e.g., `Pacific/Auckland`, `America/Mexico_City`, `Asia/Kuala_Lumpur`):

```
TIME_ZONE = 'Australia/Melbourne'
```

- c. Start a new virtual environment. We suggest the name `venv`, though it can be any name you like:

```
> python -m venv venv
```

- d. Run the `Activate.ps1` script. This puts you «into» the virtual environment:

```
> .\venv\Scripts\Activate.ps1
```

Atención: If you get an error message saying that the script isn't digitally signed, open a PowerShell with administrator privileges by right-clicking PowerShell in the Start menu and clicking «Run as administrator». Then run this command:

```
> Set-ExecutionPolicy RemoteSigned
```

Read the warning message, then type `y` to confirm. By default, the execution policy on Windows is `Restricted`, which does not permit scripts like `activate` to be run. Changing it to `RemoteSigned` relaxes it to allow local scripts to be run without checking the signature.

- e. Install Tabbycat's requirements.

If you installed **32-bit Python**:

```
> python -m pip install --upgrade pip
> pip install -r .\config\requirements_core.txt
> npm install
```

Consejo: You might be wondering: I thought I already installed the requirements. Why am I installing more? And the answer is: Before, you were installing the requirements to create a Python virtual environment for Tabbycat to live in. Now, you're *in* the virtual environment, and you're installing everything required for *Tabbycat* to operate.

- f. Initialize the database and create a user account for yourself:

```
> cd tabbycat
> dj migrate
> npm run windows-build
> dj collectstatic
> dj createsuperuser
```

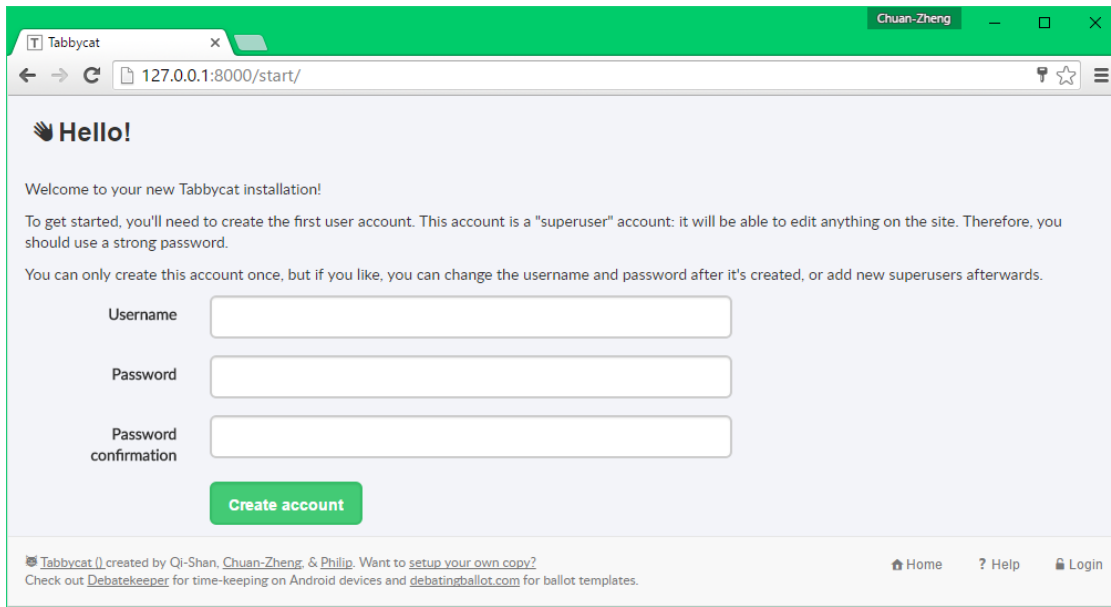
- g. Start Tabbycat!

```
> dj runserver
```

It should show something like this:

```
Starting development server on http://127.0.0.1:8000
```

- h. Open your browser and go to the URL printed above. (In the above example, it's <http://127.0.0.1:8000/>). It should look something like this:



If it does, great! You've successfully installed Tabbycat.

Naturally, your database is currently empty, so proceed to *importing initial data*.

7.6 Starting up an existing Tabbycat instance

To start your Tabbycat instance up again next time you use your computer, open a PowerShell and:

```
> Set-Location C:\Users\myusername\Documents\GitHub\tabbycat # or wherever your
↪ installation is
> .\env\Scripts\activate
> cd tabbycat
> dj runserver
```

Importing Initial Data

Once you've got Tabbycat installed, the next step is to import data for the tournament: that is, import details of teams, speakers, adjudicators and rounds. There are a few ways to do this, each with their advantages and disadvantages.

To help you decide which to choose, here's a summary:

Method	Best for	Drawcard	Drawback
Demonstration data	Trying out Tabbycat	Loads sample data in one click	Not for use with real tournaments
Simple importer	Small and medium-sized tournaments	Easy to use	Only deals with basic data
Edit database	Adding data not handled by the simple importer or editing existing data	Can handle all types of information	Adding large amounts of data is time consuming
importtournament command	Large tournaments	Easier to repeat, can handle most formats of information	Requires basic knowledge of how to use a command line interface
Developing your own importer	Large tournaments with custom needs	Easier to repeat, will take information in whatever format it is already in	Requires background in Python and learning about the importer classes

8.1 Demonstration data


If you're just learning or experimenting with Tabbycat, there are two demonstration datasets available, each with a sample set of teams, adjudicators, *etc.*, so that you can immediately start running rounds. Just be aware that these probably won't relate to anyone at your real-life tournament.

To load a demonstration dataset, click **New Tournament** link on the home page (once logged in as admin). You'll see a page titled «Create New Tournament». Scroll to the bottom of this page and click on one of the links at the bottom.

8.2 Simple importer

The simple importer is the easiest way to get a tournament going, and we recommend it for small- and medium-sized tournaments. It allows you to add institutions, teams, adjudicators, venues, venue categories and venue constraints. (If you need to add anything else, use the [Edit Database area](#) instead.)

To get started, create a new tournament using the **New Tournament** link on the home page (once logged in as admin). It'll ask you for a few basic pieces of information.

 **Create New Tournament**

Tabbycat is free to use for non-profit and non-fundraising tournaments (although donations are encouraged). If your tournament is run for profit or for fundraising, please note that there is a [required payment](#). For more details, see the [Tabbycat licence agreement](#).

Name

The full name used on the homepage, e.g. "Australasian Intervarsity Debating Championships 2016"


Short name

The name used in the menu, e.g. "Australis 2016"

Then, once you're in your tournament, click **Setup** in the left-hand menu, then **Import Data**, to open the simple importer.


DEMO


- Overview
- Feedback
- Standings
- Setup**
- Configuration
- Import Data**
- Private URLs
- Edit Database

 **Simple Importer**

There are several ways to import data into Tabbycat. Which one is best depends on the size of your tournament and your technical background. This **simple importer** is the easiest to use and works well for small- and medium-sized tournaments.

If you are having problems with this importer, you can use the [older version](#).

 [Add Institutions](#)

 [Add Teams](#)

You first need to add institutions. Once institutions are added, you can then add teams and adjudicators in the relevant sections. Each of these is a two-step process:

- For **institutions** and **venues**, it will first ask you to copy-paste a list of names and properties in a comma-separated table format. The second step is to confirm individual fields.
- For **teams** and **adjudicators**, it will first ask you how many teams/adjudicators to add for each institution (or who lack an institutional affiliation). The second step is to fill in their details, for example, names.

DEMO

- Overview
- Feedback
- Standings
- Setup
- Configuration
- Import Data
- Private URLs
- Edit Database
- R1

Add Institutions (Step 1 of 2)

Paste a list of institutions with one institution per line and following the format of: *Full Name,Nickname*. For example:

Victoria University of Wellington,VIC
University of Melbourne,MUDS
Universiti Teknologi MARA,UTMARA

Note that teams that use an institutional prefix will use the Nickname for this purpose, ie "MUDS 1".

Royal Melbourne Institute of Technology,RMIT
Stanford University,Stanford

DEMO

- Overview
- Feedback
- Standings
- Setup
- Configuration
- Import Data
- Private URLs

Add Teams (Step 1 of 2)

Specify the number of new teams to add per institution. In the next step you can specify team and speaker names.

1 California Institute of Technology (Caltech)

2 Columbia University (Columbia)

DEMO

- Overview
- Feedback
- Standings
- Setup
- Configuration
- Import Data
- Private URLs
- Edit Database
- R1
- R2

Add Teams (Step 2 of 2)

[< Previous Step](#)

Teams for California Institute of Technology

Name (excluding institution name)

Prefix team name with institution name? ☒

Speakers

Finally, if you would like to use venue categories and/or *venue constraints*, you can do so using the two last sections of the simple importer.

Nota: If copying and pasting from a spreadsheet, an easy way to make a comma-separated table is to save a spreadsheet with the relevant information as a *.csv file, then open this file in a plain text editor (such as Notepad or TextEdit), and copying it from there.

8.3 Editing the database

Sometimes, the simple importer just isn't enough—whether because you need more customization than the simple importer handles (*e.g.* adjudicator feedback questions), or because some participants changed their details after you

imported the initial data. In this case, the easiest thing to do is to edit the database via the Django administrative interface (under Setup > Edit Database).

The general pattern goes like this: Go to **Setup > Edit Database**, find the type of object you wish to add/change, and click «Add» or «Change». Then, fill in what you need to and save the object.

Prudencia: The Edit Database area is very powerful, and naturally if you mess things up, you can insert potentially catastrophic inconsistencies into the database. For participant information this is hard to do, but it's worth keeping in mind.

8.4 The `importtournament` command on local installations

We've written a management command called `importtournament` to help automate the tournament set-up. The script, however, is neither foolproof nor comprehensive, so you might find you need to modify things slightly if your tournament structure is different from ours. Be prepared to try this a few times to get it to work. Nonetheless, this is probably the fastest way to set up a tournament.

1. Copy and rename the `data/demo` folder
2. See the CSV files in the new folder, and add/replace the data as per your tournament. Note that the institutions (*i.e.* first column) in the `speakers.csv` and `adjudicators.csv` files must match the institutions in the second column of the `institutions.csv` file. And that all CSV files must end with a blank line.
3. Use this command, replacing `YOUR_DATA_DIR` with your new folder's name. (Square brackets indicate optional arguments; if you use them, omit the square brackets. All of them relate to the name of your tournament.)

```
$ ./manage.py importtournament YOUR_DATA_DIR [--slug SLUG] [--short-name SHORT_NAME]
↳ [--name FULL_NAME]
```

This script has a number of options. They're worth taking a look at before you run the script. For details, see:

```
$ ./manage.py importtournament --help
```

4. Assuming the command completes successfully without errors, you should double check the data in the Django interface, as described above in [Editing the database](#). In particular you should check that the *Rounds* have the correct draw types and that silent rounds have been marked correctly.

8.5 `importtournament` on Heroku installs

Instructions for using the `importtournament` command on Heroku installations are in steps 4 and 5 of [Installing on Heroku](#). The recommended procedure is first to import the tournament into a local installation, *as described above*, and then to push the local database to Heroku using the `heroku pg:push` command.

8.6 Developing your own importer

If our suggested file formats cause you headaches, it might be easier to write your own importer. We have a generic importer framework that should make this easier, so for some tournaments it might (very conceivably) be faster to write your own importer to conform to your data, than it is to make your data conform to our importer. You need a background in Python in order to do this. For more details, see [Tournament Data Importers](#).

Starting a Tournament

This page outlines a few things you should do at the start of a tournament, after you've *imported the initial data*. Once you've done these, proceed to *running a tournament*.

Prudencia: Tabbycat is developed for — and tested on — modern web browsers. If you are using **Internet Explorer versions 8, 7, or 6** the interface may look odd or not function properly. Switch to a newer browser if possible.

9.1 Tournament configuration

After importing all your data you can log into the site as an administrator by loading up the homepage and then using the **Login** button in the lower-right. From there you should go to the administration section of your tournament, and then go to the tournament configuration page by clicking **Setup** then **Configuration** in the menu.

Here you can adjust the debate rules and interface options to your liking then hit **Save** when finished. We also offer a number of presets that apply particular rule sets (such as the Australs rules) or feature sets (such as displaying information normally released during briefs on the website).

9.2 Special data types and options

There are a few optional fields that are not covered in the initial data templates, in the visual importer, or that may only be relevant in particular scenarios. It's worth going over these quickly to see if they are needed for your tournament. You can view and edit these fields in the **Edit Database** area (link is in the menu under **Setup**).

Adjudicator Feedback > Adj Feedback Questions

- As described in *Adjudicator Feedback*, the types of questions that can be posed for adjudicator feedback are able to be heavily customised. If you are customising your feedback form it should be done here, and before the tournament starts.

Authentication and Authorisation > Users

- Here you can add new admin users (those with full access) as well as new assistant users those (who can only do common data-entry tasks but not edit or view the full tab interface). See [User Accounts](#) for information on how to do this.

Nota: The people you're adding accounts for should be physically present when you do this, so that they can enter their password.

Participants > Regions

- Optionally, each institution may belong to a *Region*. An institution's region is used within the adjudicator allocation process to visually identify teams and adjudicators for the purposes of highlighting diversity issues. These have traditionally been used for geographic regions (such as Oceania), although could be repurposed as arbitrary markers of information — for example they could be used to denote teams from a particular State, institutional size, or circuit.

Participants > Adjudicators

- An adjudicator's *Test Score* represents their relative ability to judge important rooms, where adjudicators with higher numbers will, relative to the other adjudicators, be placed in better roles (ie as Chairs) and in the rooms you deem most important in each round. If you are running a small tournament, and plan to do your allocations manually, you can set everyone's number to the same amount.
- For larger tournaments, particularly those that collect feedback, see the [Adjudicator Feedback](#) section for more information on how test scores and other variables influence the automated allocation process.
- Regardless of how you score the ads, if you have changed the minimum chairing score in settings, you'll want to make sure there are enough adjudicators that meet this minimum threshold or the automated allocator may not function effectively.
- All types of conflicts are assigned to the relevant adjudicator. Adjudicator's can be conflicted against particular teams, particular institutions, and other adjudicators. Each of these is located in a tab at the top of the page.
- Each adjudicator's gender is optional and is not displayed publicly; it is only shown in the adjudicator allocation interface
- Each adjudicator's pronoun is optional, and is only displayed if you use tabbycat to print the ballots and feedback sheets for each round.

Participants > Teams

- Note the distinction here between full name and short name. The latter is used on pages where space is tight, such as the draw displays or the adjudicator allocation interface.
- Note that «Uses institutional prefix» option. With this option on, a team from the "MUDS" institution named "1" or "Gold" would be displayed as "MUDS 1" or "MUDS Gold".
- At present, setting a team's type to Bye, Swing, or Composite only affects very particular circumstances, and should be considered unnecessary.
- If you do have composite teams, and wish to have them be conflicted by adjudicators from each respective institution, you'll need to add a new team conflict to each adjudicator from each institution.
- If you do have swing teams, or teams that are otherwise ineligible for breaking, this is typically handled through the breaks interface in the main site

Participants > Speakers

- Each speaker's gender is optional and is not displayed publicly; it is only shown in the adjudicator allocation interface
- Each speaker's pronoun is optional, and is only displayed if you use tabbycat to print the ballots and feedback sheets for each round.

Tournaments > Divisions

- At the moment divisions are only useful for running tournaments that use round-robin style draws. Here, each division represents a draw pool within a round. Division support here is under development and not tested in many scenarios.

Tournaments > Tournaments

- Note that tournaments can have a welcome message (useful for displaying maps and other information on the homepage).

Venues > Venues

- A venue's priority determines its priority in being allocated. If there are 20 debates, and 30 rooms, the 20 rooms with the highest priorities will be chosen. Furthermore, if particular debates are marked as important during the draw process, those debates will receive the rooms with the highest priorities. In this way you can give close rooms to members of the adj core, or give larger rooms to debates that will draw a large audience.

Venues > Venue Categories

- Venue categories are not needed for most kinds of tournaments. Their purpose is to classify particular venues, such as venues all within one building or venues that are accessible. Once assigned these categories can display in the venue's name — ie «Red 01.01» or be used to assign Venue Constraints that match particular teams, institutions, or adjudicators to particular types of venues.

9.3 Information for the briefing

If you're using the online submissions feature, some things you should probably mention in the briefing:

- Adjudicators must fill out ballots completely, including motions and venues—they are entered into the system.
- There is a static URL for each person's ballots and feedback forms. It can be bookmarked, or the page can be refreshed after each round.
- If people submit a result or feedback online, they should indicate that they have done so on the paper copy of their ballot.

Running a Tournament

Once you've finished the steps in *Starting a Tournament*, you're ready to go! This page outlines what you would do for each round during the tournament. After the tournament, proceed to *Finishing a Tournament*.

This is all done from the admin area (*i.e.*, by the tab director or adjudication core member). In the admin area, tournament-wide pages (feedback, standings, and break) are at the top of the left-hand menu, while round-specific pages (availability, draw, display, motions, and results) are in dropdown's organised by each round's abbreviation.

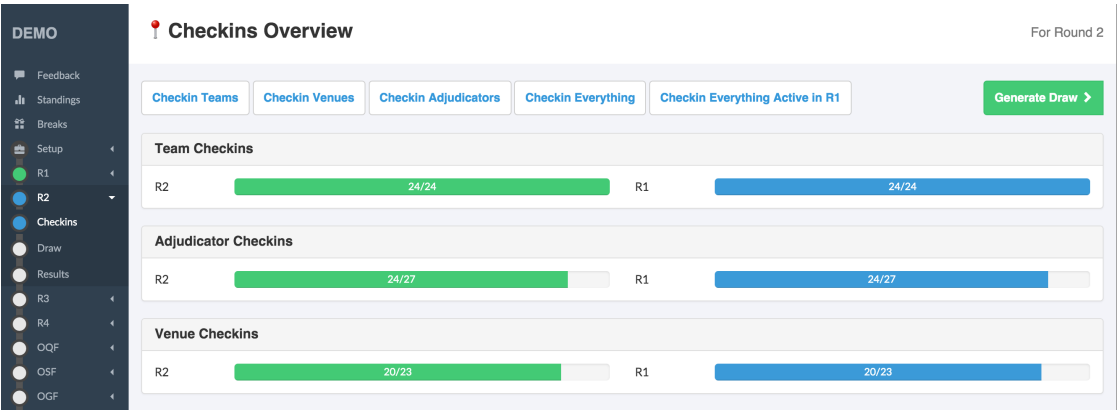
The basic workflow for each round is:

1. *Mark the teams, adjudicators, and venues present as available*
2. *Generate the draw* and allocate the adjudicators
3. *Show/release the draw*
4. *Release/enter* the motions
5. Have the debates
6. *Enter results*
7. *Advance to the next round*

10.1 Availability

Set availability. For each round, you need to set the venue, team and adjudicator availability. If any of those are not marked as available they will not be used within the draw; so this feature is mostly useful for when adjudicators or venues are only available for certain rounds.

To do this, click the round in the menu, then click **Check-Ins**. Here you can then go to the availability pages for venue, teams, and adjudicators, or check in everything at once. When you've set everything appropriately use the **Generate Draw** button in the top right to advance.



Nota: You can set availabilities in advance of the current round — ie if you know the venue/adjudicator schedules their availabilities can be set ahead of time.

10.2 Generating the draw

1. **Confirm the draft draw.** After advancing from availability section you will first be shown a draft draw that details how the draw was formulated, pointing out pull-ups and conflict swaps and the like.

Draw for Round 2 (DRAFT DRAW)

Find in Table

Confirm Draw

	AFFIRMATIVE	NEGATIVE	ASR	NSR	APTS	NPTS	AASS	NASS	AFFS	NAFFS	CONFLICTS/FLAGS
1.0	Johns Hopkins 1	Cornell 1	7	1	1	1	265.75	268.50	0	1	
1.0	Caltech 3	MIT 2	2	8	1	1	268.25	265.25	0	0	
1.0	Pennsylvania 2	MIT 1	9	3=	1	1	264.00	267.50	1	1	
1.0	Pennsylvania 1	Yale 2	10=	3=	1	1	263.50	267.50	0	1	
1.0	Harvard 2	Stanford 1	5	10=	1	1	266.75	263.50	0	0	
1.0	Berkeley 1	Columbia 1	12	6	1	1	261.50	266.00	1	1	
0.0	Chicago 2	Stanford 2	1=	7=	0	0	265.50	260.50	0	1	
0.0	Princeton 2	Stanford 3	7=	1=	0	0	260.50	265.50	0	0	
0.0	Caltech 2	Harvard 1	1=	9	0	0	265.50	259.75	0	1	
0.0	Chicago 3	Princeton 1	4	10=	0	0	264.50	259.50	1	1	
0.0	Caltech 1	Yale 1	10=	5	0	0	259.50	263.75	0	1	
0.0	Chicago 1	Yale 3	6	12	0	0	261.00	258.75	0	1	

Nota: The draft draw is for you to double-check. While there are some basic tests on the draw algorithm, it never hurts to sanity-check it again.

If you *do* find something wrong with a draft draw, you can edit the match-ups, but please also let us know what the problem was! You can find our contact details in the [Authors & Acknowledgements](#) section.

2. Once on the confirmed draw page you can click **Edit Adjudicators**.

DEMO

- Feedback
- Standings
- Breaks
- Setup
- R1
- R2**
- Checkins
- Draw
- Results
- R3
- R4
- OQF
- OSF
- OGF
- SSF
- SGF
- FGF
- NSF
- NGF

Draw for Round 2

draw is confirmed but not released; motions are not released

Edit Adjudicators
Edit Motions
Draw Details
Add Start Time
Release Motions
Release Draw
Enter Results >

One or more debates don't have a chair. [Edit adjudicators.](#)

Display Draw by Venue
Display Draw by Team
Print Ballots
Print Feedback Forms

		AFFIRMATIVE	NEGATIVE	ADJUDICATORS	CONFLICT
1.0	K05	Berkeley 1	Columbia 1		
1.0	K08	Caltech 3	MIT 2		
1.0	K04	Harvard 2	Stanford 1		
1.0	Z09	Johns Hopkins 1	Cornell 1		
1.0	Z10	Pennsylvania 1	Yale 2		
1.0	P07	Pennsylvania 2	MIT 1		
0.0	K03	Caltech 1	Yale 1		
0.0	P03	Caltech 2	Harvard 1		
0.0	P05	Chicago 1	Yale 3		
0.0	P01	Chicago 2	Stanford 2		
0.0	G01	Chicago 3	Princeton 1		
0.0	G02	Princeton 2	Stanford 3		

3. **Allocate the adjudicators.** Changes here will auto-save; feel free to return to the **Draw** when needed. See [adjudicator allocation](#) for more details about the allocation process.

< 11s ago Prioritise Allocate

Seen Institution Conflict Missing Unavailable

Break Gender Rank Region

6	9	Éothéod AK Upbourn GB	Caras Galadhon LD Arnor LJ	3.0 3.5	5.0 Yesenia A MITHLOND	2.0 Ivona P UG RINGLO 1 ago	2.0 Dennis V UG ANFALAS	1.0 Bella H SOUTHERN MIRK... 1 ag
6	7	Northern Mirkwood Rohan Military WA	Minas Tirith BV Mithlond SR	2.7 3.0	4.0 Hoan Đ SOUTHERN MIRK...	2.0 Eduardo P CARAS GALAD 1 ago	2.0 Keiran W IRON HILLS	1.0 Hrund H UPBOURN 1 ag
5	6	UG Ringló Vale RM Staddle JL	The Shire WL Lórien BD	2.6 3.0	4.0 Eleanor W ISENGARD 2 ago	2.0 Tim B EDU 2 ago	2.0 Aalivah R IRON HILLS 2 ago	3.0 Evelyn P RI ROHA 1 ago
5	5	Ithilien TN UG Ringló Vale PN	Dale TM Southern Mirkwood I	2.7 3.0	3.0 Tao H ITHILIEN	3.0 Olivia L RIVENDELL	2.0 Oliver M STADDLE	1.0 Carly C MITHLOND
5	4	Southern Mirkwood Dorwinion SS	RI Rohan MM Rivendell CL	3.5 5.0	5.0 Laila F LÓRIEN 1 ago	2.0 Davi C CARAS GALAD 2 ago	2.0 Dario M ITHILIE 1 ago	5.0 Sonny B PELAGOR
4	5	Iron Hills AD The Shire NE	Iron Hills LE Erebor BP	3.7 4.0	5.0 Jana F UG ANFALAS	3.0 Cintio A UG LOSSARNACH	3.0 Fareed S MITHLOND	
4	6	Isengard NS Lórien SB	Caras Galadhon BL Minas Tirith EN	3.4 4.3	5.0 Marvin L UG ANFALAS 1 ago	2.0 Abraham W EOTHEOD	3.0 Addison M DOL AMRO 2 ago	1.0 Chukwubuike I DORWINION 1 ag
4	5	Rivendell DD Rivendell BA	Bree MN Bree HC	3.3 4.0	5.0 Facino L ISENGARD	3.0 Camilla E ROHAN MILITARY	2.0 Anzor D THE SHIRE	
4	4	Edoras SW Minas Tirith CB	Mithlond NC Upbourn VS	3.4 3.7	4.0 Sofia R EOTHEOD	3.0 Diego B EREBOR	3.0 Kathrin F ARNOR	4.0 Uchechukwu U EREBOR

Nota: If you are using venue constraints the **Draw** page may prompt you to Auto Allocate the venues used to satisfy those constraints; see [venue-constraints](#) for more details. Regardless of whether you are using venue constraints or not you can change the Venues per-debate in the **Edit Venues** area.

10.3 Releasing the draw

Once you're happy with your adjudicator allocation, you're ready to start the round.

1. **Release to general assembly.** From the *Display* page for that round, go to **Show by Venue** or **Show by Team** (whichever you prefer). Then put it up on the projector. There are automatic scroll buttons and buttons for changing text sizing.

Draw for Round 1 debates start at 2 p.m. ESC stops scroll

Scroll
Fast
Medium
Slow
XS
XXS
Text
XS
S
M
L
XL
Find in Table
Q

	AFF	NEG	ADJUDICATORS
G02	MIT 1	Caltech 2	Mandy Estrada
K03	Yale 2	Caltech 1	Toby Wong
K04	Columbia 1	Stanford 3	Eunice McBride
K05	Cornell 1	Chicago 2	Levi Flores
K06	Princeton 1	Stanford 1	Holly Howard
K08	Berkeley 1	Chicago 1	Boyd Pierce
P01	Chicago 3	Caltech 3	Jacqueline Freeman © Tasha Jenkins Pearl Graves

2. **Release to public.** If you're using the public draw function (where the draw is posted publicly to your Tabbycat website) use the **Release to Public** button to allow the page to display.

Truco:

- To avoid the site from being overloaded by anxious refreshers, we recommend that large tournaments not release the draw to the public until after it's been seen by general assembly. That said, note that due to caching there can be a delay of up to 1 minute between when a draw is released and when it first shows up on the public site.
- Tabbycat can be set to send emails once the draw is released to adjudicators informing them of their assignments for convenience.

10.4 Entering and Releasing Motions

Tabbycat is agnostic as to whether you enter motions into Tabbycat before or after they are shown publicly. However, they must be entered *at some point* before ballots are entered.

1. **Enter the motion text.** Either before or after their public release motions can be entered in the **Motions** section for that round.
2. **Release to general assembly.** If you are entering motions *before* they are publicly revealed note that there is a *Display Motions* button in the **Display** area that allows you to do a Power Point style motion release.
3. **Release to public.** As with draws, if you have the *enable public view of motions* setting configured your Tabbycat website will display a running list of motions from the tournament. When this is on, using the **Release Motions to Public** button on the **Motions** page will mark the current set of motions as able to be displayed on this page.

10.5 Entering Results

1. Enter debate results and feedback as they come in (and/or allow online entry of results and feedback).
2. Both results and feedback entered in the tab room or online need to be confirmed before the results are counted. To confirm a debate ballot and the debate as a whole, the confirmed checkbox under *Ballot Status* should be ticked in addition to the *Debate Status* being set to Confirmed.
3. Note that you can track data entry progress from the **Overview** page for the tournament.

See [Entering Ballots and Feedback](#) for more details about the data entry process.

Advertencia: We strongly recommend entering all data using the assistant area, not the admin area. This is because the admin area (intentionally) does not enforce the data confirmation procedure.

10.6 Advancing to the next round

Once you've got all the results entered and confirmed, you're ready to progress to the next round. This can be done by going to the **Results** area, and then using the **Advance to Next Round** button.

DEMO

- Feedback
- Standings
- Breaks
- Setup
- R1
- R2
- Checkins
- Draw
- Results
- R3
- R4
- OQF
- OSF
- OGF
- SSF
- SGF
- FGF
- NSF
- NGF

Results

for Round 2

[Ballot Checkin](#)

Advance to Next Round >

0 Checked In
12 Unknown
0 Unconfirmed
0 Confirmed

BALLOTS		AFFIRMATIVE	NEGATIVE	ADJUDICATORS
✗ Enter	1.0 K04	Harvard 2	Stanford 1	Justin Ford
✗ Enter	1.0 K05	Berkeley 1	Columbia 1	Tasha Jenkins
✗ Enter	1.0 K08	Caltech 3	MIT 2	Holly Howard
✗ Enter	1.0 P07	Pennsylvania 2	MIT 1	Jamie Hodges
✗ Enter	1.0 Z09	Johns Hopkins 1	Cornell 1	Bennie Rodriguez
✗ Enter	1.0 Z10	Pennsylvania 1	Yale 2	Leigh McGee
✗ Enter	0.0 G01	Chicago 3	Princeton 1	Levi Flores ©, Boyd Pierce, Luis Bennett
✗ Enter	0.0 G02	Princeton 2	Stanford 3	Allison Martin
✗ Enter	0.0 K03	Caltech 1	Yale 1	Pedro Burgess ©, Jacqueline Freeman, Jo Allen
✗ Enter	0.0 P01	Chicago 2	Stanford 2	Toby Wong
✗ Enter	0.0 P03	Caltech 2	Harvard 1	Cassandra Wilson ©, George Willis, Kristi Elliott
✗ Enter	0.0 P05	Chicago 1	Yale 3	Pearl Graves ©, Ramona Cobb, Mandy Estrada

Advertencia: When you advance to the next round, if you've enabled public results, the results for the current round (which is now the previous round) will be released to the public **unless** the round is marked as «silent» in the database. So if you're careful about when results should be released, don't change the current round until you're ready to release those results.

Nota: There is a design assumption that you will always want to release results for non-silent rounds before you start working on the draw for the next round. If this isn't true for you, please get in touch with us so that we know. The

workaround is to make all rounds silent, then unsilent them when you're ready to release results.

Truco: Emails can be sent to speakers informing them of their team's win/loss/points record before advancing rounds. This is independent from whether the whole round's results are released to the public.

Finishing a Tournament

This page outlines some final steps to take after the conclusion of outrounds.

11.1 Tab Release

Tabs can be released using the *Tab released* option under **Setup > Configuration**. Tabbycat offers the following system tabs:

- Team Tab
- Speakers Tab
- Replies Tab
- Motions Tab

You can configure the team, speakers and replies tab to display only a certain number of speakers, *e.g.*, to show only a «Top 10 Speakers» tab.

If you defined any speaker categories (for example, Novice, ESL or EFL), a tab for each category marked «public» can also be released using the **Release speaker category tabs to public**. You can similarly limit each of these tabs to display just the top few speakers, in the definition of the speaker categories. The speaker categories not marked public are *not* released by this option.

You can also redact individual speaker's identifying details (name, team, and institution) from the public individual tabs. You can do so by going into the **Edit Database** area, going to *Participants > Speakers*, finding the speaker and clicking the **Anonymous** box (and saving).

Nota: Public tab pages are cached for performance reasons. This means that any changes that affect a tab page (say redacting a speaker or changing a speaker score) may not show up on the public site for up to an hour.

11.2 Wrapping Up

You probably want to turn off the *Public ballots*, *Public feedback*, *Feedback progress*, and *Public draw* features under **Configuration** at this stage as they no longer have any use.

You also want to go into the **Edit Database** area, find the tournament and hit «release all» so that the results for the final round can be released.

Tournament Logistics

Unlike the rest of our documentation, this section does not deal with particular features or technical concerns with Tabbycat itself. Instead it is an attempt to outline the logistics of tab direction and aims to be of general relevance for people running major tournaments. At present, it is organised by the various ‘stages’ of tabbing a tournament, and most of the content takes the form of check-lists or comments designed to highlight and provide guidance on common issues.

Whilst it aims for general relevance, we should note that this guide is relatively opinionated and mostly written from the perspective of people whose primary experience is tabbing at Australasian tournaments using Tabbycat. That said, we welcome feedback and additions that can account for different format and regional considerations. In the future, if the guide becomes more general and more extensive, it could be spun off into a separate project.

Nota: As with the rest of our documentation, this page is open-source and we welcome *feedback and contributions*. Note also that we’ve formatted this guide a single page to reduce clutter, but the sub-sections in the sidebar can be used to help navigate across sections.

12.1 Planning and Preparation

This section aims to outline concerns that occur in the months before the tournament: after you have agreed to help with tabbing and while the organising committee and adjudication core are deciding how they want to run key processes such as registration and feedback. It is organised in terms of who you should coordinate with in order to plan for a well-tabbed tournament.

12.1.1 General Notes

You should avoid being the sole person responsible for that tab unless it is a small tournament. There are many cases where you want to be in several places at once and the easiest way to accommodate that is by having co-directors or trusted assistants. Few tab decisions require a single source of authority; it is far better to have multiple people available to share responsibilities around.

In a similar manner, it is worth considering how you can use the tournament to help train other people. Typically, knowledge of tabbing is concentrated in relatively few people and gained mostly through on-the-ground experience; meaning that every tournament should be approached as rare opportunity to help spread knowledge about tabbing more widely in a circuit. Consider reaching out to institutions or the tournament as a whole to see if they have anyone who would be interested in helping out.

12.1.2 Convenors

It might sound obvious but it will pay to have a very thorough conversation about the tab process (more or less the contents of this document) with the convenors a few months out from the tournament. Do this even if you know the convenors to be knowledgeable or experienced debaters. Key concerns are:

- Whether internet access will be available and whether participants can be presumed to have smart phones. This has an obvious impact on how online feedback, ballots, and draw release is done. Note that Eduroam is not necessarily a reliable guarantee of access; many participants will come from universities who don't have access to it or will need to follow a setup process that is onerous or requires them to be at their home institution.
- What kind of room is the tab room going to be? Is it possible to optimize its placement when the bookings for rooms are made? Key details include: How large is it? Does it have a sufficient amount of desk space (for data entry)? Does it have a good projector (for allocations)?
- If they have the resources, having an adjacent room available for just the adjudication core to use can also be useful. While you want to work closely with the adjudication core, they may want to discuss sensitive information (motions, equity) in a space without volunteers present; or they might at times get in the way of things, such as by eating lunch in the middle of a frenetic ballot entry process.
- Ensure that plans are made for food to be brought to tab room. Otherwise you will starve and the adjudication core will swan off to lunch. Having regular access to caffeine can also be similarly essential to some adjudication and tab teams.
- What kind of printers will be available? Can the tournament buy/borrow one? This is obviously a key consideration for pre-printed ballots. Also try and ensure there are back-up printing options if possible. Clearly stipulate your need for ink and paper; and try and opt for a black/white laserjet, over an inkjet, if possible.
- What kind of volunteers will be available? How many, and what is their experience level? As a very broad recommendation, you probably want around 1 volunteer for every 10 rooms, assuming volunteers are performing a dual role as data-enterers and ballot-collectors.
- Will the tournament make a donation to whoever maintains the tabbing software you are using? Depending on the license of your tabbing software and the nature of your tournament (for profit vs not for profit) this may be required. Also, if your tab is self-hosted or independently hosted (such as how Tabbycat is generally deployed on Heroku) accounting officers should also be aware that there will be some costs associated with hosting the tab.
- You should also ensure that people helping with the tab are fairly compensated for their flights, registration, etc and that any volunteers are invited along to socials and/or given some other recompense.
- Will Swing teams be available? You should plan to have at least one more than you need. For example, with 39 teams, you should have both an 40th swing team to fill in the draw, and the option to easily assemble an 41st swing team in case a team goes missing. At very large tournaments (say over 150 teams) you should plan for even more swing team capacity — it's not unheard of for say three teams to vanish all in a single round. In these cases, you should try and ensure that the swing teams are always ready to go — i.e. that they are pre-formed, you have a clear communication channel with them, and that they distributed/waiting near the debating rooms so they can fill in at a moment's notice (often you will only find out that teams are missing right as debates are scheduled to start).
- How will critical information be communicated to participants? Consider that in general, Facebook announcements do not reach many people, although paying to boost the posts is often a very cheap way of dramatically

raising their effectiveness. In particular also ensure or check how you manage to get in touch with teams or adjudicators who go missing: will they have reliable phone numbers? Can you get a list of institutional reps who can be reliably called? You want to have processes in place for chasing up adjudicators who do things such as make scoring mistakes as soon as possible in order to minimise delays.

- How will critical information be shared between the tab team, adjudication core, and logistics/convening teams? For smaller/medium sized tournaments a group chat augmented by phone calls (assuming everyone knows everyone else's number) can be sufficient, but even then, you need to ensure that any critical information conveyed privately (i.e. in a call or in person) is conveyed back to the group channel. At very large tournaments (or if you have the resources) walkie-talkies are an excellent way to manage communication — just make sure you have (ahead of time) reserve the different channels to a distinct and known purpose (i.e. general discussion; just the tab team & adjudication core; just convenors).
- As part of this it is ideal if the organising committees can procure local SIM cards for members of the tab team and adjudication core who are not local. These should be relatively generous in their plans — you don't want to worry about running out of minutes or data if on a critical call or using a hotspot to make critical allocation adjustments.
- At major tournaments you want to arrive at least a day before check-in; and ideally whenever it is that the adjudication core is arriving for their own preparation.

12.1.3 Registration

Having effective registration systems and processes is one of the most important aspects of preparing to tab a large tournament. Bad registration data *will* make setting up a tab extremely painful and introduces the chance for mistakes or inconsistencies in tab data that will only come to light in the first round. As such:

- You should check in with the registration team and see what they plan to do as soon as possible after being brought on-board. As part of this you should make it clear that you should be consulted on any decisions they make about what data to collect, when to collect it, and how to collect it.
- Registration data should be collected into a shared and live-updating source, such as a Google Sheet. There should be as few canonical sources (ideally one) of data as possible; i.e. there should be a single sheet for individual details, a single sheet for team details, etc; and these should be maintained all the way through to check-in. For both you, and the registration team, having multiple conflicting or outdated copies of data will lead to errors. However, for the registration team these errors can usually be easily sorted out in person (at check-in) but for you that information always needs to be reliable and up to date otherwise what is imported into the tab cannot be trusted.
 - At this point our recommendation is to, in most cases, not use specialised registration systems as they are somewhat less intuitive and less flexible than setting up good Google Forms/Sheets.
 - If, for whatever reason, the registration team are not able to give you “live” access to the data they have on hand, make sure they send you copies of it (even if it is incomplete) well before you need it to setup the tab itself. You want to be able to verify what data is actually being collected and how it is formatted well in advance.
- You should have access to *all* of the data collected; often registration teams will make (false) assumptions about what you do or do not need. It is better to have everything and then selectively filter out what is not relevant to the tab.
- It is critical that the registration team should check in with you before setting up forms asking for information. Every additional time that registration asks for data there will be less and less participation in the process, so you should aim to gather all that you need at the first opportunity; typically during the canonical individual registration phase. Particular information that should not be overlooked for tab purposes:
 - Individual registration should ask whether a participant is a speaker or an adjudicator.
 - If that person is a speaker it should ask for their team name/number (reconciling these later is painful).

- Individual registration should ask for any accessibility requirements of both adjudicators and speakers.
 - Individual registration should ask for the previous institutions of both adjudicators and speakers.
 - Individual registration should ask for the email addresses of all participants.
 - Individual registration should ask for the phone numbers of adjudicators.
 - Individual registration should ask for the gender identity of both adjudicators and speakers. Even if you are not *planning* on using this to inform processes, such as adjudicator allocations, you want it on hand in case plans change.
- Independent adjudicators and the adjudication core should follow normal registration procedures. Having them not go through the normal process makes it easy to overlook their data or not get a complete picture of it. For example, adjudication core members might forget to nominate conflicts, or neglect to provide their previous institutions.
 - You should confirm how the registration team plans to manage how people check-in to the accommodation in particular. Check-in is when issues with registration data come to light and it is vital that these changes are noted and recorded. Some form of validation of registration data *must* occur at check-in — in particular all adjudicators should be (individually) verified as present and all members of a team should confirm their presence along with their team's name/number and speakers.
 - After check-in you need to have a definitive list of who is physically present at the tournament so you can run a first-round draw with confidence. Registration must know this and have processes in place for recording people individually as they arrive, and for that data to filter back to you.

Nota: If you are using Tabbycat's secret links for feedback or ballots these are best distributed at check-in. The registration team should know about this, prepare for it, and be provided with the pdfs to print and distribute.

12.1.4 Adjudication cores

If there is a group chat for the adjudication core you probably want to be part of it; even if you don't contribute much. There are lots of small things that end up being discussed without consideration of how they will affect tab issues and it is also a chance to get to know — ahead of time — the people you will be working with closely over the tournament.

Members of the adjudication core will often leave tab-relevant decisions until the days prior to the first round or whenever it is that they can first meet with the tab team in person. This often wastes critical time and forces rushed decisions. Many considerations can instead be raised and discussed prior to the tournament. These could include:

- How to manage the feedback process. This typically benefits from foresight and pre-planning, rather than being decided on the ground. Key considerations are:
 - Who submits feedback on whom? Do trainees do so on their chairs? Do panellists do so on each other? (Presuming your tab software supports these options).
 - Is feedback mandatory? If so, how will this be enforced exactly?
 - How much weight does each adjudicator's test or CV score have over the course of the tournament? By Round 3, or by Round 8, what proportion of an adjudicator's score is derived from their test and what proportion is derived from their feedback?
 - Will the adjudication core tweak an adjudicator's score to "artificially" increase or decrease it to where they think it should be. For example, this could be done by adjusting a test/CV score upwards in order to compensate for bad feedback that (for whatever reason) they did not think was reliable or fair? Depending on your adjudication core's preferences and your tab software's allowances it is not unheard of for them to maintain full manual control over scores by reading/processing feedback results but only ever manually adjusting scores as a result (rather than having it automatically adjust due to the ratings in the feedback).

- What is the score scale going to be? What do each of those numbers represent? How will this be communicated to participants so they can score accurately and consistently?
 - What kind of questions will feedback forms ask? If using *customisable printed or online forms* consider how these questions be used tactically to identify key issues (say discriminatory scoring) or more easily identify people who should be promoted/demoted. While managing feedback is often a messy and subjective task, it can often be improved by being more targeted in what data it collects.
 - How will feedback be monitored, and how will this information feed back into the scores and allocations? At large tournaments it is not unusual for an adjudication core member to sit off each round to review and process feedback — there isn't really a good stretch of available time to do so otherwise. However even if doing this note that there are communication issues to manage here, as each adjudication core member will each end up with a relatively incomplete overview of the total volume of feedback.
- If possible it's nice to plan in advance for when the tab will be released (i.e. on the last night; the day after; etc.) as this often gets left to the last minute to be decided. Also the possibility of whether people can redact themselves from tabs should be raised, as that might be useful to inform participants of during online registration or tournament briefings. In a similar fashion, some adjudication cores might also want to limit speaker tabs to only a certain number of places, particularly at novice-centric tournaments.
 - How to handle conflict collection; see the following section.
 - How to handle the submission of scoresheets and feedback, primarily in terms of which parts of the process should be done online and offline. Some adjudication cores will have strong thoughts here; others will happily follow whatever you recommend. Key considerations:
 - Paper-based feedback is much more taxing to enter than paper-based scoresheets — typically there is much more of it; it asks for a greater variety of data; and it is submitted at inconsistent times. The one advantage is that it is easier to make feedback mandatory with paper, as you can ensure all teams and adjudicators have done so prior to leaving the room. Thus, in most cases, a good online feedback system is much more preferable than paper. If using paper be aware that you will need a lot of volunteers to ensure the feedback is collected promptly. If internet or smartphone access is limited at your tournament it is probably best to accommodate both paper-based and online methods.
 - The consequences of having incorrect or missing ballots are much more severe than for feedback. As such major tournaments use paper ballots in some form as the final stage in a checking process to ensure that the results of a debate are definitely correct — adjudicators will always make mistakes and while digital ballots can catch/prevent some types of error (i.e. a low point win) they can't catch others (assigning the wrong scores to the wrong speaker, nominating the wrong winning team, etc.). Assuming your software supports both options, the choice is thus whether to use a hybrid approach (online submission followed by paper verification) or to rely entirely on paper. A fully-paper based approach will be simpler for both yourself and adjudicators, and can be almost as efficient if you have a sufficient number of volunteers. In contrast, a hybrid approach will be potentially much faster if you are short of volunteers and if you expect that almost all adjudicators will have access to the internet, a smartphone, and are capable of following instructions.

Nota: In some circuits, and when using some particular tab software, tournaments might run a “dual tab” where there is a second, independent, version of the tab software and database into which all data is *also* entered. From what we understand this performs a dual role, as both a backup system that can take over from the main one (say if internet access drops) and as a way of verifying ballot data (by comparing draws or databases between software rather than having a two-step entry process operating for a single tab). This practice seems obsolete when working with modern web-based tab software that is capable of backing up and restoring to an offline system, but we would like to hear your feedback if you think that is not the case.

12.1.5 Conflicts/Clashes (registration/equity/adjudication core)

- There should always be a *single* means of collecting conflicts (i.e. a single Google Sheet/Form) and all conflicts should go through it. Because the nature of this data is sensitive and evolving, there must be a single location where it can be easily captured and verified as having been entered into the tab. Conflicts data should never be spread across a loose collection of emails/personal messages/spreadsheets; otherwise keeping track and knowing which ones have been entered into the system will be painful and error prone. Get in touch in with equity and registration in advance and make it clear that they should not make their own conflicts form; or if they've already made one, make sure you adopt it and have access/control of it.
- Conflicts should, ideally, *only be collected after a participants list has been published* and requests for people to nominate conflicts should also be sent out as few times as possible. Most people will only fill this form in once, so it is vital that when asked to nominate conflicts they have as much information as they need to do so comprehensively. Without a public and reasonably-complete participants list people will either nominate conflicts that are not present (wasting your time in cross-referencing data) or not realise someone is present and raise the conflict at a latter, less opportune time.
- In some circuits only adjudicators are allowed to nominate conflicts because of the risk of teams using conflicts “tactically” to block adjudicators that they think are terrible judges. However, having teams nominate conflicts can be useful: adjudicators may overlook a conflict or there may be equity-based reasons that a conflict is non-symmetrical. This trade-off can be handled in two ways:
 - Not allow teams to nominate conflicts during registration; but allow them to approach equity teams before, or during, the tournament to identify the conflict. Equity can then raise the issue with the tab team and adjudication core and it can be added to the tab.
 - Allow teams to nominate conflicts during registration; but have the adjudication core review the data for “tactical” conflicts. These are usually relatively easily identified, although can be overlooked if the adjudication core does not know the participants or their region/circuit well. The adjudication core can then override the conflict, discuss it with the teams, or raise it with equity. However, if going down this route, the tab team should discuss with the adjudication core how to manage this process well-ahead of the tournament, and ensure they actually do review the conflicts prior to the first round — otherwise it will likely surface during an allocation and become a major distraction during a critical time period.
- As mentioned in the previous section, the adjudication core (possibly with equity) should provide some degree of guidance about what kinds of debating-related conflicts should be provided. People should be able to self-define what constitutes a conflict, but there are circumstances where they are overly cautious and can be reassured that it is not necessary. The opposite problem may occur also, where many people may have a very high bar for what defines a conflict which could lead to perceptions of bias from other participants.
- Generally, it is preferable that each form nominates a single conflict, and people are asked to re-submit for each conflict they are adding.
 - To save you some hassle the conflict form should make this very clear (i.e. that one conflict = one submission; ensure the field labels reinforce this)
 - The conflict form should also make clear that you shouldn't use the form if you don't have any conflicts (i.e. people will submit “None”, “None” etc)
 - The conflicts form should also make clear that adjudicator's don't need to submit a conflict for their current institution and that team's don't need to submit conflicts for adjudicators from their current institution.
- In poorly-structured conflict forms, identifying exactly who is doing the conflicting and who is being conflicted is a nightmare. You want to structure the questions to minimise this ambiguity. A form should definitely ask:
 - Who are you (the conflict-specifier)?
 - Are you a team or an adjudicator?
 - Which institution are you from?

- If part of a team, which team are you in?
 - Who are you conflicting?
 - Are they a team or an adjudicator?
 - Which institution are they from?
 - If they are in a team, which team is it?
 - Have previously attended any other institutions; or have other reasons to conflict entire institutions? If so, specify those institutions.
- Note that this last question can be tricky to deal with; good tab software will let you conflict an adjudicator from an institution other than their own, but it is harder to mark an individual team as having members previously attending another institution. These circumstances are rare and typically very “soft” conflicts but are probably best handled by creating individual conflicts between that team and adjudicators from the previous institution in question.
 - Adjudication core members will often not nominate their own conflicts; presuming that they will notice and correct them during allocations. They often forget or overlook this. Their conflicts should be entered as per normal.

12.1.6 Scheduling (convenors / venue organisers)

One of the easiest ways to have things run late is to set an unrealistic schedule. As much as possible the timing allocated to rounds (inclusive of events such as lunch or committee forums) should conform to an even distribution of how long it takes to process results and create a draw/allocation — you don’t want to be in a position where particular rounds have too much time and others too little time to spend on allocations and other crucial tasks. This is something that should definitely be working on in conjunction with convenors and other critical parties before they lock down timing details with food suppliers or the operators of the debating venues.

Note also that in most circumstances it is preferable to create a draw and allocation for the first day of the next round at the night before. This time should be built in to the schedule of the previous day, and raised with the adjudication core so they don’t expect to be able to immediately depart after the day’s rounds are done.

Below is the time taken within each round at Australs 2017. For context, this was neither a particular efficiently or inefficiently tabbed tournament. Notable details:

- The tournament was ~40 rooms each round and had access to 3-6 runners and data enterers. Paper ballots were pre-printed and distributed by runners to rooms prior to the debates starting, then collected sometime after the 15 minute deliberation period. Feedback was submitted online. At Australs all adjudicators (excluding trainees) submit their own ballots.
- The adjudication core were neither particular slow nor fast in allocating adjudicators compared to other adjudication cores. At Australs most adjudication cores will create allocations by using first running an automatic allocation then extensively tweak the results.
- There were no serious issues that delayed the tabbing of any particular round beyond the routine and expected issues of last-minute draw changes, adjudicators producing incomprehensible ballots, etc.
- Whilst the tab ran relatively quickly, there were minor delays because of mismatches between the planned schedule and the optimal schedule from a tab perspective.
- A round at Australs takes around 2 hours from a debater’s perspective: 30m of prep, ~60m for a debate, ~15m for deliberation, and ~15m for the oral adjudication and feedback.
- We didn’t note the timing of data-entry in Round 8 as there was no time pressure. After data entry was finished, finalising and double-checking the breaks took through to ~7-8pm.

Day	One			Two			Three	
Round	1	2	3	4	5	6	7	8
Draw generated	<i>Night prior*</i>	12:43	16:12	19:17*	12:05	15:46	19:10*	12:07
Allocation finished	<i>Night prior*</i>	13:17 +34m	16:36 +24m	20:28* +71m	12:58 +53m	16:24 +38m	21:30* +140m	13:25 +78m
Motions released	09:28	13:50 +33m	16:47 +11m	09:22	13:14 +16m	16:40 +16m	9:30	14:18 +53m
First ballot received	11:51 +143m	15:46 +116m	18:52 +125m	11:18 +116m	15:13 +119m	18:40 +120m	11:35 +125m	?
Last ballot confirmed	12:38 +47m	16:07 +21m	19:15 +23m	12:05 +47m	15:44 +31m	19:09 +29m	12:06 +31m	?

12.2 Tab Setup

Setting up a tab site is the most technically challenging (or at least annoying) part of tabbing. It is where you need to reconcile large amounts of data and configure a variety of settings to ensure everything will run without issues during rounds. While this is often done a day or two before the tournament, ideally you should look to do as much as possible in the week or two beforehand where there is much less time pressure.

12.2.1 Importing data: workflow

- First check with registration people if their data is complete, and if not who is missing. If it's only a few people it's viable (for tab purposes) to use place-holders for them, as long as you remember to follow up and edit their data manually later.
- Familiarise yourself with the different methods for importing data into your tabbing program. Typically, these include options for bulk-importing spreadsheets, for adding information piece-by-piece through a graphical interface, or a hybrid systems. Depending on your tabbing software it may be easiest to first setup your tournament on a local copy of the tab (where it will be faster to rectify mistakes) and transfer the data to the live site when everything is mostly complete.

Nota: If you are using Tabbycat our spreadsheet importer is definitely easiest to use on a local copy; however using the visual importer is perfectly viable for larger tournaments if you are not comfortable with the command line. When using the spreadsheet importer note that it will likely take several iterations to get the data to import cleanly as there will typically be small mismatches in speaker/institution names and the like.

- If the tournament (or the host society) has their own domain name and your tab software is self-hosted consider whether you want to setup the tab site on their domain so that the URL is nicer and/or easier to type.

Nota: If you are using Tabbycat, and deploying to Heroku, be sure to read our documentation about the size of Postgres database your tournament will require. Setting up the correct size of database from the start is the best way to go, as transferring information at a later stage is a hassle and could delay the tab at inopportune times.

12.2.2 Importing data: regions/societies

- Societies will often have special names that they like to use in draws (that are not the same as their institution's name or acronym). These can be gathered from institutional reps or from prior tabs. When in doubt err on the colloquial / most recognisable name; particularly for formats where teams need to find each other prior to the debate.
- If your tabbing software has methods for assigning region information to teams and adjudicators (for diversity purposes) determine with the adjudication core the types of regions that will be used.

12.2.3 Importing data: participants

- Check you have emails/phone numbers included in your data that will be imported (presuming your tabbing software supports this) there are useful to have on hand later for either emailing out information or quickly following up errant adjudicators.
- Often, the easiest way to prepare registration data for tab imports is to create new tabs in the registration spreadsheet, and use referencing to automatically order and arrange their data into the format your tab software wants. If the registration data changes significantly this will also make it easier to re-import things.
- Often some adjudicators, typically local independents, may not be available for all rounds. Try and find out who this affects and when; once data has been imported you can *pre-check these adjudicators in and out of rounds* (if your tab software supports this; otherwise note it for later).
- Remember that the swing team(s) probably also need to be imported into the tab.

12.2.4 Data import: rooms

- Ideally you want not just a list of rooms, but also of their types and categories — i.e. what building a room is in and/or it will be coded so that participants can find it.
- You want to know if access to some rooms is conditional; i.e. if some rooms are only available for some rounds. Again, if your tab software supports it you can *record this availability information into the system* (once data is imported) otherwise you can note it for later.
- Registration should have collected information about accessibility requirements; they should be imported into your tab software (if it *supports automatically matching accessibility requirements*) or note for later. In general you will also want to use a similar process to ensure that members of the adjudication core are assigned rooms that are close to the tab room.
- You also want some idea of priority; that is to say if some rooms are inconvenient (and you have more rooms than you need) they should be marked as a low priority so they will be allocated only if needed. Again, this might be automatically done by your tab software or something you will need to note and manually change after each draw is made.

12.2.5 Data import: adjudicator test/CV scores

- Ideally the adjudication core should do this themselves as they are marking the test or scoring CVs. If they won't, or you don't trust them with full tab access, be prepared to do so yourself.

12.2.6 Data import: tab access

- Set up user accounts for the adjudication core with dummy passwords (they can change them later).

- Set up user accounts for runners/assistants with dummy passwords (they can change them later).

Nota: If using Tabbycat and using online ballots or feedback with the private URLs method, participants should be emailed out their private URLs before they start travelling to arrive at the tournament (i.e. when they have a reasonable chance of checking their email). This can be done using the inbuilt pages on Tabbycat, or by importing participants data into a service such as Mailchimp.

12.3 Pre-Rounds Setup

12.3.1 Setting up the tab room

This is typically the first order of business, as all future pre-round setup tasks (i.e. training the adjudication core, testing printing, etc.) are better for being done in the same space that will be used throughout the rounds. Once you're in the space there are a couple of small checks to run through before the larger question of how to arrange and use the space should be tackled:

- Check with convenors whether things can be left in the tab room overnight. If they can't you'll need to make plans for how to move any big items (printers; ballot stacks) to and from the tab room each day.
- Check that the internet access in the tab room is reliable.
- Check that the projector system works, both with whatever wired-in computer is in the room and when connected to your laptop.
- Check what items either yourself, or the organisers, have at hand and check if anything needs to be acquired before the next day. Critical items for tab rooms are typically:
 - An extension cord with multi box; ideally a pair of each.
 - Whiteboard markers (assuming there is a whiteboard) otherwise permanent markers and large sheets of paper (i.e. A2) can suffice.
 - Boxes. Lots of boxes. Loose ballots are a source of confusion and error, so you want some way of temporarily storing ballots as they proceed through the entering and checking process. You probably want at least three large boxes (for ballots to-enter, ballots to-check, and finished ballots) but more will be useful.
 - Spare printing ink/toner, and paper for the printer. Ideally your paper would be multi-coloured, with each colour being used for a different round. Pastel colours are ideal, and you ideally want at least three different colours so that you don't have to repeat a colour within the same day. Be sure to calculate how many sheets you will need per round and ensure you have a generous number of spares.
 - If tabbing a format that can produce multiple ballots per-debate, staplers are essential to keep those ballots organised. Buy at least two sturdy ones.
- Non-essential, but often useful to have items:
 - Whatever dongles/adapters you need to connect your laptop to the projectors, both in the tab room and in the briefing room.
 - An Ethernet cable (or two) as a backup option if WiFi drops or is overloaded.
 - Post-it notes are a great way to temporarily mark ballots with information; typically used to indicate ballots that need correcting.
 - You'll often need to make impromptu signs; sticky tape and/or blu-tack are good here
 - Spare pens for the people doing data entry to use

- Trash bags for collecting rubbish as it accumulates
- A Chrome Cast can occasionally be very useful if a projector or screen doesn't have accessible input cables or so that you can use a projector without having your laptop tethered to a particular podium and desk.

If you haven't already it's a good idea to check your printing setup by printing off a bunch of generic ballots and feedback forms to have on hand if the need arises (i.e. a ballot is missing and needs to go out ASAP; or if someone can't do feedback online and needs to do so on paper). At worst, the blank ballots you print can be used for the out-rounds. While printing these off, time how long it takes the printer to print say 25 ballots and extrapolate from that to figure out how long it will take to print the entire round's worth of ballots. Note that if printing off a round's ballots is relatively quick it can be useful to delay it in order to better accommodate any last-minute changes to the draw that happen post-announcement. It's also worth thinking about how you (or at least who will) group up the printed ballots in order to distribute them to runners.

At this point you should also setup whatever process you need for managing runners and the ballot collection process. At a minimum, this should probably be a spreadsheet or a list on a whiteboard outlining the different groups of rooms with spaces to mark in which runners are delivering/collecting ballots for each location. Who is running where might change from day to day and should be kept updated. It should also have some method for contacting each runner (i.e. a cell phone number).

The question of how to arrange the actual room is one with many answers, and is obviously shaped by the peculiarities of the space itself. However there needs to be some system behind it so that people know exactly where to go and what to do when there is time pressure.

The key consideration behind this system is typically the "flow" of ballots: what happens after they are brought back from runners, but before they are completely entered into the system. Think through how you want this process to operate and how the space can be arranged to make each step as smooth as possible. Considerations:

- When runners initially return a big stack of ballots, what happens? They could be transferred directly to the data-enterers to start on, but it is often useful to have preliminary checks here in order to keep the job of the data-enterers as simple as possible. These checks could include:
 - For formats with multiple ballots per-debate, you typically want to identify and staple together all the ballots from a given panel.
 - For tournaments where ballots are liable to go missing (or for when you have plenty of data-enterers and want peace of mind) it is worth using the *ballot "check-in" system of your tab software* (if it has one) to mark off ballots as physically present in the tab room. This allows you to quickly identify which ballots are missing and begin tracking them down earlier than you would do otherwise if just waiting for the "to enter" pile to be exhausted.
 - Depending on your preferences and resources, ballots could at this stage be checked for errors. This could include a basic sweep for missing information (i.e. totals) or a comprehensive sweep that includes checking math errors, ambiguous handwriting, low-point wins, etc.). While this will delay the time between ballots arriving and being entered, it will mean that you can start correcting ballots sooner, and lessens the burden on (potentially inexperienced) data-enterers to check and catch these. If you have many runners, and they are familiar with how debating scoring works, this is recommended.
- Once this preliminary step has occurred the next task is actually entering the ballots. The number of steps here is dependent on your tab software and tab settings; you might have had the "draft" ballot be submitted online by chairs or you might have the whole two-step process of a "draft" ballot entry and the "confirmed" ballot entry taking place within the tab room. Considerations:
 - Regardless of whether you are working with a one-step or a two-step process, you want to arrange the tables where data-enterers are sitting such that their need to move is minimised. That might mean either have a central inbox of ballots to enter in the centre of the tables (such that everyone can reach it) or having multiple "clusters" of enterers around boxes.
 - If work with a two-step process you want those two steps to be an active part of the spatial arrangement. That is to say, typically there will be a grouping of enterers who are working on the initial ballot entry

(clustered around a box or boxes) and then a separate “downstream” grouping of enterers that work on confirming/validating those entries. Depending on the size of tournament and quantity of runners, you either want it so that individuals from the first group can easily pass their ballots to the box of the second group; i.e. by reaching across the table or walking a short distance. At huge tournaments, you might want a dedicated person to transfer ballots between boxes to prevent enterers having to get up.

- In a two-step process people may need to transfer roles, as generally you want to prioritise entry and then validation. Often this isn’t necessarily much more efficient, but if “rebalancing” the roles make sure that the spaces assigned to each role can accommodate extra people, and that people physically move to occupy each role.
- In general, you want to minimise the number of ballots that each enterer feels the need to “hoard” to work through to keep the work evenly distributed. If people are taking a large number of ballots to process, at the final stages of entering some people will have a bunch to work through while others will be finished. Making it easy to collect and pass on ballots in the space itself helps cut down on this while keeping entry efficient.
- While the exact spatial arrangement depends on your numbers and what furniture is available, a long rectangle is a good starting point as the ballot process is in general linear (check, enter, validate, finish). Typically, this might look like a series of tables in a row with enterers sitting on either side and with the various ballot boxes in the middle.
- When ballots have finished being enter/validated there definitely should be some sort of final “done” box. Take care how ballots are put here, a common source of error is people putting ballots there before they are fully finished.
- When ballots need to be corrected you generally want to “extract” them from this process and hand them off to a tab-director or assistant to chase up and collect. There should be a forethought process for managing this; and ideally a dedicated space for it to prevent ballots being lost and to make it easy to identify ongoing issues. This might look like a process of sticking a post-it note (outlining the error) to the ballot, and then pulling it from entry/validation and placing it on a desk. Ideally you also want one of the tab directors always *not* doing data entry so that they are immediately available to manage this process.

12.3.2 Training volunteers

If at all feasible you want to train that volunteers acting as runners and/or data enterers the day *before* the tournament starts otherwise the first round will be rough. It’s generally a good idea for this training session to generally mirror the process of running a round. It’s also generally a good idea that — even if you have enough people for dedicated runner and data-enterer roles — to train all volunteers so that they are familiar with each role and can fill in if needed. This has a couple of stages:

1. Introductions and details

- Volunteering is typically thankless and often stressful. It’s also quite a dull and mechanical process: deliver paper; collect paper; enter numbers; check numbers. Given the rather unglamorous nature of their role you want your volunteers to feel welcome and a crucial part of a wider team. When meeting everyone for the first time try and run the introductions in a non-perfunctory manner and get to know people’s background/interests and outline how valuable they are to the tournament.
- As part of this process you should, note their cell phone numbers or whatever means you will use to coordinate communication between the team.
- Figure out what will be happening during downtime and how you can make it more enjoyable. Would volunteers like to watch debates, work in the tab room, etc. Is there anything they would like during those down times (music, snacks, coffee, etc.).

2. Rooms and Running

- If runners are unfamiliar with debating in general, outline the basics of what draws are, what ballots are actually for, and what this process looks like from a debater's perspective.
- Outline how/when the printing process occurs and who will sort/assign the ballots. Now is a good time to assign different runners to the different groups/rooms that they will be working with.
- It is critical that, as a group, you actually go to everyone one of the venue groups and identify all of the venue rooms that are listed so that everyone knows exactly where to go. This may take some time. But it is a good chance to both check those rooms actually exist and pre-identify any problems that might occur with runners and debaters finding them.
- Outline in general what happens during ballot collecting: when to do it, how to approach chairs, what to do if they are slow or delaying. You should raise the chance of chairs being belligerent and outline how they (and you) should deal with this.
- If you are having runners pre-check ballots it's a good idea to fill out a few "bad" ballots to demonstrate the kinds of checking required. If you are using any communication systems (i.e. having runners mark off buildings as "done" in an online system) go through that now also.

3. Data entry and checking

- Before starting, setup logins for everyone and show them how to login. Also get an idea of what devices they will be using, or can bring, for data entry purposes. Check/ensure that they will have internet access on those devices.
- Run through this in the actual tab room; illustrating examples with actual ballots and going through the roles in the actual spots which they will occur.
- Run through how the seating/table/box arrangement works and the types of roles at different positions.
- Emphasise that in general, any ambiguities should be raised with the tab directors/assistants; i.e. that you should never guess about ballots but instead always delegate resolving issues to someone else.
- Run through the different edge cases and things to check during entry. For example Iron Person speeches, mismatched totals, entering the wrong ballot for the wrong panellist, etc (see section below). Be sure to also go through what happens when the validation step fails; i.e. when a ballot needs to be re-entered.

12.3.3 Training the adjudication core

Typically making the first-round's draw and allocation is the best time to really run through how your tab software and processes work in a "real" environment as well as the expectations surrounding their and your role. Generous amounts of time should be budgeted for this; it's not uncommon for it to take up most of an evening. It's also worth having an older tab, or a tab full of fake data handy in order to show them how, say, the feedback or allocation interfaces look like when full of data.

To kick off you should probably setup tab logins for the adjudication core as necessary, outline what kinds of access they have, and (particularly if they haven't used your tab software before) outline broadly what pages they should and shouldn't access. In particular, show them how to find and parse feedback as that is often the interface where they will be spending most of their time individually. As part of this tour outline (if you haven't already) how feedback will work, as well as the means by which the adjudication core can use the tab software to keep track of feedback as it comes in. Ideally some sort of general strategy should be formed for this, so that particular people sit out rounds, or are delegated the task of catching up on feedback at other points.

Depending on how many runners you have it may be necessary, or beneficial, if the adjudication core helps out with data entry. However, if you go down this route the adjudication core need to be highly trained; they are often much more likely than volunteers (who are less self-confident and have more experience) to make errors. Whether you do or don't do this, ensure that adjudication core members know to come to the tab room ASAP after they have finished adjudications rather than swanning around socialising or going to lunch. Draws will often be held up just by the fact that not enough adjudication core members are present to start or finish an allocation.

The first-round allocation is the last thing you want to cover. It is typically your only change to slowly and comprehensively walk the adjudication core through the allocation interface and the allocation system.

Allocation interfaces, while often complex, should be stepped through so that the adjudication core knows precisely how to operate it themselves (if needed). They should know what it can (and can't do) and how the different features can be used and activated. For example, diversity highlights might be an optional toggle (in which case you explain how to active it, when to do so, and what it represents) or there might be parts of the interface that detail information such as a room's liveness, energy, or bracket which should be highlighted and explained (i.e. how "liveness" is determined).

Secondly, and most importantly, is outlining how the automated process of adjudicator allocation operates, and how this can be made to match the adjudication core's preferences. Typically, you want to rely on automatic adjudicator allocations as much as possible in order to decrease the time taken to do an allocation; however every adjudication core has a different philosophy on what their perfect allocation looks like, and it is your job to try and align that ideal with what the automated system produces as much as is possible. The precursor to this is yourself knowing how your tab system allocation works: what is the relationship between a debate's bracket (or assigned priority/energy) and the numeric ranking of the automatically generated panel? Does the software optimise panel strength for a voting majority, or across all panellists? When does the software allocate solo chairs over panels? How does it avoid conflicts? Does it have (and enforce) particular expectations for a given adjudicator's score; or does it rely on a more relative comparison? The answers to the questions will often be dramatically different between different programs and you should know them in advance.

Most tab software will have at least some options for you to configure those automated processes — either by changing the automatic allocation's parameters directly or by controlling the ranking and feedback systems that feed into it. The first round is the prime opportunity to configure these options so that they align as close as possible with what the priorities of the adjudication core. If your feedback ranking system is mismatched with how you expect the automatic allocation to place adjudicators, or if the distribution of adjudicators across the draw is not what you expect, the adjudication core will end up wasting significant amounts of time adjusting allocations. Even if things work well using the default settings, ensure you experiment and demonstrate the consequences of changing the settings just to show that it can be done, what the general effects are, and to see if there are even-better configurations.

Nota: This process of tweaking the automatic allocation settings is one you should also revisit as the rounds progress.

How to approach diversity (typically in terms of region and gender) across an allocation in particular is something that some members of an adjudication core will not have had to consider in the context of a large tournament with time pressure or in terms of having to make explicit trade-offs. Again, you should make it clear how the software can accommodate this, and get the adjudication core to plan for how (in general) they want to approach this. Often it will form the final phase of the allocation process, and so can easily be forgotten or skipped over; or people will have different philosophies of how to approach this which are only raised at critical points.

Outline that there will usually be a trade-off between the quality of each allocations and the speed at which the tournament runs. When time is not a factor, many adjudication cores will often take an hour or more in order to create a perfect allocation; but they should know though that aiming for perfect during many rounds will break the schedule. You should try and get them to set some sort of time goal for allocations, and (during the rounds) ensure that they are aware of when they are going too fast or too slow. Depending on your personal preferences and the norms surrounding tab direction in your circuit you may want to actual enforce these time limits.

Finally, outline how you will all communicate. Again, there should be a single medium for this so that everyone knows what is going on; and this is ideally something that has been planned out beforehand with them and the organising committee. But at this point the tab team may have expanded, or there may be better options than what was being used previously. It's also worth outlining which parts of the tab team will generally be doing what roles and where — i.e. who will be rolling the draw, who will be chasing up people, etc.

12.3.4 Preparing a briefing

- At large tournaments there should be some form of briefing covering ballots and feedback process, even if it is just quick one. Usually you will want to be the person to design and deliver this; other people less-familiar with the system may miss details.
- Liaise with convenors and the other people doing briefings to ensure (a) they know you're doing one; and (b) you are not overlapping in terms of content.
- See the last section of this document for notes on what can be useful to include here

12.3.5 Final checks

- Check if the convenors have made a map that clearly outlines where the rooms are. Ensure it's clear and post it to either the tab site (ideally) or somewhere like Facebook.
- Check that convenors have some sort of way-finding system in place, i.e. chalked directions or colour-coded signs. Check these colour codes match the names of your venues.
- Check that the draw types are correct for each round in the tab system.
- Check with adjudication core if/when there are secret rounds and that these are correct in the edit data base area.
- Check how the draw will be displayed and managed. Is the projector good; how big does the text size need to be? How fast is the scroll?
- If you will pre-print ballots check that you've set the «return ballots to» configuration setting; even if it just says «to runners».

12.4 Managing Rounds

Once everything has been setup and everyone knows what they should do, the actual process of running each round should go smoothly. It probably won't though. The earlier sections should have laid out what the ideal process for managing data entry and allocations, so this section will instead focus on what can go wrong and what to keep an eye out for.

12.4.1 Disaster scenarios

There are two broad classes of disaster scenario here. The first, and more rare case is when either internet access at the venue goes out or if a web service that your tab software depends on has an outage (for example, both Tabbie 2 and Heroku-deployed Tabbycat instances depend on Amazon Web Services). The first can at least be solved temporarily if tethering is available, but if that is not possible (or the latter case occurs) you may need to switch to using an offline copy of that tab by restoring from a backup if the outage is non-transient.

Obviously, for this to work, you should be taking regular backups using whatever mechanism your tab software allows. Key times to do so are critical events such as finishing entering a round's data or finalising an adjudication allocation as these are especially difficult to recreate. Importantly, these backups are only useful to you if you have a downloaded copy of them; ideally download to a Dropbox or some other cloud service that will spread them across multiple computers and an online service.

Having an outage of internet access or a key web service go down to the point of having to switch to an offline tab is an exceedingly rare event, but one worth planning for at large tournaments. That is to say you should have ideally have an offline copy of your tabbing software setup on your local machine, and know how to restore a backup to it if necessary.

Backups are also useful as guards against a much more common source of error: data loss caused by user error. It is not unheard of for even experienced tab directors (or inexperienced adjudication core members) to accidentally delete an entire allocation, delete a round, or some other form of destructive action that would require a lot of work to redo. Taking backups at key points, and knowing how to restore them (to the online copy of the tab) is a useful — and occasionally essential — skill.

Nota: The much more common source of a major tab disruption is a major user-error or a bug within your tab software itself. Fixing these will be highly-context dependent and the best way you can prepare for them is to know your tab software well enough to understand what might have caused it or be able to contact someone else who does. That said, having backups on hand can also allow you to restore your database to before the bug or user-error occurred and try to proceed without re-triggering it.

12.4.2 Expected problems

Incorrect ballots are an inevitable tragedy. Many more optimistic tab directors will imagine that these can be prevented through sufficiently detailed briefings, recurring public shamings, or fool-proof ballot designs. While these might help in cutting down the number of errors, eliminating them entirely seems to be an unachievable goal. Note that this is particularly true at international tournaments and/or at tournaments that draw participants from circuits which have more than one predominant format.

While debaters as a whole display astonishing levels of innovation in discovering new ways to incorrectly fill in a ballot, there are a couple of broad cases that you should look out for an prepare people to deal with:

1. Not adding up score correctly. Pretty much everyone who does this will note that this is the first time that it has ever happened to them.
2. Omitting some information. Most common are not filling in total scores, the nominating winner, or the margin. Having omitted an entire team's scores or speaker names is not uncommon.
3. Scores that are outside the range.
4. Low-point wins, or tied-point wins. Typically occurs in conjunction with (1).
5. Poor handwriting rendering numbers illegible. While one could “guess” whether a number is in fact a 6 or a 5 based on a team's total score, doing so is dangerous as it assumes that the person hasn't also done (1).
6. “Correcting” information in an ambiguous way. For example, using arrows to swap a speaker's order (which is typically circular/ambiguous) or drawing numbers over other numbers in a way that makes it unclear which is the original and which is the replacement.
7. Ballots just going entirely missing because either a runner missed the room, the chair forgot to return it, or the chair just left it in the room.

Ballots aside, there are a number of other common occurrences that will necessitate changes to the drawn and allocations:

1. Teams will not turn up to debates, or turn up to debates extremely late. In both cases they will often not notifying anyone. Aside from needing to swap in a swing team in their place in the draw, it's worth keeping in mind that the necessity of a swing team might not be known until right when debates are about to start (which can lead to issues if you assume trainees or runners will be filling up the “spare” swing team).
2. Adjudicators will also go missing. As with teams this can usually be caught during roll call; but might also not be known up until debates start. If the adjudication core is available they can make adjustments, but often you will need to make a call as to whether to form an even-sized panel or to redistribute adjudicators from elsewhere.
3. When a draw is released there will often be conflicts that were unknown to the tab system, and will necessitate making changes to the draw post-release. It's important that when making these changes you keep a clear record

of what needs to change (if there are multiple swaps needed it can get tricky to keep track of) and ensure that all parties involved know about where they are being swapped to.

12.4.3 Ongoing checks

You will have a decent amount of downtime during rounds when debates are happening. A couple of things its worth keeping an eye on during that time:

- Ensuring your backups have been taken and downloaded.
- Ensuring the tab room isn't devolving into mess.
- If you can be bothered (and if no adjudication core member is doing so) reviewing feedback for critical issues (i.e. comments highlighting severe issues, or chairs getting very low scores) is a good way to be useful. If using paper-based feedback this can look like physically separating out these feedback forms for the attention of the adjudication core; while if using online feedback systems you may want to keep a collection of browser tabs to show.
- Chasing up the language committee (if one exists for this tournament) to confirm which teams are in which category and what their break preferences are (if multiple breaks are not allowed). You want to have this information confirmed as soon as possible as it becomes of critical value to allocations once the draw starts segmenting into live/dead rooms.
- Reviewing how efficiently things are running and whether there are any bottlenecks that can be better addressed in the next round. It's generally a good idea to (on a whiteboard or a spreadsheet) keep track of how long each stage of a round is taking (running, data-entry, allocation) and what (if anything) is causing delays.

Nota: If hosting Tabbycat on Heroku keep an eye on the metrics section of the dashboard area, noting if there are "timeout errors" and what the average response times are. Adding more dynos should help with both.

12.5 Breaks and Break Rounds

12.5.1 Generating the adjudicator's break

Determining the adjudicator break generally involves a complex set of considerations rather than strictly ranking based on feedback. As such most adjudication cores will use whiteboards or Google docs to draft and discuss the possible options. One thing to note here is that breaking adjudicators will need to be marked as such in the tab at some point (both so they can be on future draws, and for publication) so you want to be careful that the tab is the final source of authority here — it is easy for information to get out of sync between what the adjudication core is using to draft the break and the system.

When the adjudication core is determining the break ensure that they have an idea of the *quantity* of adjudicators needed (breaking too few or too many will cause issues) and whether there are any special considerations (such as having conflicts with large portions of the draw, or leaving at a given point) that involve a specific adjudicator being considered.

12.5.2 Generating the team break

Before doing so in an automated fashion, first check in your tab software whether all teams are assigned to the right break categories. Depending on whether your software supports multiple formats you probably also want to check that each break category is using the right "rule" specified by the tournament (i.e. a WUDC- or Australs- compliant break ranking). Also double check the break size itself is correct in the software.

Hopefully the automated system will generate a correct break, but this should always be checked against what you'd expect the results to be from standings. Note also that there are cases, such as when a team has to leave, or when teams are or are not double-breaking, that mean the automated break results need to be overridden (typically in Tabbycat you would add a marker or note to include their ranking, but exclude them from having a break rank).

12.5.3 Announcing the break

Mistakes are made surprisingly often during results announcements. Again, this is often a problem with incomplete or out of sync data, where print-outs, slides, or the tab site itself might not reflect (for example) last minute changes about breaks or have potentially mixed up teams or adjudicators with similar names. Things that can help:

- Have a single source for what is being read out — i.e. a printed list (recommended) or the tab site itself — but don't mix and match. If making slides (often a good idea for large/crowded venues) copy the data from the canonical source being announced.
- Double check what is being read out against the tab site, and/or whatever draft lists were used to determine the adjudicator's break. Verify with the adjudication core that everyone who should be there is, and that nobody is missing.
- Clarify what information should be on the print-outs and the general order in which things are read. For example, it might be easy to omit breaking adjudicator's institutions, to use ambiguous abbreviations over full institution names, or to have an inconsistent approach to how the information is read (i.e. whether it is read as *wins* then *team points* then *team name*).
- Without revealing any details try and get at least some guidance on how to pronounce names that people are not familiar with pronounce.
- Have backup copies of whatever is being read from and clarify who is reading off what portions.
- Try to publish the break list on the tab website (or via some other online method) shortly after it is announced in order to minimise the chance of misinformation spreading.

12.5.4 Managing the out-rounds

Out-rounds are generally under less time pressure and can be managed by just one or two members of the tab team. However, they tend to be run in a more haphazard fashion, so there are a couple of things to keep on top of:

- You should keep track of which adjudicators have or have not been used throughout the finals allocations. It is easy for adjudication cores to forget to allocate someone and have to either drop them or promote them beyond what they had originally intended.
- It is very easy for ballots to get lost in break rounds as chairs have less defined roles and processes in what they do with their ballots. While having correct speaker scores correctly entered for break rounds isn't a strict necessity, it is nice to have and the alternative (using fake speaks just to record the winner) can cause confusion. Closely manage distributing ballots to the chairs and collecting them as soon as possible afterwards; especially if there is any time pressure. Generally it is not worth printing off per-debate ballots; just print a stack of generic ballots at the start of the out-rounds and distribute as needed.
- You should know, in addition to when the break rounds are, when the results announcements are. Often these announcements are saved (for suspense or logistics reasons) until particular points of time (i.e. until the evening social; or until other out-rounds are finished). Obviously it's important not to accidentally release results; but often convenors and the adjudication core will often have different ideas about when results are meant to be released.

Nota: If using Tabbycat to manage out-rounds with multiple break categories, note that the round progression is no longer strictly linear. So be careful with when/if results are released online and note that often you can't rely on online

interface to release draws publicly.

12.5.5 Preparing for tab release

At some point, if you haven't already, have a discussion with the adjudication core about when the tab itself will be released and what data will be released. Well before the tab is due to be released you want to check that anonymisations and any speaker flags (i.e. Novice, ESL) are up to date in your tab software.

12.5.6 Managing the tab release

Almost there!

If hosting Tabbycat on Heroku it's worth increasing the resources available to the server for the ~12 hour period following tab release; it's by far the most concentrated burst of traffic the site will receive. Because Heroku bills by the hour, even going to a relatively expensive option, such as performance dynos with auto-scaling, will be very cheap if run just for this period. That said the site should be relatively resilient even in the face of large amounts of traffic; even running with the most basic resources allocated, at worst pages will be temporarily slow or not load.

To get an idea of how the site is performing in the Heroku dashboard keep an eye on the average request time number and adjust the number of dynos to try and keep it under say two seconds; ideally just one. When you first turn on the tab release settings, make sure you go through and load every page before announcing it to the public, doing so will trigger the caching mechanism that means potentially complex pages (say the speaker tab) don't need to be calculated from scratch each time someone loads the page.

12.5.7 Post-tournament

Once you have sufficiently recovered, consider writing up and sharing a post-script about how things went; noting things that did or didn't go well. Next year's tab directors would certainly appreciate it, and it would be great to see this kind of knowledge spread more widely. The developers of your tab software would also appreciate hearing your feedback; particularly if there were issues that could have been prevented or ameliorated by the software itself.

12.6 Appendix: Briefing Notes

This is a very loose, but not exhaustive, collection of things that are useful to communicate to speakers and adjudicators in a tab briefing. While briefing fatigue is real, having clear expectations about how things like ballots and feedback work are highly valuable uses of the tournament's time if they can at all help cut down the kinds of problems that delay the tab.

12.6.1 How feedback works

- Is it online, or offline? If online did people receive links? What do they do if they have lost it?
- Is feedback mandatory? What accountability mechanisms are there? Will you publish the shame list online or raise it in between rounds?
- Who will be submitting feedback on who? Do trainees do so?
- Remind teams that only one of their feedbacks count; they should coordinate who is doing it.
- What is the feedback scale? What does it correspond to? Common sources of confusion:

- Feedback scales are not like Uber. You do not get five stars for being adequate and generic.
- Feedback scales are not relative to position; it is an absolute scale. That is to say, if your trainee was good, they probably do not deserve the highest rating; they get whatever rating indicates they should be a panellist or low-chair.
- Consider accompanying the score/scale with a statement characterising how these numbers correspond to positions - e.g. a 4.0 means “should continue on good panels, should chair low rooms”
- If using online submission options, what should people without phones or internet access do?

12.6.2 How ballots work

This part of the presentation will be condescending. It is also necessary. The two causes of delays in the draw running late, and thus the tournament running late are (1) people not filling out ballots correctly or (2) people’s ballots going missing. Emphasise that this should be taken seriously; minutes spent chasing bad ballots are often minutes that delay every single person at the tournament from doing what they are actually here to do. You should highlight, ideally with illustrated examples:

- Which parts of the ballot *must* be filled in; people will often overlook margins, or special fields such as motion vetoes.
- That people must specify the full names of speakers; not nicknames or just-first names. Often names will be written poorly or have ambiguities (i.e. two speakers on a team called James) and having the full name is the only way to resolve it.
- That people should **not draw arrows to swap the order of speakers** as these are impossible to decipher. Here, and in other areas, always *cross-out* information clearly and write it again rather than using arrows or drawing over what is there.
- That people should try and write numbers in a manner that makes them crystal clear. Put cross-bars in 7s; bases on 1’s. Make 8’s actually look like two circles. If people know they have poor handwriting maybe consider writing the literal words — *seventy-one* below the numbers.
- That for styles that do not have a single ballot for a panel, reiterate that everyone fills in their own ballots. At Australs, if this isn’t made absolutely clear someone will average their panels ballots in order to try and “help” you.
- That runners do not fill out ballots. In BP, remind them that only chairs should fill out ballots (i.e. it cannot be deputised to a wing). In formats with individual ballots, remind chairs to make sure their wings have actually filled out a ballot, and get them to check for errors or ambiguities.
- That everyone is bad at math. People who think they are good at math just haven’t messed up their ballot *yet*. Emphasize that people should always use their phone’s calculators to check totals. At typical tournaments using exclusively paper ballots math errors happen multiple times a round, almost every round.
- How long people have to fill out their ballots. Suggest that chairs actually keep track of this time during a stopwatch, and start moving towards critical steps (i.e. scoring) well *before* the time is up, not *once* it is up.
- Outline what chairs should do to return ballots. If ballots are being run by runners, outline what they should do if a runner doesn’t appear. If they are not being run by runners remind people that returning ballots should be there number one priority, over say giving a lengthy adjudication or team feedback. Or getting lunch.
- Remind people to *be nice to runners* and that being mean to runners will have serious consequences.
- Remind people that the tab team and adjudication core will not, except for absolutely exceptional circumstances, accept photos or messaged descriptions of ballots; that all results must be on paper and handled in the same manner. The adjudication core should also be reminded of this.

12.6.3 How to locate the tab room

People should know how to get to the tab room, either to raise issues with the adjudication core or to correct ballot errors. Make it crystal clear where it is and how to get there. Also ensure people know not to barge in; that they should knock and wait.

Clearly communicate the contact details of the tab directors and get people to take them down. In most cases you do not want people going through convenors or the adjudication core for any tab-related issues.

12.6.4 Misc

Now is a good time to encourage people to consider getting involved with tabbing and tab-development. Emphasize that both do not necessarily require technical skills and that tabbers are (or should be) open to feedback and ideas from the wider community. Tell people to come find you and chat if they are interested and put up a link to the [Facebook tabbing group](#).

If you appreciated this guide we'd appreciate a slide promoting [Timekept](#) and [Debatekeeper](#). This would also be a good point to remind people that their timekeeping apps shouldn't be making noise *unless* they have been explicitly assigned to keep time by the chair.

Tab Software Comparisons

If you're reading this, you're probably interested in using Tabbycat, and wondering how it compares to other options. Perhaps you're a long-time user of another tab system, and wondering what Tabbycat can do better. This page is our effort to help answer this. Tabbycat's been around since 2010, but since BP support is a recent addition (2017), we thought it would be useful to outline the differences between Tabbycat and other BP software.

Obviously, this page is written by the developers of Tabbycat, and naturally, we have our biases. But rarely is there a single best option for everyone and every situation: different tab programs imagine the tabbing process in a different ways and have made unique trade-offs in their development process and design decisions. So we've tried to be as fair and accurate as we can, and we've consulted experienced tab directors (other than us!) and chief adjudicators to help provide a balanced overview.

At present, this guide just focuses on the major options available for the British Parliamentary format, although we'd like to expand this to incorporate the other formats that Tabbycat supports at some point in the future. As with all of our documentation, the source for this page [is on GitHub](#), and we welcome feedback and contributions.

13.1 On feature lists

In the first draft of this document, we had a table that listed every feature we could think of, along with which software does and doesn't support it. This ended up not being a great idea, for a couple of reasons.

Firstly, the largest feature disparities are for relatively niche features. All of the software discussed can do the basics necessary to run a tournament: generate draws, allocate adjudicators, enter results, etc. As a result, we will — like a good whip speech — be comparative and note key feature disparities when discussing each alternative software directly.

Secondly, we felt that the “checklist” approach to comparing tab software would do a disservice to the reasons you would actually choose one software over another. Except where a niche capability is essential, raw technical specifications rarely define the experience of using a product such as a phone, a car, or indeed, tabbing software. With Tabbycat, we've spent eight years continuously refining the tabbing workflow and smoothing out rough edges, and we believe you'll find the result extremely user-friendly and robust. As always, the best way to check this out is by *[setting up a demo site and taking it for a spin!](#)*

13.2 Comparison with Tabbie2

13.2.1 Centralised site vs individual sites

Tabbie2 and Tabbycat are both internet-based systems. Tabbie2 hosts all tournaments on a single site. However, when using Tabbycat, each tournament or organisation sets up its own site. Each model has its advantages and disadvantages in different areas:

User identification. Tabbie2’s centralised model allows for site-wide user accounts for all tournament participants. This means that they can use the same login information for all tournaments, and perform tasks such as submitting ballots and feedback through that unified account. If you’re in an established circuit, most of your participants probably already have user accounts which are identified and collected (via e-mail addresses) during registration. If you’re in a newer circuit, or one where Tabbie2 is rarely used, most of your participants will probably need to create an account — a process which Tabbie2 handles by e-mailing them a request to do so when that person is added to your tournament.

In Tabbycat’s decentralised model, there is no persistent “account” for tournament participants on each tab site or across different tab sites. Indeed, the only people who can log in to the site are those who have been given accounts by the tab director, such as tab staff and members of the adjudication core.

For secure e-ballot and e-feedback submissions, Tabbycat assigns a «private URL» to each adjudicator or team. This is essentially a password that allows a participant to only submit data that they should have access to in that specific tournament. This means participants don’t need user accounts and you don’t need to collect user account information; however if your tournament uses e-ballots or e-feedback you will need to distribute those private URLs to participants. Tabbycat can e-mail these to participants for you, or print them to give them to participants, or you could distribute the URLs using your own means.

Control over data. Some participant information in Tabbie2 is shared between tournaments, like their names and conflicts (discussed below). This means participants can manage it directly through their user accounts, without needing to go through you. On the other hand, this requires your participants to co-operate in keeping their accounts up to date, and to provide the correct e-mail address during registration (you’d be surprised how many don’t). Furthermore, participants may look to you for assistance, and your ability to help is limited to directing them through Tabbie2 channels—easy enough if they forget their password, but not so much if they forget their account’s e-mail address.

Because each Tabbycat site is its own, you’ll need to collect and import all participant details yourself. This might seem like more to do, but it also means there’s no need to match your data to existing accounts, which can be time-consuming and prone to participant error. It also means you can freely change data, for example, to correct a participant’s name or institution, or to add data like conflicts on an ad-hoc basis.

Data privacy. Conflicts are typically entered into the tab, and are sensitive information. Tabbie2’s centralisation allows for conflicts to be self-nominated by users and stored in their user accounts. This, in theory, saves the need for users to report conflicts to tab directors and other tournament staff. In practice, however, only special «super» users on Tabbie2 have access to the stored conflicts of users (otherwise anyone could access a user’s conflicts by creating a new tournament and adding that user as a participant), so many tournaments need to collect this information from participants anyway.

Tabbycat’s decentralised model means that no-one will have access to conflict information except for the tab staff of each individual instance of Tabbycat. Unlike Tabbie2, Tabbycat’s developers do not have any access to your tournament’s data — conflicts or otherwise. However, to help us continually improve the software, Tabbycat does send error reports to its developers if there is a serious bug or crash in the code, which could potentially contain confidential information depending on which page triggered the report. As a result of Tabbycat’s decentralised data storage, each tournament does need to collect and enter conflicts as part of their registration process.

When things go wrong. In our view, this is probably the most important factor. Obviously, we all hope you never have to fix things. But no software is perfect, and software developed by volunteers in their spare time (as both Tabbie2 and Tabbycat are) is especially imperfect. On occasion, glitches or edge cases occur, and fixing them requires you to directly edit the offending data in the database. Being able to do this without assistance can be the difference between a delay of minutes and a delay of hours.

In Tabbycat, because it's your site, you have full control of the database, and can edit it through Tabbycat's «Edit Database» area. This allows you to fix things (or break things, if you're not careful!). Tabbie2's centralisation prevents this—for obvious reasons, only Tabbie2's developers have direct database access, which makes their intervention necessary if direct database access is required to resolve a problem.

13.2.2 Running your tournament

Tabbie2 and Tabbycat have broadly similar workflows for running rounds; at least on paper. Key differences are discussed below:

Data import. Tabbie2 takes CSV files for import. Tabbycat has a CSV file importer, but it's (for now) only accessible through a command-line interface and is only expected to be used for large tournaments by experienced tab directors. As a more user-friendly alternative, Tabbycat also has an import wizard that's designed to make it easy to copy and paste CSV data. This works well for small and medium scale tournaments, but is cumbersome for large ones.

Public interface. Tabbycat can optionally publish the entire draw, as well as current team point standings and results of previous rounds, online. Tabbie2 shows to a logged-in user information about the debate that user is in for that round, but doesn't allow people to check up on people who are not themselves.

Position rotation. Tabbie2 uses an algorithm known as the «Silver Line algorithm», which keeps swapping pairs of teams until no further improvement is found. Because it stops at any “local optimum”, this method isn't guaranteed to be the best possible distribution of positions, and for large tournaments it often isn't. Tabbycat instead uses the [Hungarian algorithm](#), an well-known algorithm that finds the (globally) optimal allocation of positions. (One might describe this algorithm, in technical terms, as “powerful”.) Tabbycat also produces a position balance report, so that in every round you can see which teams have unbalanced position histories.

Venue allocations. Both Tabbie2 and Tabbycat allow for debate venues to be automatically assigned and manually edited. Tabbycat also allows you to specify “venue constraints” that can automatically match particular participants with their accessibility requirements, or alternatively allow for tournament staff, such as a convenor or chief adjudicator, to be allocated rooms close to the briefing hall or tab room.

Ballot entry. Both Tabbie2 and Tabbycat support entering ballots online (“e-ballots”) and entering ballots from paper from the tab room. Tabbie2 was built with e-ballots in mind, while Tabbycat was originally built for tab room staff, and the ballot entry paradigms reflect that. Both are flexible, just a little different—the best way to understand the difference is to try a demo of each. Also, Tabbycat takes note of the order in which speakers in a team spoke (i.e. who was PM and who was DPM), whereas Tabbie2 just records scores.

As discussed earlier in *User identification*; Tabbie2's e-ballots are tied to unified user accounts, whereas Tabbycat's e-ballots are tied to per-tournament and per-adjudicator “private URLs”.

Break and speaker categories. Tabbie2 has ESL, EFL and novice markers, which you can enable in a tournament's settings. Tabbycat supports user-defined break and speaker categories, so if your tournament has ESL, EFL, novice or any other form of category, you can define and customise those categories as needed.

Adjudicator allocation algorithm. Both Tabbie2 and Tabbycat use an algorithm to recommend an initial allocation of adjudicators to debates. In principle, they both work by assigning «costs» to allocations, and trying to find the minimum-cost assignment. Some notable differences:

- Tabbie2 uses simulated annealing, which is not guaranteed to be optimal and technically needs to be tuned to be effective (which you're probably not doing). Tabbycat uses the Hungarian algorithm, which guarantees an optimal solution.
- On the other hand, the Hungarian algorithm can't account for relationships between adjudicators on a panel, so adjudicator-adjudicator conflicts aren't considered by Tabbycat's algorithm (though they are highlighted in the interface).
- Tabbycat's cost function is simpler and more naive. On the other hand, Tabbie2's is more complicated and can be rather opaque (even if you read its source code).

- Tabbie2 allows for single-gender panels to be charged an additional cost. Tabbycat’s algorithm doesn’t, but the interface does provide a way to easily check for this visually.
- Tabbie2 automatically calculates the importance of a room based on its bracket (team points). In Tabbycat, debate importance can be assigned for all debates automatically based on a room’s bracket or the quantity of live break categories inside it. Instead of — or subsequent to — automatic classification any importance value can be manually tweaked as desired. These options mean there is a greater flexibility in determining which debates the allocation algorithm should prioritise.

Adjudicator allocation interface. While both interfaces use drag and drop interactions, and allow for color highlights to help identify adjudicators by gender, region, and feedback rating, Tabbycat’s allocation interface was designed to be usable on both small screens and projectors, and has a number of extra features that can help inform allocations. These features include:

- Clashes are shown directly in the interface when they apply, but dragging an adjudicator will also show you the potential conflicts that would occur if they were relocated in a new panel. This can make it much easier to avoid creating new clashes when shifting adjudicators around the draw.
- An inline display of an estimate of whether a team is “live” for each of their break categories — i.e. whether they are “safe” (have enough points to break); “dead” (cannot gain enough points to break); or “live” (still in contention).
- “History” conflicts (where an adjudicator has seen a team before, or previously was on a panel with another judge) are displayed so they can be avoided.
- Each adjudicator is present as occupying a particular position (chair, panellist, trainee) rather than having those positions calculated automatically.
- Chairs can be “swapped” by dragging adjudicators on top of each other, and an “unallocated” area can be used to view and store adjudicators that have not been allocated.

Adjudicator feedback customisation. Both Tabbie2 and Tabbycat have built-in adjudicator feedback forms, and allow you to specify the questions on the feedback form. Notable differences:

- Setting up questions is painless on neither system. Tabbycat requires you to use the Edit Database area; Tabbie2 makes you click through a slightly more opaque maze of pages and forms.
- Tabbycat allows for a richer range of types of questions than Tabbie2 does.
- Tabbie2 allows you to specify different questionnaires for team-on-chair, chair-on-panellist and panellist-on-chair. Tabbycat only differentiates between team-on-adjudicator and adjudicator-on-adjudicator.
- Tabbycat gives you more control over who is expected to submit feedback on whom; e.g. whether teams submit on panellists, and whether panellists submit on each other. In Tabbie2, you can effect this with blank questionnaires, but only for the three options listed above.
- Tabbycat can, optionally, automatically incorporate feedback into adjudicator scores using a naive weighted average with the adjudicator test score. This can be disabled by simply setting feedback weight to zero, as some adjudication cores prefer. Tabbie2 has no ability to automatically incorporate feedback.
- Tabbycat produces a «shame list» of unsubmitted feedback, which you can optionally publish on the public-facing site to try to incentivise submission.

(How participants access adjudicator feedback submission is discussed in *User identification* above.)

13.2.3 Other considerations

Offline availability. If you like, you can also install Tabbycat on your own computer, rather than host it as website on a server. This means that you can use it offline. However installing Tabbycat in this manner will require the (at least brief) use of a command line interface.

Cost. Tabbie2 is free to use. Tabbycat is free to use for not-for-profit, not-for-fundraising tournaments; tournaments for profit or fundraising must make a donation of A\$1 per team. In addition, larger tournaments that run on Tabbycat's recommended web host (Heroku) may need to purchase an upgraded database service (the free tier has storage limits) which will cost around ~US\$3 to use for the duration of a week-long tournament.

Documentation. Tabbycat has [relatively extensive documentation](#) that can be useful for learning how to use a particular feature or understanding what is happening at a technical level.

Hosting location. Tabbycat recommends using Heroku, an established cloud platform service for deploying web applications. Heroku is in turn hosted on Amazon Web Services (AWS). Both Heroku and AWS are highly reliable and widely used; downtime for both has historically been (at worst) less than 0.05 % over an annual period. Tabbie2 is hosted on [Uberspace](#); a pay-what-you-want web hosting service. To the best of our knowledge, uptime statistics are not available.

Multi-format support. If you are interested in tabbing both four- and two- team formats there may be some value in using and learning Tabbycat as it will let you use the same software in both settings.

13.3 Comparison with Tournaman

13.3.1 Native app vs web app

The crucial strength — and limitation — of Tournaman is that it is a Windows desktop application. Naturally, being a desktop app limits the features it can offer, relative to web apps like Tabbycat or Tabbie2, since it can't offer any online access. On the other hand, working with a desktop app can often be simpler than a web app.

Installation. You'll need to run (or emulate) a Windows machine to run Tournaman. Assuming you're using Windows, Tournaman's installation process is easy and familiar.

Tabbycat has a simple one-click installation process if you're deploying online (*to Heroku*). However, if you want to run Tabbycat on your own computer rather than a website, this is substantially more complicated. Local installations of Tabbycat work by having your computer emulate a web server, and while we've tried to make this as accessible as possible, a technical background is definitely helpful for this. Using our [Docker-based method](#) should be simple, but it's not 100 % reliable, and if it fails it can be difficult to figure out why. If internet access is available, we recommend running Tabbycat on Heroku.

Online features. Because Tournaman runs fully offline, it naturally can't support many internet-based features: electronic ballots, online publication of draws and live team standings, and integrated tab release. Typically, if you wanted to publish anything online from Tournaman, you'd do so by publishing the files that Tournaman generates locally. In Tabbycat, all of these are built in, so there's a single website for all tab information.

Multi-user access. Tournaman can be configured to allow networked ballot entry, but in order to set it up, you need to be comfortable with basic computer networking. This works best on small isolated networks that you control directly, e.g. a dedicated router set up in the tab room. It's not a great idea to set this up on computers connected to a university-wide network: many IT departments won't permit it, and even if they do, it's insecure, since anyone on the network can access it.

Tournaman's multi-user access is designed primarily to allow tab assistants to enter data. Key administrative tasks, such as draw generation and adjudicator allocation, must still be done on the computer on which Tournaman is installed. In contrast, web-based systems like Tabbycat and Tabbie2 allow users to login from any internet-connected device to access the functionality permitted by their account. This is often extremely useful if, say, you want to log in to a lectern computer, or have tab assistants work on mobile devices that they have with them.

If you choose to install Tabbycat offline (rather than on Heroku), it's also possible to have the computer on which the local installation resides serve the website to other computers on the same network. This then permits anyone on the same network to access the «local» installation as if it were hosted on the internet. However, like Tournaman, such a

configuration requires at least basic networking experience, and for security reasons is only advisable on small isolated networks that you control.

Backups and portability. Both Tournaman and Tabbycat (unlike Tabbie2) store data in a way that is completely accessible to you. Tournaman does this by saving files on your computer's hard drive, while Tabbycat stores data in a SQL database that belongs to you.

It should be emphasized that in both Tournaman and Tabbycat, actually needing to revert to a backup is extremely rare. Almost always, any glitch or error that breaks the tab can be resolved by editing data directly, without needing to «roll back» to a previous state. In Tournaman, this is done by editing the files that it writes to your hard drive (they're just XML files). In Tabbycat, this is done through the «Edit Database» area.

Tournaman's storage of data as XML files makes backups easy, although effort should be made to have backups stored on other computers or the cloud (e.g. on Dropbox) in case the tab computer breaks or is lost. Restoring data from those backups (or transferring the tab to a different computer) is typically a simple process of copying the files back to the original location.

As for Tabbycat, in online installations, backups can be taken easily using Heroku's [database backup capability](#). However, restoring backups requires you to have the Heroku command line interface installed. In offline installations, PostgreSQL's «dump» and «restore» commands are recommended, and may require some perseverance to get going reliably, particularly if you don't have prior SQL experience.

Generally there is no need for data portability when working with an online copy of Tabbycat — the website can be accessed anywhere. However if working with an offline/local copy, a tab can be transferred between machines by creating a backup of the database and restoring it to the other machine's database (doing so requires technical knowledge).

13.3.2 Running your tournament

Adjudicator feedback. Tournaman lets you assign judges rankings, however it does not directly manage or assist the process of collecting judge feedback. As such tab directors generally need to run a parallel feedback system, and then manually copy over changes to an adjudicator's ranking into Tournaman itself. In contrast, Tabbycat has integrated methods for collecting judge feedback that allow it to be more easily issued, collected, viewed, and automatically translated into modifications to an adjudicator's rank.

Adjudicator allocation. Tournaman has a fixed judge ranking scale and (from what we understand) has a relatively fixed procedure for allocating panels according to their absolute ranks. We are unsure about the exact mechanics of how this works, but broad details are [available here](#).

As with the discussion of allocation interfaces vis-à-vis Tabbie2, there are a number of features in the Tabbycat allocation interface that mean it is more easily used in a collaborative setting and can display additional information to inform draws.

Flexibility in draw rules. As we've said, all major tab systems are WUDC-compliant. But if you want to deviate from WUDC rules, Tournaman has a few more options. Whereas Tabbycat allows you to use intermediate brackets (rather than pull-ups), Tournaman allows you to sacrifice power-pairing integrity for position balance (though this generally isn't necessary to achieve position balance), fold within brackets and avoid teams hitting their own institution. On the other hand, Tabbycat allows you to tune how position balance trades off between teams (which the WUDC constitution doesn't precisely specify).

13.3.3 Other considerations

Stability and development. Tournaman has been in use for over a decade and is generally considered to be stable and reliable. However, new features are relatively rarely added, and its being a native app means that it doesn't boast as many features as Tabbycat or Tabbie2.

Cost. Tournaman is free to use. Tabbycat is free to use for not-for-profit, not-for-fundraising tournaments; tournaments for profit or fundraising must make a donation of A\$1 per team. In addition, larger tournaments that run on Tabbycat's recommended web host (Heroku) may need to purchase an upgraded database service (the free tier has storage limits) which will cost around ~US\$3 to use for the duration of a week-long tournament.

Availability of source code. Tournaman's code is closed-source, meaning it is not publicly available. If you do not have any coding experience this is probably not relevant to you, but if you do, having access to the source of Tabbycat can help you understand how the program works and customise it as needed.

13.4 Comparison with hand tabbing

Hand tabbing is easy, until it isn't. Traditionally, using a spreadsheet has been the go-to option for smallish tournaments because, hey, you're pretty handy with Excel, right? Making draws in spreadsheets (or on paper) seems like a pretty approachable task; ultimately it's all cells and formulae and numbers unlike the more arcane underpinnings of actual tab software.

However, hand tabbing does require you to have a good working knowledge of how your format's rules work and how your spreadsheet software of choice can be made to work them. That process might be easy for you, or it might not be. But, either way, we'd like to think that Tabbycat offers a better alternative to hand-tabbing; regardless of how well you can actually hand-tab. The setup costs of creating a copy of Tabbycat are pretty low and you can speed through the process of draw creation, adjudicator allocation, and result entry at a pace. It's still not going to be as fast a spreadsheet for a small tournament, but we think it's getting pretty close. And in exchange for a little speed you get a much stronger guarantee of your draws being correct, options for online data entry, a more comprehensive and shareable final tab, and much more. Give it a shot!

Scaling & Performance on Heroku

If you expect your Tabbycat site to gain lots of traffic — either because it has lots of participants or will be followed by lots of people online — there are a number of strategies to ensure that it will remain fast despite this attention. This is typically only necessary for very large tournaments that will display information on the public-facing version of your Tabbycat site; and even then only minor amounts of “scaling” are needed at particular moments. However, if your site does become slow you want to know what to do and how to do it ahead of time in order to prevent disruptions.

By default, a Tabbycat installation runs on Heroku’s free tier. This is a resource-constrained environment and improving performance will typically require paying for higher levels of Heroku services. These services are billed on a per-minute (not monthly) basis, meaning that if you make use of any upgrades mentioned below, you will only be billed for the actual duration used — i.e. if an upgrade is listed at \$100/month, but you only use it for 4 hours, the final bill will be around \$0.50. As such you can make use of high-performing resources just when they are needed, without needing to pay the full monthly price to maintain the resource indefinitely.

Nota: This page was largely written before a number of improvements in the 2.2 release of Tabbycat that have significantly increased traffic capacity. Its recommendations may be overly cautious given these improvements.

Nota: Scaling for performance reasons is a separate issue to that of [upgrading the database capacity](#) of your tab which just provides the ability to store data. That said, typically only tournaments that are large enough to need an upgraded database will be the ones that ever need to consider scaling for performance anyway.

14.1 Introduction to Scaling

Heroku’s primary resource is that of a “dyno”. Each dyno can be thought of as running a copy of Tabbycat. Following the “many hands make light work” principle, running a greater number of dynos will generally improve the ability of your site to serve lots of pages simultaneously because traffic is divided up amongst all of the available dynos. In normal circumstances most Tabbycat pages on the public site are served by Heroku in under a second; however they can take noticeably longer than that when the quantity of incoming traffic exceeds the capacity of the existing dyno(s)

to serve it. In such cases the delay accrues because the excess page loads need to first wait for an available dyno in addition to the standard wait for a page to be processed and transferred to the user.

Thus scaling is almost always a process of increasing *throughput* in order to decrease the chance that any page load needs to wait in order to be served by a dyno. On the flip side, having extra dynos or upgraded dynos won't make Tabbycat itself run any faster in general — they are essentially wasted unless there is sufficient large amounts of traffic that a single dyno cannot serve all of it without creating a queue.

Dynos can be scaled by adding more dynos (“horizontally”) or by adding upgraded dynos (“vertically”). In general, horizontal scaling should be the first and most effective strategy — the problem traffic causes is due to *concurrency* or lots of people loading lots of pages all at once. The traffic on Tabbycat sites typically fluctuates a lot, with moments of high intensity clustering around draw releases, round advances, and the final tab release. As such you generally only need to scale your dynos for very short periods of time.

In order to increase the number of dynos you first need to be using at least the **Standard 1X** level of dyno. Once set the Heroku Dashboard will show you a slider that can be used to increase the dyno count. While dynos higher than the **Standard 1X** level may help serve pages a tiny bit faster, having *more* dynos is far more effective than having *upgraded* dynos. Upgrading beyond **Standard 1X** is generally only required if you need additional memory (as described below) or want to use auto-scaling (also described below).

Nota: There should be no need to increase the number of “worker” dynos. While “web” dynos are responsible for serving traffic, the worker only handles a few rare tasks such as serving email and creating allocations.

At large tournaments you should always upgrade your existing “**Free**” dyno to a “**Hobby**”-level dyno. This upgrade is crucial as it will enable a «Metrics» tab on your Heroku dashboard that provides statistics which are crucial to understanding how your site is performing and how to improve said performance. If you are at all unsure about how your site will perform it is a good idea to do this pre-emptively and keep an eye on these metrics over the course of the tournament.

Nota: If pages are not loading it could be due to two things: your site being overloaded with traffic, or a bug in Tabbycat itself. Generally, if you see a generic or Heroku-branded “application error” page that means it is the former problem, whereas a Tabbycat-branded page indicates the latter.

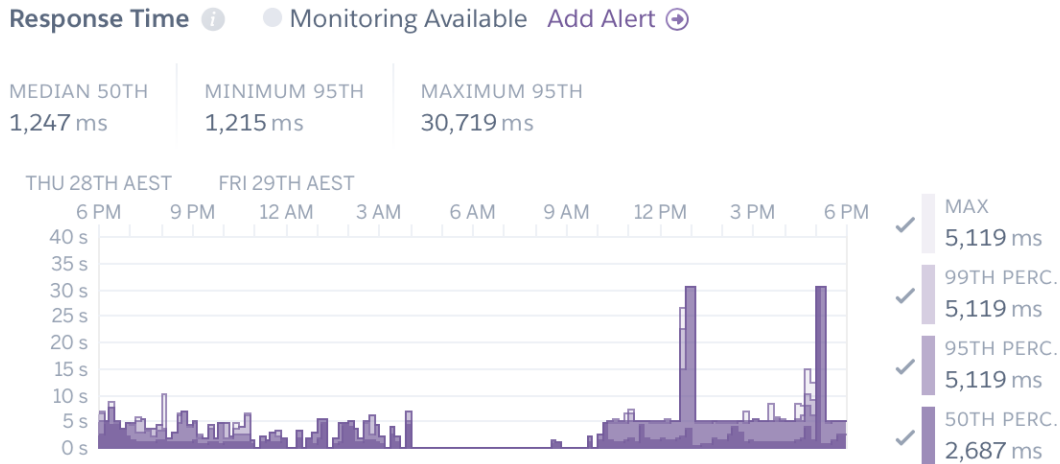
14.2 Scaling Dynos

Once you have upgraded your dyno to the **Hobby** level, the metrics tab provides a number of graphs that can be used to help identify how your site is performing. The first graph, **Events** provides an overview of any problems that are occurring:



Red marks, or those labelled *Critical* typically indicate some sort of problem. H13 errors are a good measure of the amount of pages failing to load during heavy traffic, however they can be triggered under normal conditions — you want to check how many are occurring within a given time period and ensure its more than a handful.

14.2.1 Response Time & Throughput



The response time is the amount of time it takes a dyno to load a page and serve it to your users. Smaller response times are thus good, and long response times (over fifteen seconds) indicate that a site is struggling to serve its content due to a large amount of queueing.

Heroku dynos have a maximum response time of 30 seconds, at which point they will stop serving the request. To users this appears as an error page or as a page that never loads. Thus if you see the graph is at or approaching 30 seconds at any point you need to try and diagnose what is causing this and add more resources to reduce the response time.

Nota: You can toggle the percentiles in this graph. Problems are much more severe if it affects the 50th percentile which represents a site that is probably not loading for the majority of its users. There are also a couple of Tabbycat functions (like automatic adjudicator allocation) that naturally take 10-20 seconds which means that the maximum or 99th percentile metrics are not very reflective of general traffic.

Closely related to this is the **Throughput** graph (further down) which shows how many pages your site is serving per second. Normally this is not particularly interesting, however note that the red part of the bar graph shows the amount of failed page requests. Like the **Response Time** graph, this can indicate issues with the site — normally this red portion should be well below 1rps (and ideally 0rps). If it is above 0.5 it represents a site that is producing a significant number of failed page loads.

You can verify if pages are not being served to users by checking the **Events** graph and looking for H12 errors, although occasionally they are not reported properly. A large amount of H13 errors can also be a cause for concern.

14.2.2 Dyno Load

Dyno Load ⓘ



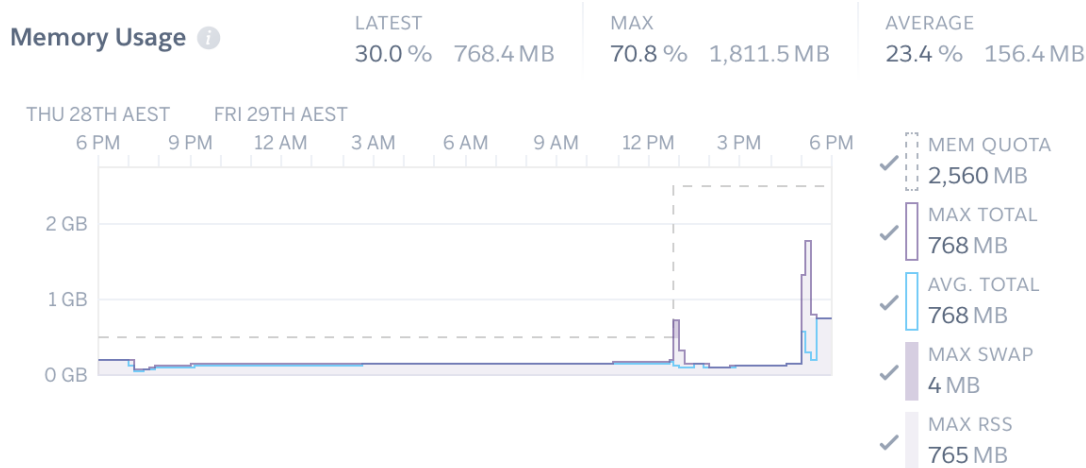
This graph shows how well your dynos are being utilised. It is scaled relative to the total number of dynos you are running (or have run previously). So if you have 10 dynos and the bar graph is near the “10” this shows that each dyno is being utilised 100 % (either on average over a 1-minute period or as the maximum use over a 1-minute period).

If this bar graph is hitting the top it will usually mean that a site that is slow or failing to load pages — if each dyno is busy it can’t serve a new page until it is finished. This issue can often compound, with more traffic coming in than it is possible to serve and clear.

If your average, rather than maximum, dyno load is approaching the upper limit of however many dynos you are running now (remember the y-axis will often exceed however many dynos you are currently running) that is a very good sign that you should increase the quantity of dynos being run. Continue adding dynos and evaluate how this effects load so that the bar is not hitting its limit.

If you are consistently needing to scale things (or having previously had issues and are expecting a very heavy burst of traffic) it may be worth upgrading to the **Performance-M** dyno type, which will then allow you to enable the *Auto-scaling* feature. This will automatically add dynos as needed to cope with traffic, and remove them when they become unnecessary. This is very effective; however, note that this dyno-type is \$250/month per dyno and will self-add dynos (within an upper limit you can specify). While this is not a huge price on a per hour/minute basis (even running 10 for an hour is only \$4) you definitely want to ensure you keep a close eye on it and turn it off when it is not necessary. Note that it also tends to be quite aggressive in how many dynos it “thinks” you need — you may want to increase the default response time threshold to prevent it scaling so quickly.

14.2.3 Memory Usage



It is very rare that Tabbycat sites will hit the memory limits of the Free or Hobby level dynos — its almost always hovering around 256MB of the (standard dyno) limit of 512MB. However, if the graph is approaching the dashed line you may want to first restart the dynos (in the *More* dropdown in the upper-right) and see if that resolves it.

You can also confirm that memory limits are causing the app to fail by checking for the presence of R14 errors in the Events chart. If your site continues to come very close to that memory limit you will want to upgrade your dynos to the higher level dynos which have increased memory.

14.2.4 Alerts

If the site is really struggling to perform its usually relatively obvious. However, if you want to be pre-warned of this, or just ensure things are as fast as possible, you can use the «Configure Alerts» feature on the Metrics page to receive emails when things are getting slow. We would recommend creating a Response Time Threshold warning of 15000ms and a Response Requests Percentage Threshold of 5 %.

14.3 Understanding Caching

When a page is “cached” it means that Tabbycat has stored a copy of the final output of the page. It can then send that copy directly to a user who is loading that page without needing to go through the normal process of fetching the data from the database, running any calculations, and formatting the results. Pages that are cached will serve quickly — if a page is taking more than a few seconds to load it usually means that page has not been cached (or your site is having too much traffic to serve pages quickly in general). The downside is that changes to the underlying data won’t update until the cache has “expired” and is regenerated. So, for example, a cached copy of the draw will not immediately reflect change to panels and a newly-release motion or tab page will not show up in the menu immediately.

By default Tabbycat caches public pages according to three levels: a 1-minute timeout, a 3.5-minute timeout, and a 2-hour timeout. The only pages on the 2-hour timeout are those that come with a full tab release — such as speaker standings, the motions tab, etc. Public pages that need to update quickly, such as the draw and homepage, are on the 1-minute timeout to ensure data is up to date. Public pages that update less frequently such as Standings, Results, Participants, and Breaks are on the 3.5-minute timeout.

Caching means that a Tabbycat site should actually perform *faster* when it is being viewed by many people at once, as the caches are constantly up-to-date and can be used to serve the majority of requests. When there is less traffic the caches are more likely to be regenerated each time someone goes to a page resulting in slower page loads. Most

often performance problems come when a popular page, such as a newly-released draw gains a large amount of traffic suddenly (such as by people constantly refreshing the draw). If the page hasn't finished caching it has to do a full page calculation for each of those new loads, which will spike the amount of resource use until the page load queue is cleared.

One way to help mitigate this is to try and load those pages first yourself to ensuring the cache is populated before other people access it. To do so you would generally open a new private browsing tab, and navigate to the specific page(s) immediately after you have enabled them. This may require going to the URL directly rather than relying on the homepage or menu (which may not have been updated to show the new information). In the case of draw releases, this can also be mitigated by not release online draws until they have been first shown on a projector (so that people aren't trying to get draw information ahead of time).

You can also increase the 1-minute timeout for the pages that are popular during the in-rounds, by going to the **Settings** section of your Heroku dashboard, clicking *Reveal Config Vars*, and creating a new key/value of `PUBLIC_FAST_CACHE_TIMEOUT` and `180` (to set the timeout to be 3 minutes i.e. 180 seconds). This should only be necessary as a last resort. Turning off public pages is also an option.

If you ever need to clear the cache (say to force the site to quickly show an update to the speaker tab) you can install [Heroku's Command Line Interface](#) and run the following command, replacing `YOUR_APP` with your site's name in the Heroku dashboard:

```
$ echo "FLUSHALL\r\n QUIT" | heroku redis:cli -a YOUR_APP --confirm YOUR_APP
```

14.4 Postgres Limits

The free tier of the Postgres database services has a limit of 20 “connections”. As with Redis, it is rare that a Tabbycat site will exceed this limit; most Australs-sized tournaments will see a maximum of 12 connections at any point in time.



You can monitor this in your Heroku Dashboard by going to the **Resources** tab and clicking on the purple Postgres link. The **Connections** graph here will show you how close you are to the limit. The first tier up from the “free” Hobby tiers (i.e. `Standard-0`) has a connection limit of 120 which can be used to overcome these limits if you do encounter them.

14.5 Mirror Admin Sites

If you *really* want to be safe, or are unable to resolve traffic issues and unable to quickly complete tasks on the admin site, it is possible to create a “mirror” of the tab site just for admin use. This site can be configured to share the same database as the primary site — meaning it is in effect always identical — but because it is at a separate URL it won’t have to respond to public traffic and so can’t be swamped with a large page load queue.

Advertencia: This requires some technical knowledge to setup and hasn’t been rigorously tested. It works fine in our experience but we haven’t tested it extensively. If using this make sure you backup (and now how to restore backups) before setting one up.

To do so you would deploy a new copy of Tabbycat on Heroku as you normally would. Once the site has been setup, go to it in the Heroku Dashboard, click through to the **Resources** tab and remove the Postgres and Redis Add-ons. Using the [Heroku Command Line Interface](#) run this command, substituting `YOUR_APP` with your *primary* tab site’s name (i.e. the app that you had initially setup before this):

```
$ heroku config --app YOUR_APP
```

Here, make a copy of the `DATABASE_URL` and `REDIS_URL` values. They should look like `postgres://` or `redis://` followed by a long set of numbers and characters. Once you have those, go to the *Settings* tab of the Heroku dashboard for your *mirror* tab site. Click **Reveal Config Vars**. There should be no set `DATABASE_URL` or `REDIS_URL` values here — if there are check you are on the right app and that the add-ons were removed as instructed earlier. If they are not set, then add in those values, with `DATABASE_URL` on the left, and that Postgres URL from earlier on the right. Do the same for `REDIS_URL` and the Redis URL. Then restart the app using the link under **More** in the top right.

Once you visit the mirror site it should be setup just like the original one, with changes made to one site also affecting the other as if they were just a single site.

14.6 Estimated Costs

As a quick and rough benchmark, here is a list of typical prices you would encounter if scaling to meet the performance needs of a high-team-count high-traffic tournament at the approximate scale of an Australs (~100 teams) or above. This is a probably an overly-conservative estimate in that it is based on tournaments run on the 2.1 version of Tabbycat. Versions 2.2 and above should perform dramatically better and thus have less need to scale using Standard and Performance dynos.

- **1x Hobby Basic Postgres Plan (\$9/month) run all day for 14 days = ~\$4**
 - A tournament of this size will require an upgraded database tier for the time when you are adding new data; i.e. during registration and rounds. Once the tab is released (and no further data changes needed) however you can downgrade it back to the Hobby Dev tier.
- **1x Hobby Dyno (\$7/month each) run all day for 7 days = ~\$2**
 - As recommended, 1 hobby dyno should be run as a baseline in order to see the metrics dashboard; but this can be downgraded a day or so after the tab has been released and traffic is sparse.
- **3x Standard 1X Dyno (\$25/month each) run 10 hours a day for 4 days = ~\$4**
 - This higher quantity of dynos should only be necessary during traffic spikes (i.e. draw releases, immediately after round advances, and tab release) but unless you want to be constantly turning things on/off its usually easier just to upgrade them at the start of each day of in-rounds (or when the tab is

published) and downgrade them at the end of each day. As mentioned earlier, you should occasionally check the *Dyno Load* in the Metrics area and adjust the number of dynos as needed.

■ **Autoscaled Performance M Dynos (\$250/month each) average of 5 run for 1 hour = ~\$2**

- For just round 1 it is a good idea to upgrade to the `Performance M` tier so you can enable auto-scaling and thus have the site automatically adjust the number of dynos to the amount of traffic it's getting (rather than having to guess the number of dynos needed ahead of time). Doing so ensures that the first round runs smoothly and means that you can then review the Metrics graphs to see what your "peak" load looks like and resume using whatever quantity of `Standard 1X` Dyno will accommodate this peak load in future rounds.

CAPÍTULO 15

Upgrading Tabbycat

Generally only want to upgrade copies of tab sites that are used on an ongoing basis for multiple tournaments, or if there is a bugfix release between when you setup your site and when the tournament is running.

Nota: Going from any version of Tabbycat 1 to any version of Tabbycat 2 won't work with the below instructions — check out the CHANGELOG for the 2.0 version for details of how to perform this upgrade.

15.1 Upgrading a Local Copy

Assuming you haven't made any changes to the Tabbycat code, upgrading a locally installed copy should just be a matter of [downloading the latest source code](#) and replacing the existing files with the new ones. If you used git to download these files initially you can just pull down the latest copy of the master branch to do this.

You would then repeat the «Install Tabbycat» instructions for your original installation method.

15.2 Upgrading on Heroku

The easiest way to upgrade a Heroku site is to create an account on Github and then to “fork” the [Tabbycat repository](#).

Once you have done this you can login to your Heroku Dashboard, go to your app, and then navigate to the Deploy tab. In this tab, adjacent to *Deployment method* select the GitHub option. This will bring up a new “Connect to GitHub” section where you can search for “Tabbycat” to find the copy of the repository you made earlier and connect it.

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. [Learn more.](#)

 master

Deploy Branch

Once connected a new *Manual deploy* section will appear. Make sure you select the *master* branch (not develop) and then click *Deploy Branch*. This will then show the app deploying and notify you when it has finished; which may take several minutes.

CAPÍTULO 16

Adjudicator Feedback

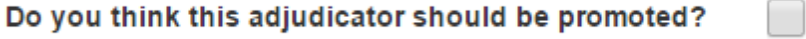
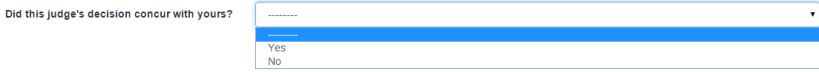




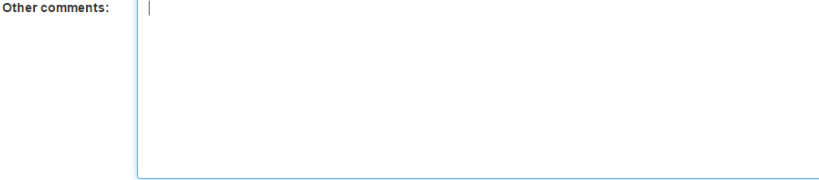
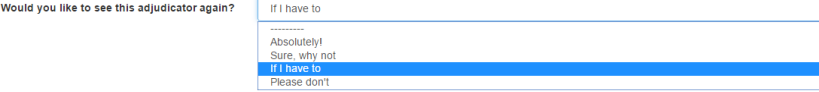
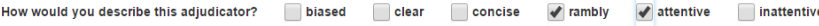
You can set the questions that are used on adjudicator feedback forms. The only field that is permanently there is the `score` field, which is an overall score assessing the adjudicator. All other questions (including a generic comments section) must be defined if you want them to be on the form.

Currently, there are two methods of setting questions:

- Through the *edit database area*. Go to **Setup > Edit Database**, then click **Change** next to *Adjudicator feedback questions*. You can add questions here.
- Using the *importtournament command*.

Most of what you need to know is explained in help text in the edit database area. (Even if you're using `importtournament`, you might find the field descriptions in the edit database area helpful.) Some more details are here.

16.1 Answer types and options

Type	Relevant options	Appearance
checkbox	-	
yes/no (dropdown)	-	
integer (textbox)	min_value, max_value	
integer scale	min_value, max_value	
float	min_value, max_value	
text	-	
long text	-	
select one	choices	
select multiple	choices	

Options:

- **min_value** and **max_value** specify the minimum and maximum allowable values in the field. Mandatory for «integer scale» types and optional for «integer (textbox)» and «float» types.
- **choices** is used with «select one» and «select multiple» types, and is a //-delimited list of possible answers, *e.g.* `biased//clear//concise//rambly//attentive//inattentive`
- **required** specifies whether users must fill out the field before clicking «submit». This requirement is only enforced on public submission forms. It is not enforced on forms entered by tab room assistants.

The exception to this is the «checkbox» type. For checkboxes, «required» means that the user cannot submit the form unless the box is checked. Think of it like an «I agree to the terms» checkbox. This isn't a deliberate design decision—it's just a quirk of how checkboxes work on web forms.

16.2 Want another answer type?

We don't really intend to add any further complexity to the built-in feedback system. If the above answer types don't cover your needs, we suggest using a third-party feedback system. You might be able to adapt [SurveyMonkey](#), [Google](#)

Forms or Qualtrics to your needs.

We may be persuaded to make an exception if the new question type you have in mind is easy to add: that is, if it is straightforward to implement using standard web page elements and fits into the existing questionnaire framework (see *Different questionnaires* below). If you think there is such a case, please contact us using the contact details in the *Authors & Acknowledgements* section.

16.3 Different questionnaires

Tabbycat allows you to specify two questionnaires: one for feedback submitted by teams, and one for feedback submitted by adjudicators. You must specify in each question whether to include the question in each questionnaire.

- **from_team**, if checked, includes the question in feedback submitted by teams
- **from_adj**, if checked, includes the question in feedback submitted by adjudicators

16.4 Who gives feedback on whom?

Tabbycat allows for three choices for which adjudicators give feedback on which other adjudicators:

- Chairs give feedback on panellists and trainees
- Chairs give feedback on panellists and trainees, and panellists give feedback on chairs
- All adjudicators, including trainees, give feedback on all other adjudicators they have adjudicated with

You can set this in the **feedback paths** option under *Setup > Configuration > Feedback*. Your choice affects each of the following:

- The options presented to adjudicators in the online feedback form
- The printable feedback forms
- The submissions expected when calculating feedback progress and highlighting missing feedback

The feedback paths option only affects feedback from adjudicators. Teams are always assumed to give feedback on the orallist, and they are encouraged to do so through hints on the online and printable feedback forms, but there is nothing technically preventing them from submitting feedback from any adjudicator on their panel.

Advanced users

If you need a different setting, you need to edit the source code. Specifically, you should edit the function `expected_feedback_targets` in `tabbycat/adjfeedback/utils.py`.

Unless we can be convinced that they are very common, we don't intend to add any further choices to the feedback paths option. If your needs are specific enough that you need to differ from the available settings, they are probably also beyond what is sensible for a built-in feedback system, and we recommend using a third-party feedback system instead.

16.5 How is an adjudicator's score determined?

For the purpose of the automated allocation, an adjudicator's overall score is a function of their test score, the current round's feedback weight, and their average feedback score. This number is calculated according to the following

formula:

$$\text{score} = (1 - w) \times \text{test score} + w \times \text{average feedback score}$$

where w is the feedback weight for the round. Note that because the feedback score is averaged across all pieces of feedback (rather than on a per-round total) rounds in which a person receives feedback from many sources (say from all teams and all panellists) could impact their average score much more than a round in which they only receive feedback from one or two sources.

Under this formula, each round's feedback weight can be used to determine the relative influence of the test score vs feedback in determining the overall score. As an example, say that an adjudicator received 5.0 as their test score, but their average feedback rating has thus far been 2.0. If the current round's feedback weight is set to 0.75, then their overall score would be 2.75. If the current round's feedback weight is set to 0.5 their score would be 3.5. If the weight was 0, their score will always be their test score; if the weight was 1 it will always be their average feedback value.

Nota: To change the weight of a round you will need to go to the Edit Database area, open the round in question, and change its *Feedback weight* value. It is common to set rounds with a low feedback weight value early on in the tournament (when feedback is scant) and to increase the feedback weight as the tournament progresses.

Nota: A participant's test score can, in conjunction with feedback weight, also be used as a manual override for an adjudicator's overall ranking. At several tournaments, adjudication cores have set every round's feedback weight to 0, and manually adjusted an adjudicator's test score in response to feedback they have received and reviewed. In this way complete control over every adjudicator's overall score can be exerted.

Nota: If feedback from trainee adjudicators is enabled, any scores that they submit in their feedback are not counted towards that adjudicator's overall score.

16.6 Ignoring/Discarding feedback

There are some cases where feedback should be discarded or ignored, but there are some differences between the two. Discarded feedback is mostly due to having another piece of feedback that supersedes the discarded ones. Ignored feedback is different as it still counts the affected feedback as submitted, just inconsequential, ignored in the adjudicator's score calculation.

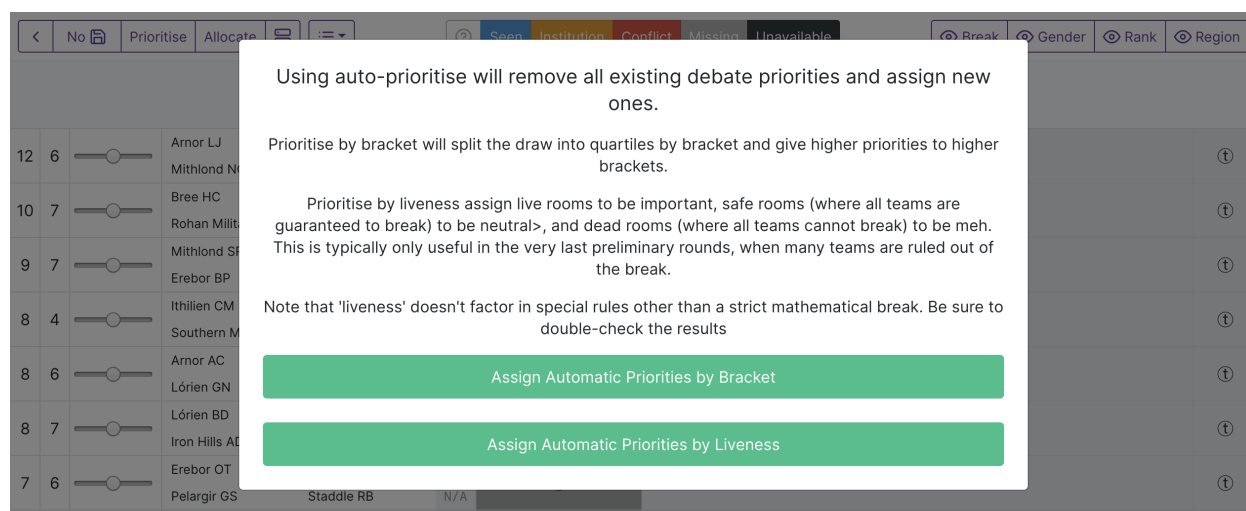
Feedback can be marked as discarded in the database view, under the `confirmed` field. It can also be marked as ignored in the same view. Controls to reverse these designations are also available there. To mark feedback as ignored, an option is available in the administrator's and assistant's feedback adding form, as well in the form of a toggle link at the bottom of each card.

Adjudicator Allocation

The adjudicator allocation screen offers the ability to automatically generate an allocation and/or allow you to create or edit an allocation manually. This interface is somewhat complex as it attempts to provide all of the information needed to inform allocation while also providing a number of automatic and manual tools for the allocation process itself.

17.1 Assigning Debate Priority

For tournaments with more than a few debates you generally want to begin the allocation process by applying a priority value to your debates. A debate's priority value is used by the automatic adjudicator allocator (or the preformed panel indicator) to match the strongest panels to the debates that need them most.



The prioritise button in the top-left allows you to assign a priority value automatically based on a debate's bracket or its "liveness". Remember that in early rounds there are usually not enough results for the liveness of each debate to be

distinct and that Tabbycat measures liveness based on the sum of all live break categories — a debate can have more liveness points than it has teams if there are teams that are live in multiple categories.

Nota: The automatic prioritiser never uses the “highest” priority value so that you can easily use this to highlight the debates that need the strongest panels without needing to redistribute the priority of other debates.

Regardless of whether you automatically assign priority, there are sliders to the left of each team that can be used to manually specify priority. These are usually used to override the automatic priority of a debate if that matchup needs and especially strong/weak/mediocre panel for reasons that are not reflected in its bracket/liveness.

Round	Break	Team 1	Team 2	Priority	Panel	Notes
9	7	Mithlond SR	Éothéod AK	N/A	©	①
		Erebor BP	Dale TM	N/A		
8	4	Ithilien CM	Dol Amroth CW	N/A	©	①
		Southern Mirkwood I	Isengard NS	N/A		
8	6	Arnor AC	Uppourn VS	N/A	©	①
		Lórien GN	UG Ringló Vale PN	N/A		

Nota: If each debate’s priority is the same, the automatic adjudicator will fall-back on using the debate’s bracket as a substitute for priority. Thus, you can skip the prioritisation process (or only prioritise the most/least important debates) if you want a relatively even spread of panellists (e.g. during Round 1).

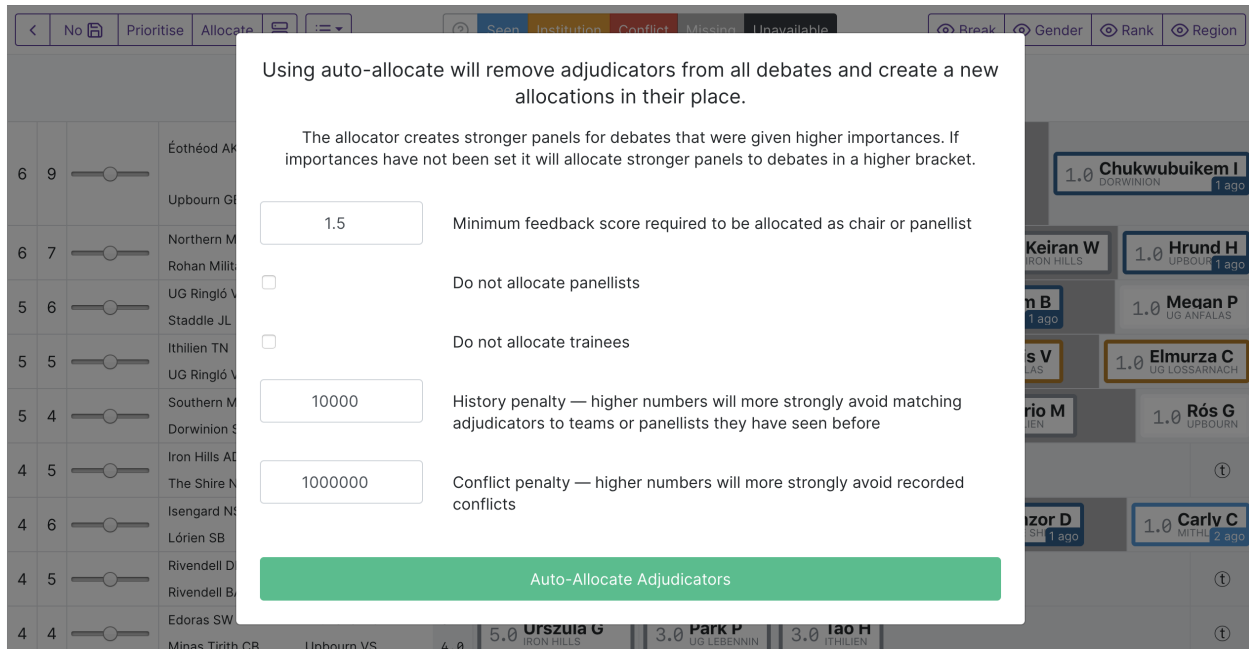
17.2 Automatic Adjudicator Allocation

Once you are happy with your priorities you can begin assigning adjudicators. This also has an (optional) automatic process that will create panels for you. In creating these panels automatically, the allocator will:

- Avoid creating “hard” conflicts (i.e. personal or institutional clashes) between adjudicators and teams while also trying to avoid “soft” conflicts (i.e. avoiding an adjudicator seeing the same team again).
- Form panels whose relative average voting score matches the relative priority you assigned to each debate.
- Allocate trainees (unless disabled or none are under the threshold) to panels (allocated first to the highest-strength panels).
- Violate the above intents in cases where there are inescapable constraints — e.g. if there are too many soft or hard conflicts to avoid creating panels that do not trigger them.

Nota: Remember that Tabbycat will only automatically allocate adjudicators that have been marked as available for this round.

To begin this process, click the *Allocate* button in the top-left. If you have *formed preformed panels* for this round, the modal will first ask whether you want to assign adjudicators using those panels; otherwise the modal will contain a number of options that can be used to control the allocation. In general, the *minimum feedback score* value is the most important setting to consider as it determines the threshold needed for adjudicators to not be allocated as trainees.



Once you click *Auto-Allocate* the modal should disappear and your panels should appear. At large tournaments, and in the later rounds, it is not unheard of for this process to take a minute or longer.

Nota: You can re-run the automatic allocation process on top of an existing allocation. Thus it is worth tweaking your priorities or allocation settings if the allocation does not seem optimal to you. Also note that the allocation process is not deterministic — if you rerun it the panels will be different.

Once your adjudicators have been allocated you can drag and drop them on to different panels. You can also drag and drop them to the “unused area” (the gray bar at the bottom of the page) if you wish to store them temporarily or remove them from the draw. Dropping an adjudicator into the chair position will “swap” that adjudicator into the previous position of the new chair.

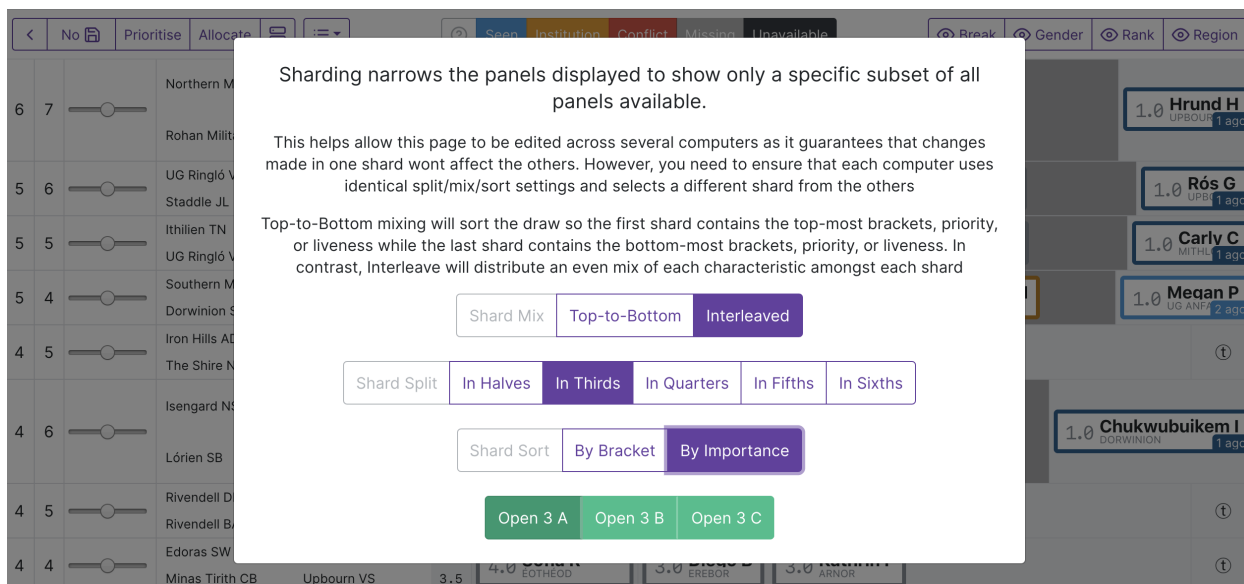
17.3 Saving, Live Updates, and Sharing

Changes to your panels save automatically and you can exit the allocation page whenever you wish.

In addition, the allocation pages maintain a “live” connection to the server that shows updates made by other users who are viewing/using the same page. That is to say, if two people on two computers are both viewing the allocation page they should see each other’s changes in real-time. This allows you to “distribute” the task of allocation across individual people (rather than sharing a screen/projector) if desired.

Nota: It is possible to have users “undo” or “overwrite” each others changes despite this live system, e.g. if both users drag and adjudicator somewhere at the same time.

If you are dividing the task of allocation across multiple people, the *Sharding* system can help by allowing individuals to limit their view of the draw. The use case here is usually to divide the draw up into mutually-exclusive subsets so that individuals (or groups) of the adjudication core can focus on creating panels across the draw in parallel.



To activate sharding, click the small icon to the right of the «Allocate» button. This then presents a number of options for how the draw is divide up into distinct sets. If you want to ensure that each person opens a completely distinct set of the draw, you will need to coordinate the options that each user selects here. They will need to set the **same** options for *Shard Mix*, *Shard Split* and *Shard Sort* but select a **different** *Open* option (i.e. opt-in to viewing a different shard from the other users). The ability to exit a sharded view is available in the same dialogue.

17.4 In-Place Highlights

Adjudicators and teams may have borders of varying colors. These borders indicate that there is a clash — soft or hard — within a debate and highlights the teams/adjudicators that have triggered this. There is a key for these colors available at the top of the page — e.g. orange means *institutional conflict* while blue means *this adjudicator has seen this adjudicator/team before*.

6	9	Éothéod AK Upbourn GB	Caras Galadhon LD Arnor LJ	3.0	5.0 Yesenia A MITHLOND	2.0 Ivona P UG RINGLÓ 1 ago	2.0 Dennis V UG ANFALAS	1.0 Bella H SOUTHERN MIRKY 1 ago
6	7	Northern Mirkwood S Rohan Military WA	Minas Tirith BV Mithlond SR	3.7 4.5	5.0 Lea B ITHILIEN	2.0 Eduardo P CARAS GALAD 1 ago	4.0 Hoàn Đ SOUTHERN MIRKWOOD	1.0 Hrudn H UPBOUR 1 ago
5	6	UG Ringló Vale RM Staddle JL	The Shire WL Lórien BD	2.6 3.0	4.0 Eleanor W ISENGARD 2 ago	2.0 Tim B EDQ 2 ago	2.0 Aalivah R IRON HILL 2 ago	3.0 Evelyn P RI ROHA 1 ago
5	5	Ithilien TN UG Ringló Vale PN	Dale TM Southern Mirkwood I	2.7 3.0	3.0 Tao H ITHILIEN	2.0 Kevin W RI ROHA 2 ago	3.0 Olivia L RIVENDELL	2.0 Oliver M STADDLE
5	4	Southern Mirkwood RI Rohan MM		3.5	5.0 Laila F	2.0 Davi C	2.0 Dario M	5.0 Sonny B

In general, you want to be on the lookout for red borders (“hard conflicts”) and for teams with orange borders (institutional conflicts). Blue borders on teams/adjudicators and orange borders between adjudicators are usually of lesser concern.

Nota: There are two “special” types of highlight — a gray background in the chair position (no chair) or in the panellist position (the panel is not an odd-size). Adjudicators may also have a black background if they have not been marked as available.

17.5 Hover Highlights

When you hover over an adjudicator or team, they will take on a purple background and other adjudicators or teams may suddenly have different colored backgrounds. These indicate the conflicts that this team/adjudicator has with those other teams/adjudicators. By showing this information you can avoid swapping that adjudicator into a new debate which they have a conflict with.

Kevin Werner (M)		Ri Rohan (Rohan)		2.0 Feedback Score B- Relevant Rank																																													
UG Lebennin		Ri Rohan		Gabriel Curran		-1R		Carly		Laila		Boubker		Dario		The Shire WL		Arnor LJ		Minas Tirith BV		Iron Hills HS		-2R		Tellervo		Yesenia		Crispus		Aaliyah		UG Lossarnach OR		Northern Mirkwood NS		Upbourn VS											
6	9			Upbourn GB		Arnor LJ		1 ago		3.5		5.0		Yesenia A		MITHLOND		2 ago		2.0		Ivona P		UG RINGLO		1 ago		2.0		Dennis V		UG ANFALAS		1.0		Bella H		SOUTHERN MIRK		1 ago									
6	7			Northern Mirkwood		Minas Tirith BV		1 ago		3.7		5.0		Lea B		ITHILIE				2.0		Eduardo P		CARAS GALA		1 ago		4.0		Hoàn Đ		SOUTHERN MIRKWOOD		1.0		Hrund H		UPBOURN		1 ago									
				Rohan Military		2 ago		Mithlond SR		4.5																																							
5	6			UG Ringló Vale RM		The Shire WL		1 ago		2.6		4.0		Eleanor W		ISENGARD		2 ago		2.0		Tim B		EDG		2 ago		2.0		Aaliyah R		IRON HILL		2 ago		3.0		Evelyn P		RI ROHA		1 ago							
				Staddle JL		Lórien BD				3.0				2.0		Kevin W		RI ROHA		2 ago																													
5	5			Ithilien TN		Dale TM				2.7		3.0		Tao H		ITHILIE				3.0		Olivia L		RIVENDELL				2.0		Oliver M		STADDLE				1.0		Carly C		MITHIL		1 ago							
				UG Ringló Vale PN		Southern Mirkwood I				3.0																																							
5	4			Southern Mirkwood		Ri Rohan MM				3.5		5.0		Laila F		LÓRIEN		1 ago		2.0		Davi C		CARAS GALA		2 ago		2.0		Dario M		ITHILIE		1 ago		5.0		Sonny B		PELARGIR									
				Dorwinion SS		Rivendell CL				5.0																																							
4	5			Iron Hills AD		Iron Hills LE				3.7		5.0		Jana F		UG ANFALAS				3.0		Cintio A		UG LOSSARNACH				3.0		Fareeq S		MITHLOND																	
				The Shire NE		Erebor BP				4.0																																							
4	6			Isengard NS		Caras Galadon BL				3.0		5.0		Marvin L		UG ANFALAS		1 ago		2.0		Abraham W		EOTHEOD				3.0		Addison M		DOL AMROD		2 ago		1.0		Chukwubikem I		DORWINION		1 ago							
				Lórien SB		Minas Tirith EN				4.0										2.0		Johanna H		NORTHERN MIRK		1 ago																							
4	5			Rivendell DD		Bree MN				3.3		5.0		Facino L		ISENGARD				3.0		Camilla E		ROHAN MILITARY				2.0		Anzor D		THE SHIRE																	
				Rivendell BA		Bree HC				4.0																																							
4	4			Edoras SW		Mithlond NC				3.2		4.0		Sofia R		EOTHEOD				3.0		Diego B		EREBOR				3.0		Kathrin F		ARNOR				2.0		Keiran W		IRON HILLS									
				Minas Tirith CB		Upbourn VS		2 ago		3.7										4.0		Uchrechukwu U		EREBOR																									
3	8			Iron Hills RM		Ithilien CM				3.3		4.0		Rashidah B		SOUTHERN MIRK...				3.0		Zac G		UG LEBENNIN				3.0		Merry B		UPBOU		2 ago		1.0		Elmurza C		UG LOSSAR		2 ago							
				Éothéod YA		UG Anfalas BL				3.5																																							

When you hover over an adjudicator or team the top-most area of the screen will show additional information about them, such as all of their previous institutions, their conflicts, their break category, their team members, their region, and who they saw in the last few rounds.

17.6 Toggle Highlights

In the top-right of the interface are a number of toggles that changes the color of adjudicators and teams to more easily check specific types of information. For example, selecting the gender toggle will color-code teams and adjudicators with the gender that has been recorded in Tabbycat. Note that when a toggle is active, the color key will update to show the meaning of these new colors.

<

at 18:12

Prioritise

Allocate

Male

Female

Other

Unknown

Break

Gender

Rank

Region

6	9	<div><div>Éothéod AK</div><div>Upbourn GB</div></div>	<div><div>Caras Galadhon LD</div><div>Arnor LJ</div></div>	3.0	3.5	<div>5.0 Yesenia A</div> <div>MITHLOND</div>	<div>2.0 Ivona P</div> <div>UG RINGLO 1 ago</div>	<div>2.0 Dennis V</div> <div>UG ANFALAS</div>	<div>1.0 Bella H</div> <div>SOUTHERN MIRKY 1 ago</div>	
6	7	<div><div>Northern Mirkwood S</div><div>Rohan Military WA</div></div>	<div><div>Minas Tirith BV</div><div>Mithlond SR</div></div>	3.7	4.5	<div>5.0 Lea B</div> <div>ITHILIEN</div>	<div>2.0 Eduardo P</div> <div>CARAS GALAI 1 ago</div>	<div>4.0 Hoàn Đ</div> <div>SOUTHERN MIRKWOOD</div>	<div>1.0 Hrund H</div> <div>UPBOUR 1 ago</div>	
5	6	<div><div>UG Ringló Vale RM</div><div>Staddle JL</div></div>	<div><div>The Shire WL</div><div>Lórien BD</div></div>	2.6	3.0	<div>4.0 Eleanor W</div> <div>ISENGARD 2 ago</div>	<div>2.0 Tim B</div> <div>EDC 2 ago</div>	<div>2.0 Aalivah R</div> <div>IRON HILL 2 ago</div>	<div>3.0 Evelyn P</div> <div>RI ROHA 1 ago</div>	<div>1.0 Rós G</div> <div>UPBEI 1 ago</div>
5	5	<div><div>Ithilien TN</div><div>UG Ringló Vale PN</div></div>	<div><div>Dale TM</div><div>Southern Mirkwood I</div></div>	2.7	3.0	<div>3.0 Tao H</div> <div>ITHILIEN</div>	<div>3.0 Olivia L</div> <div>RIVENDELL</div>	<div>2.0 Oliver M</div> <div>STADDLE</div>	<div>1.0 Carly C</div> <div>MITHLOND</div>	
5	4	<div><div>Southern Mirkwood</div><div>Dorwinion SS</div></div>	<div><div>RI Rohan MM</div><div>Rivendell CL</div></div>	3.5	5.0	<div>5.0 Laila F</div> <div>LORIEN 1 ago</div>	<div>2.0 Davi C</div> <div>CARAS GALAI 2 ago</div>	<div>2.0 Dario M</div> <div>ITHILIE 1 ago</div>	<div>5.0 Sonny B</div> <div>PELARGIR</div>	<div>1.0 Megan P</div> <div>UG ANF 2 ago</div>
4	5	<div><div>Iron Hills AD</div><div>The Shire NE</div></div>	<div><div>Iron Hills LE</div><div>Erebor BP</div></div>	3.7	4.0	<div>5.0 Jana F</div> <div>UG ANFALAS</div>	<div>3.0 Cintio A</div> <div>UG LOSSARNACH</div>	<div>3.0 Fareeq S</div> <div>MITHLOND</div>		

Nota: When finalising an adjudication you may want to ensure you have turned off any toggle highlights — often they make it more difficult to see the border colors that indicate conflicts.

Tabbycat doesn't provide an in-built backup system; instead you should create copies of your database directly. Heroku provides a very good backup utility for all sites hosted on Heroku which makes this easy, and for Heroku-based Tabbycat sites, we strongly recommend it.

You should **always** back up the database before deleting *any* data while in the Edit Database area, because deleting data cannot be undone. It is also a good idea to back up the database before doing anything in the Edit Database area, unless you're very familiar and confident with editing the Tabbycat database directly.

You may, as a matter of standard practice at large tournaments, wish to back up the database twice per round: Once just after you've generated the draw and allocated adjudicators, and once just after you've finished entering results.

If you're using an online version of Tabbycat, it's a good idea to download the backups. While it's extremely rare to lose internet access or have an outage in a critical web service (*i.e.*, Heroku), having a local copy of your backups allows you to *restore your tab to a local installation* if this ever happens.

18.1 Installations on Heroku

Heroku provides a utility to easily back up and restore the entire site database.

18.1.1 If you don't have the Heroku CLI

You can capture backups from the Heroku Dashboard:

1. Go to the [Heroku Dashboard](#) and click on your app.
2. Under *Installed add-ons*, go to **Heroku Postgres**.
3. Scroll down, and click on the **Capture Backup** button.
4. Once the capture has finished, a **Download** button will be available.

You can't restore a backup without the Heroku Command Line Interface (CLI), so if you end up needing your backup, you'll need to install the [Heroku CLI](#), and then follow the instructions below.

18.1.2 If you have the Heroku CLI

The best guide to backing up databases is the [Heroku Dev Center's PGBackups guide](#).

To capture a backup:

```
$ heroku pg:backups:capture
```

To download the most recently captured backup:

```
$ heroku pg:backups:download
```

To restore a backup:

```
$ heroku pg:backups:restore
```

If you have multiple Tabbycat sites, you'll need to specify which one by adding `--app mytournamentname` to the end of the command.

18.2 Local installations

There are lots of ways to back up local PostgreSQL databases, but we'd suggest using the `pg_dump` and `pg_restore` commands.

18.3 Restoring a Heroku backup to a local installation

As detailed in the [Heroku Dev Center](#), you can restore a downloaded Heroku backup to a local installation. This might be useful if, say, your internet connection breaks irrecoverably in the middle of a tournament and you need to run offline. Of course, for this to work, you need to have downloaded your backup before your internet connection broke—a good reason to download a copy of your backups as soon as you make them.

Assuming your download is called `latest.dump` (this is the default name), your PostgreSQL username is `tabbycat`, and you wish to call your local database `fromheroku` (if not, replace arguments as appropriate):

```
$ createdb fromheroku -h localhost -U tabbycat
$ pg_restore --no-acl --no-owner -h localhost -U tabbycat -d fromheroku latest.dump
```

Breaks and Break Rounds

In Tabbycat, elimination rounds (sometimes called *outrounds* or the *final series*) are called «break rounds», and the qualification of teams to compete in the elimination phase of a tournament is called the «break».

19.1 About break categories

Tabbycat supports multiple and arbitrarily-named break categories. Most tournaments will have just one category, typically called «Open», leading to the grand final. Some tournaments also have restricted-eligibility break categories, for example, for novice teams or teams for whom English is a second language.

Having multiple break categories is intended for tournaments where multiple *parallel* elimination phases derive from the *same* preliminary rounds (inrounds). It's not for parallel but distinct competitions—for those, you should create distinct tournaments.

19.1.1 Break qualification rules

Tabbycat supports several break qualification rules, and each break category must be configured to use one of them. Most tournaments will use «Standard», which is the default.

Rule name (string to use in <code>importtournament</code> CSV files)	Description
Standard (<code>standard</code>)	The top n teams break. This is the default, and most tournaments use this rule.
AIDA 1996 (<code>aida-1996</code>)	The top n teams that are also in the top three teams from their institution break.
AIDA 2016 (Austral) (<code>aida-2016-australs</code>)	The top n teams that fulfil either of these criteria break: <ul style="list-style-type: none"> ■ They are in the top n teams overall, and in the top three teams from their institution. ■ They have at least as many wins as the nth-ranked team, and they are the top team from their institution. If fewer than n teams fulfil either criterion, then the best teams not fulfilling the criteria are added to make n teams.
AIDA 2016 (Easters) (<code>aida-2016-easters</code>)	As for AIDA 2016 (Austral), except that if fewer than n teams fulfil either criterion, then only the best teams who are in the top three teams from their institution are added to make n teams.
WADL division winners first (<code>wadl-div-first</code>)	The division winners are taken first, then the best teams who did not win their division are added to make n teams.
WADL division winners guaranteed (<code>wadl-div-guaranteed</code>)	The division winners are guaranteed to break, and the best teams who did not win their division are added to make n teams. (Teams are sorted in their original rankings, unlike the «division winners first» rule.)

Nota: The break generators are somewhat more complex than described in the above table: among other things, they also handle cases where there is a tie for the last place in the break, and for those break categories marked «general», they will show where ineligible teams would have broken, had they been eligible.

19.2 Setting up break categories and rounds

For each break category in your tournament, you need to do three things:

1. Create (and name) a break category
2. Create break rounds for the category
3. Set the eligibility of teams to compete in the category

If you only have one break category (open) and you create your tournament using the «Create New Tournament» page, simply enter the number of teams in the break (*e.g.*, 8 if you're breaking to quarterfinals). Tabbycat will create the break category and break rounds for you. You'll still need to set the eligibility of teams though, as in (3) below. For any further break categories, you'll need to do all three steps yourself.

If you create your tournament using the `importtournament` command or in **Edit Database**, you'll need to do all three steps above yourself.

19.2.1 1. Creating break categories

If using the `importtournament` command, there is an example file, `break_categories.csv`, that you can copy and adjust. If using **Edit Database**, add categories under **Break Qualification > Break categories**.

Most of the fields are self-explanatory or described on the Edit Database form, except for one: «rule», which sets the break qualification rule. Permissible values are described in [Break qualification rules](#) above. If using `importtournament`, be sure to use the correct string (in brackets in the table). The rule defaults to «Standard» (`standard`).

Nota: The «institution cap» field was removed in Tabbycat 1.0. All Australs break qualification rules are now hard-coded to a cap of three teams per institution.

19.2.2 2. Creating break rounds

You should create a round for every break round you intend to hold, including it in *rounds.csv* if using *importtournament*, or adding them under **Tournaments > Rounds** if using **Edit Database**. Be careful to set the following fields correctly:

- *Break category* must be set to the relevant break category.
- *Stage* and *draw type* must both be set to «Elimination».

19.2.3 3. Setting break eligibility

Once a break category has been created it will not have any teams eligible for it, even if it was marked as «Is general». To edit the eligibility of teams for any break round go to the **Breaks** item in the left-hand menu for a particular tournament and then click **Edit Eligibility**.

Here you can select «all» or «none» to toggle all team eligibilities or edit them using the tick boxes. Once you **save** it should return you to the main break page which will display the number of teams marked eligible.

Nota: Adjudicators can be marked as «breaking» on the **Feedback** page; clicking **Adjudicators** on the breaks page will take you straight there.

19.3 Generating the break

Unlike team or speaker standings, each category's break (and the break ranks of teams) are not determined automatically and updated continuously. Instead each can be generated (and regenerated) as desired.

To do so go to the **Breaks** item in the left-hand menu and then click the white button that corresponds to the break category you'd like to determine the rankings for. When prompted, select **Generate the break for all categories** to display the list of breaking teams.

From this page you can update the breaking teams list for this break category (or all categories) as well as view and edit "remarks" that account for cases in which a team may not break (such as being capped or losing a coin toss).

Prudencia: Please double-check the generated break before announcing or releasing it. Although the break generation code is designed to handle edge cases, we don't test the code for such cases.

19.4 Creating draws for break rounds

Creating a draw for a break round proceeds as normal, except that the team availability process is skipped. Instead, when you visit the availability page for that round it will have automatically determined which teams should be debating based upon the determined break for that category. Once a draw has been generated it will then use the

relevant break ranks to create the matchups (ie 1st-breaking vs 16th-breaking, 2nd vs 15th, *etc.*). Subsequent break rounds will then also automatically determine matchups based on the previous round's results and room ranks.

If the «break size» of a break category is not a power of 2, it will treat the first break round as a partial-elimination draw and only create a draw for the teams not skipping the partial-elimination round. Subsequent break rounds will then process as described above.

Check-Ins

A “Check-in” is a record of a speaker, adjudicator, venue, or ballot’s status at a particular point in time. Typically these are used at large tournaments to reliably track who is or is not present for the first round of each day.

Check-ins serve a similar purpose to the *availability* system. However availabilities are tied to a particular round rather than to a particular time — they are generally used to record instances where you know ahead of time whether a person should or should not be included in a draw. In contrast, check-ins are useful for when you *don’t* know ahead of time whether a person will be able to be put into the draw and so want to be able to confirm their presence with a high degree of confidence. That said the two systems interact — the standard availability pages allow you to easily set all adjudicators or teams who have checked-in as available for a given round.

20.1 Check-In Identifiers

Check-ins are associated with a “identifier” — a number that is unique to each speaker and adjudicator. To generate these numbers go to the *Identifiers* section under the Check-Ins menu. From here you generate identifiers for Speakers, Adjudicators, and Venues as needed. Note also that Identifiers can be manually added or edited in the *Edit Database* area if necessary.

Identifiers Overview

Speakers	Adjudicators	Venues
With identifiers 174/178 <div><div></div></div>	With identifiers 0/80 <div><div></div></div>	With identifiers 24/24 <div><div></div></div>
View barcodes >	Generate all identifiers	View barcodes >
Generate missing identifiers		

Once this number has been generated it can be transformed into a barcode so that it can be easily included on tournament badges or otherwise printed and disbursed. On the same *Identifiers* page you can use the *View barcodes* option to open up a page that lists all the barcodes for the speakers, adjudicators, or venues.



Nota: The identifiers for ballots are automatically generated when printing ballots.

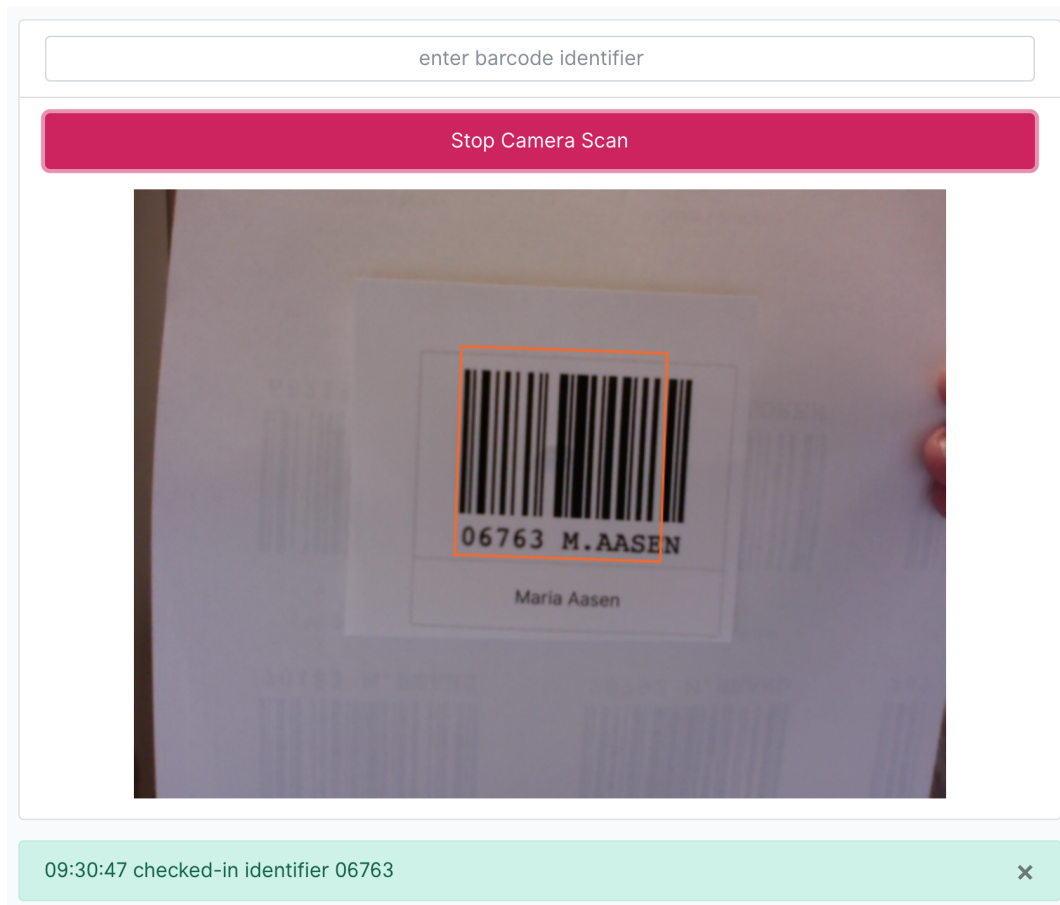
20.2 Recording Check-Ins

On the *Scanning* section of Check-ins you can record a particular check-in. This can be done in a few different ways:

1. You can type in the Identifier number into the box. Once five numbers have been identified it will automatically issue the check-in and clear the input field for the next number.
2. If you have purchased barcode scanners and configured them as USB keyboards they should then be compatible with this page: upon page load the cursor should be positioned in the input field, and any scanned barcodes should populate it with the specified number, issue the check-in, and then clear the box for the next scan.

Nota: Barcode scanners are probably cheaper than you think. A perfectly serviceable model should be around \$20 USD or less although a cursory Google search might at first turn up models that are many times that amount. If you are a curious tab director, or are involved in a tournament with a little spare money, having just one or two around for critical tasks can be highly valuable.

3. If your device has a (web)cam you can use the Scan Using Camera button. Any barcodes put in front of the camera's video stream will be scanned into the form. A sound will play when a barcode has been identified — so be aware that turning sound on or using headphones can help identify when a scan has been made.



Nota: Camera scanning works on most modern browsers although it will only work with Safari 11 or higher (iOS 11+ and macOS 10.13+). Camera scanning may also not work when using a local-installation of Tabbycat in all browsers, *except* Firefox where it seems to. Depending on the quality of your camera barcodes that are less than 4cm wide may not be recognised — ideally barcodes should be at least 5cm if using this method as your main way of checking-in things.

4. The Check-in status page (described below) allows assistants and administrators to manually check-in particular people or entire institutions without needing to know their identifiers.

20.3 The Check-In “Window”

Because Check-In events are not explicitly linked to rounds there is essentially a “window” or time period in which a check-in is still considered valid. The time of this “window” in hours can be set in *Setup > Configuration > Data Entry*.

At tournaments the run check-ins during the start of each day the check-in “window” (i.e. the time before check-ins expire) you can leave this window setting at the default time (12 hours) which should give enough time to distinguish between the first check-ins of that day as compared to the last check-ins of the previous day. At tournaments where you want to run a check-in process at the start of every round you may want to set the time to around 2 hours or something much shorter.

20.4 Viewing Check-Ins

On the *People Statuses* section of Check-ins you can view who has or has not been checked-in. This page will live-update with the latest check-ins so you should be able to leave it open to monitor income attendances.



Check In Statuses

Both Missing Present

Both Adjudicators Speakers

Sort by Institution Sort by Name

Institute of the Shire

Check-In All ✓

ALHASAN ATIYEH ✓	CHÂN VÕ ✓	CORDELIA SAGESE ✓	ERLEND SANDBAKKEN ✓
JOSEPH STEELE ✓	MUSETTE LAJOIE ✓	RUAIRIDH KENNEDY ✓	

Northern Mirkwood University

Check-In All ✓

CZESŁAWA KUCHARSKA ✓	JULIE JOHANSEN ✓	LELIO TOSCANI ✓	LORÁND HERCEG ✓
MEGAN PEARSON ✓	RENATA HAJNÁ ✓	SETH SIMOS ✓	SONNY BJÖRK ✓

Royal Institute of Rohan

Check-In All ✓

BIRTHE DAVIDSEN ✓	CINTIO ARMIJO ✓	DAVI CORREIA ✓	ODILI NDUBUISI ✓
OLIVIA LORENZEN ✓	TEA VAN DER POEL ✓	THEODOR MASCARENAS ✓	THINDRA JONASSON ✓

The blue «tick» boxes allow you to manually check-in people and/or entire institutions (for *People*) or venues and/or venue groups (for *Venues*) , without the need to scan their identifiers. This style of check-in is designed for use an auditorium roll-call type situation where you might be running through a list of people to the room or identifying absences on a per-institution basis.

Nota: A public version of this check-in status page can be enabled under *Setup > Configuration > Public Features* which can be useful for allowing people to self-police check-ins and/or validate their check-in worked.

20.5 Check-Ins for Ballots

Ballots can be checked-in to quickly validate which ballots are physically present in the tab room. This can help more quickly identify ballots that are missing. Which ballots have or have not been checked-in will show up on the Results page. Ballots can be scanned using the standard “Scan Identifiers” page.

Barcodes are automatically assigned and included to ballots when they are printed. Barcodes have no check-in window — any relevant check-in event counts regardless of how long ago it was.

20.6 Check-Ins for Venues

Venues can be checked-in, but what a “venue check-in” means is a bit more flexible. It might be used to validate which rooms are debate-ready at the start of a day (i.e. unlocked; has a desk) or it could be used during the rounds to record which rooms have returned their ballots.

Venues have a separate check-in window setting to that of people.

Venues have their own Status page (like people) and can be checked-in there manually. Like speakers and adjudicators their barcodes can also be printed off.

Entering Ballots and Feedback

21.1 Ballot check-in

For tournaments that require it, there is a «ballot check-in» page that can be used to record the arrival of ballots to the tab room. When there's a missing ballot, it can help establish whether the ballot never made it to the tab room, or whether it's probably floating around in the room forgotten. Also, it can help enforce early checks that panels return the correct number of ballots to the room.

To get to the ballot check-in, click the relevant round in the menu of the admin area, and then click «Results» and then «Ballot Check-In». This requires superuser privileges.

There's no adverse effect from not using the ballot check-in. Data enterers will still be able to enter and confirmed ballots, even if not checked in.

Truco:

- Since the ballot check-in tends to require a dedicated computer or two, it can be worth creating a separate superuser account for ballot check-in, so that it doesn't appear on the action logs as being by a particular person.
 - Don't forget to provision a computer or two for this if you're planning to use it.
 - Ballot check-ins can be a bottleneck, so you might decide they're not worth using. Alternatively, you might have multiple computers for this purpose, or you might dedicate a tab room helper to driving the process (since this is probably faster than runners doing the typing in turn).
-

21.2 Ballot entry

demo
Enter Results
Enter Feedback
View Draw

New Ballot Set for Caltech 1 vs Pennsylvania 1

Round demo - Round 2 @ K04

Motion

Chosen
1. This House would not buy Apple

Aff Veto
2. This House would end all research and development tax credits for highly profitable technology companies

Neg Veto
3. This House regrets the rise of computer science at the expense of the humanities

Ballot from Leigh McGee (Harvard)

Caltech 1		Pennsylvania 1	
1A	Georgia Crawford 76	1N	Alma Spencer 75
2A	Helen Keller 77	2N	Leigh Holmes 79

Most tab rooms run some sort of check system to ensure data is entered accurately. In Tabbycat, this is built into the system, which also helps speed it up.

As a general principle, Tabbycat requires all ballots to be looked at by two people. The first person enters the data from the ballot, and the second person checks it. The second person isn't allowed to modify the data—they either confirm it or reject it, and if they reject it, then the whole process starts again. This is by design: to be confirmed, the *same* data must have been seen by at least two people.

Prudencia: The administrator area does **not** work like this. It's designed to be flexible, so allows you to edit, confirm or unconfirm any ballot at any time. For this reason, you should use the **assistant** area to enter ballots, even if you have a superuser account.

Truco:

- Don't forget to check the totals against the ballot—they're a useful integrity check too.
- Don't forget to check the winner against the ballot! If the adjudicator gets it wrong, it's worth asking to clarify.
- It can be helpful to think about the room layout to maximize efficiency.
- Some tab rooms like to assign some to data entry and some to verification. This isn't really necessary, since Tabbycat doesn't let the same person enter and verify the same ballot. (This is one of many reasons why every person should have their own account.)
- Emails can be configured to be sent to adjudicators as a receipt of their ballot once confirmed.

21.2.1 Duplicate/Swing Speeches

'Iron' speeches

Yes; there were speakers who spoke multiple times

Ballot from Toby Wong (MIT)

Yale 3		Cornell 1	
1A	<div></div> <div>Mark this as a duplicate speech: <input type="checkbox"/></div>	1N	<div></div> <div>Mark this as a duplicate speech: <input type="checkbox"/></div>
2A	<div></div> <div>Mark this as a duplicate speech: <input type="checkbox"/></div>	2N	<div></div> <div>Mark this as a duplicate speech: <input type="checkbox"/></div>


When entering the ballots there is a toggle label “*Iron*” *speeches*. When set to «yes» this allows you to have the same speaker deliver multiple speeches provided their extra speeches are labelled on the form as “duplicates”. Typically, most tournaments require that lesser “iron man” speech is discarded from the tab, which would mean that you would mark the lower speaker of the two scores as the duplicate (note that this may require you to check each score’s average across a panel).

Speeches marked as duplicates are not included in the speaker tab. This means that they can also be used to exclude swing speakers from the tab as needed; even if they do not actually speak twice. To do so, change the name of the swing speaker to be that of an existing team member and ensure that that speech is marked as a duplicate.

Truco: There is also an option under **Standings** in the **Configuration** section that specifies the number of debates a speaker can miss before you will not show on the tab. By default there is no limit, but if need be this can be set to hide swing speakers from the final speaker tab.

21.3 Feedback entry

demo
Enter Results
Enter Feedback
View Draw

 **Add Feedback from Caltech 3**

Debate adjudicator
Jacqueline Freeman (Round 1)

Overall score
(0.0=lowest, 5.0=highest)
5

Did you agree with the decision?
Yes

Choose one word to describe this adjudicator.
Great

Do you think this adjudicator should be promoted?
☒

Do you think this adjudicator should be demoted?
☐

How well was this adjudicator's reasoning explained?

☐ 0
☐ 1
☐ 2
☐ 3
☐ 4
☐ 5
☐ 6
☐ 7
☒ 8
☐ 9
☐ 10

Feedback doesn't have the same verification process as ballots. Feedback that is entered by the tab room is assumed to be confirmed. If feedback is entered multiple times, all copies are retained but only the last one «counts» (is considered confirmed).

21.4 Online entry

There are two methods of allowing ballots and feedback to be submitted online. Both are set in the **Data Entry** page of each tournament's **Configuration** section and can be set independently; both in whether each can be submitted online at all and in which method of online submission are available.

21.4.1 Private URLs

The first method of data entry is using “private URLs”. When this setting is enabled you can create a special URL that is unique to a participant. This link contains a number of random characters and is not displayed publicly; it is in effect a secret that only that a specific participant should know. Presuming people do not share these links to others, this provides a means to (relatively) securely identify who is submitting what information. Because Tabbycat knows which participant has which URL it will only allow them to submit feedback/ballots for debates that they were speakers/adjudicators in.

Advertencia: Private URLs should provide more than adequate security for almost all tournaments’ purposes, but they aren’t foolproof. Anyone with access to the URL for a participant can submit feedback or ballots on their behalf, so it’s important that participants not share their URLs. This also means participants need to be careful when submitting from devices they do not own, because the URL will be logged in that device’s browser history.

These links must be generated within Tabbycat after the preference is enabled. To do so go to the **Feedback** section and then the **Private URLs** area. Once there you will be prompted to generate those URLs for all participants, which — once generated — will be presented in separate tables (one for teams; one for adjudicators).

Adjudicators		Speakers	
👤	URL	👤	URL
Allison Martin	http://localhost:8000/demo/privateurls/oofazleb/	Adrian May	http://localhost:8000/demo/privateurls/w2d97lma/
Angela Reed	http://localhost:8000/demo/privateurls/u2boi0rm/	Alma Spencer	http://localhost:8000/demo/privateurls/g3purjyw/
Bennie Rodriguez	http://localhost:8000/demo/privateurls/yuibhtbw/	Andy Lane	http://localhost:8000/demo/privateurls/8rwdnrza/
Billy Griffin	http://localhost:8000/demo/privateurls/j9fanwp5/	Angel Rhodes	http://localhost:8000/demo/privateurls/l1x973jt/

These URLs can then be distributed to each person in a number of ways. There are pages within Tabbycat for printing them out (one URL per page labelled by recipient) or emailing them out (providing participants have been assigned email addresses). In the past tournaments have also used data from this table to send out SMSs by bulk, or distributed them to institutional representatives to disburse.

Truco:

- You can assign email address to participants using the *importtournament command* when importing your registration data, or by going to the *Edit Data* area and looking up each Speaker/Adjudicator.
- If, after generating the private URLs, you add additional Teams or Adjudicators you can go to the *Edit Database* area, look up each Speaker/Adjudicator, and type in a bunch of random characters as their *Url key* to assign them a private URL.
- You can delete the current set of URLs by running this command in a shell on your server (replacing TOURNAMENT_SLUG with the appropriate value): `python manage.py privateurls delete --tournament TOURNAMENT_SLUG`

21.4.2 Public URLs

The second method of data entry is using “normal URLs”. This essentially means that any users visiting the public version of the site is able to submit a ballot or feedback (as specified by their respective settings). They do so by self-selecting which Team or Adjudicator they are then entering in a form as normal.

This is, rather obviously, not a particularly secure method of data entry — nothing is stopping anyone on the site from entering data as someone else. The data can be checked, verified, and edited as normal by admins however. As such, this method is only recommended for small tournaments where you can trust those present to enter accurate information (or where accuracy is not crucial).

Truco: There is an additional setting to set a “tournament password” that needs to be submitted to enable the form. It is imagined, that if enabled, this password would only be distributed to tournament participants. However this only helps (at best) prevent non-participants from entering information; the fundamental problem of not verifying who is submitting what information is still present.

The draw generator is quite flexible. You can specify a number of settings to suit different tournaments’ rules.

22.1 Summary of options

Options are set in the **Configuration** page as described in *starting a tournament*.

Option	Description	Allowable values
<i>Odd bracket resolution method</i>	How to resolve odd brackets	<ul style="list-style-type: none"> ■ Pull up from top ■ Pull up from bottom ■ Pull up from middle ■ Pull up at random <p>If sides are <i>Random</i> or <i>Balance</i>:</p> <ul style="list-style-type: none"> ■ Intermediate ■ Intermediate with bubble-up-bubble-down <p>If sides are <i>Pre-allocated</i>:</p> <ul style="list-style-type: none"> ■ Intermediate 1 ■ Intermediate 2
<i>Side allocations method</i>	How to allocate aff/neg	<ul style="list-style-type: none"> ■ Random ■ Balance ■ Pre-allocated ■ Manual ballot
<i>Pairing method</i>	How to pair teams within brackets	<ul style="list-style-type: none"> ■ Slide ■ Fold ■ Adjacent ■ Random
<i>Conflict avoidance method</i>	How to avoid history/institution conflicts	<ul style="list-style-type: none"> ■ Off ■ One-up-one-down
<i>Pullup restriction</i>	Whether and how to restrict pullups	<ul style="list-style-type: none"> ■ No restriction ■ Choose from teams who have been pulled up the fewest times so far

Prudencia: The valid options for intermediate brackets change depending on whether sides are pre-allocated, but these are **not** checked for validity. If you choose an invalid combination, Tabbycat will just crash. This won't corrupt the database, but it might be momentarily annoying.

22.2 The big picture

When generating a power-paired draw, Tabbycat goes through five steps:

1. First, it divides teams into «raw brackets», grouping them by the number of wins.
2. Second, it resolves odd brackets, applying the odd brackets rule to make sure there is an even number of teams in each bracket. This is often called «pulling up» teams.
3. Third, within each bracket, it pairs teams into debates using the pairing method.
4. Fourth, if enabled, it adjusts pairings to avoid history or institution conflicts.
5. Finally, it assigns sides to teams in each debate.

For each of these steps except the first, Tabbycat allows you to choose between a number of different methods.

22.3 Explanations of options

22.3.1 Odd bracket resolution

The **draw odd brackets** option specifies what you do when a bracket has an odd number of teams. (Obviously you have to do something, otherwise you can't pair off teams within the bracket.) There are two groups of methods: pull-up and intermediate brackets.

- **Pull-up methods** take one or more teams from the next bracket down, and move them into the odd bracket to fill the bracket.
- **Intermediate brackets** take the excess teams from the odd bracket and move them down into a new bracket, which sits between the odd bracket and the next one down (the «intermediate bracket»). It then takes teams from the next bracket down and moves them up to fill the new intermediate bracket.

The exact mechanics depend on whether or not sides are pre-allocated.

When sides are not pre-allocated

- **Pull-up methods:** Take a team from the next bracket down, and add them to the odd bracket to form an even bracket. You can choose to pull up the top team from the next bracket, or the bottom team, or the middle team, or a randomly chosen team. (If you pull up the middle team, and the bracket has an even number of teams, then it will choose randomly from the two middle teams.)
- **Intermediate brackets:** Take the bottom team from the odd bracket and match them against the top team from the next bracket. An intermediate bracket always has two teams.

If you're using conflict avoidance and intermediate brackets, you will probably want to use **Intermediate with bubble-up-bubble-down** instead. This uses the «bubble-up-bubble-down» rule to swap teams out of an intermediate bracket if there is a history or institution conflict. This is defined in the Australs constitution and is analogous to the «one-up-one-down» rule.

Prudencia: Using *Intermediate* with *One-up-one-down* does **not** imply *Intermediate with bubble-up-bubble-down*. You must enable *Intermediate with bubble-up-bubble-down* specifically.

When sides are pre-allocated

When sides are pre-allocated, an «odd bracket» is one that has an uneven number of affirmative and negative teams. (So odd brackets can have an even number of teams, *e.g.* 4 affs and 2 negs.)

- **Pull-up methods:** Take as many teams from the next bracket down as necessary to fill the bracket. If there aren't enough teams in the next bracket down, take teams from the bracket after that, and so on, until the (original) odd bracket is filled. Higher brackets are always filled first. You can choose to pull up the top teams from the next bracket, the bottom teams, or a random selection of teams.
- **Intermediate brackets:** Take the unpaired teams in a bracket, and move them down to a new intermediate bracket. Then, take the number of teams necessary from the opposite side, from the next bracket down, to fill the next bracket.

Intermediate 1 and **Intermediate 2** differ only in what happens if there aren't enough teams in the next bracket to fill the intermediate bracket. In **Intermediate 1**, it will just take teams from the bracket after that, and so on,

until the intermediate bracket is filled. In **Intermediate 2**, it will split the intermediate bracket: the teams that can be paired with the next bracket form the first intermediate bracket, and then the teams that aren't form a new (unfilled) intermediate bracket, to be filled from teams from the bracket after that. This keeps going, splitting into as many intermediate brackets as necessary, until all excess teams from the original odd bracket are paired.

22.3.2 Side allocations

There are four methods:

- **Random** allocates randomly. Some tournaments might like this, but most will probably want to use Balance, because Random doesn't guarantee that a team won't be (say) affirming the entire tournament.
- **Balance** assigns the team that has affirmed less so far the affirmative side (and, therefore, the team that has negated less the negative side). If both teams have affirmed the same number of times, it assigns sides randomly.
- **Preallocated** is used for pre-allocated sides. If used, you must enter data for pre-allocated sides into the database, as specified below.
- **Manually enter from ballot** is used for tournaments where the sides of the teams involved are not assigned in advance, but are instead determined by the teams themselves

Pre-allocated sides

There isn't currently any way to edit side allocations from the front end. To do so from the back end, you need to create one `TeamPositionAllocation` entry for each team in each round. All teams must have an allocation for every round. There are a few ways to do this, take your pick:

- If you're using the *importtournament command*, it reads sides from the file `sides.csv`.
- You can do this from the Django admin interface (under Setup > Edit Database) by going to the relevant team and adding a **team position allocation** entry. That is:
 1. Click **Admin** on the bottom right of any page after logging into an account with *superuser access*.
 2. Next to **Teams**, click **Change**.
 3. Click on the name of the team you want to edit side allocations for.
 4. Add or edit the entry or entries in the **Team position allocations** table at the bottom.
- You can also do this by writing a script that creates `TeamPositionAllocation` objects and saves them. Have a look at [draw/management/commands/generatesideallocations.py](#) for an example.

22.3.3 Pairing method

It's easiest to describe these by example, using a ten-team bracket:

- **Fold**: 1 vs 10, 2 vs 9, 3 vs 8, 4 vs 7, 5 vs 6. (Also known as high-low pairing.)
- **Slide**: 1 vs 6, 2 vs 7, 3 vs 8, 4 vs 9, 5 vs 10.
- **Adjacent**: 1 vs 2, 3 vs 4, 5 vs 6, 7 vs 8, 9 vs 10. (Also known as high-high pairing.)
- **Random**: paired at random within bracket.

Teams are always paired within their brackets, after resolving odd brackets.

22.3.4 Conflict avoidance method

A **conflict** is when two teams would face each other that have seen each other before, or are from the same institutions. Some tournaments have a preference against allowing this if it's avoidable within certain limits. The **draw avoid conflicts** option allows you to specify how.

You can turn this off by using **Off**. Other than this, there is currently one conflict avoidance method implemented.

One-up-one-down is the method specified in the Australs constitution. Broadly speaking, if there is a debate with a conflict:

- It tries to swap teams with the debate «one up» from it in the draw.
- If that doesn't work, it tries to swap teams with the debate «one down» from it in the draw.
- If neither of those works, it accepts the original conflicted debate.

It's a bit more complicated than that, for two reasons:

- History conflicts are prioritised over (*i.e.*, «worse than») institution conflicts. So it's fine to resolve a history conflict by creating an institution conflict, but not the vice versa.
- Each swap obviously affects the debates around it, so it's not legal to have two adjacent swaps. (Otherwise, in theory, a team could «one down» all the way to the bottom of the draw!) So there is an optimization algorithm that finds the best combination of swaps, *i.e.* the one that minimises conflict, and if there are two profiles that have the same least conflict, then it chooses the one with fewer swaps.

Nota: Teams imported without an institutional affiliation are (for conflict avoidance purposes) considered to all be from the same institution and will trigger conflicts as described above. If this is a concern it can be assigning “fake” institutions (*i.e.* *Swing 1*) to each unaffiliated team.

22.3.5 Pullup restriction

You can restrict which teams can be pulled up, by configuring the draw generator to choose a pullup team from among only those teams who have been pulled up the fewest times in rounds preceding the current round. Most of the time, this is equivalent to saying that a team cannot be pulled up more than once. The difference is that if *all* teams in a bracket have been pulled up at least once, it then chooses from among teams who have been pulled up *only* once (if any), and so on.

Pullup restrictions only apply when the *odd bracket resolution method* is a pullup method. They have no effect on intermediate brackets.

22.4 What do I do if the draw looks wrong?

You can edit match-ups directly from the draw page. Functionally, you can do anything you want. Of course, operationally, you should only edit the draw when you *know* that the draw algorithm got something wrong. If you need to do this, even just once, please file a bug report by creating a new issue on [our issues page on GitHub](#).

Draw Generation (BP)

The draw generator for British Parliamentary tournaments tries to rotate teams through positions by assigning them positions they've been in less often before the current round.

23.1 Summary of options

Options are set in the **Configuration** page as described in *starting a tournament*. Options in *italics* with an asterisk are not WUDC-compliant. The recommended options are shown in **bold**.

Option	Description	Allowable values
<i>Pullup distribution</i>	Where pullup teams get placed	<ul style="list-style-type: none"> ▪ Anywhere in bracket ▪ <i>All in the same room*</i>
<i>Position cost</i>	Which cost function to use to indicate which position profiles are preferred	<ul style="list-style-type: none"> ▪ Simple ▪ Rényi entropy ▪ Population variance
<i>Rényi order</i>	Order of Rényi entropy	Any non-negative number (default: 1 , <i>i.e.</i> Shannon entropy)
<i>Position cost exponent</i>	Degree to which large position imbalances should be prioritised	Any non-negative number (default: 4)
<i>Assignment method</i>	Algorithm used to assign positions	<ul style="list-style-type: none"> ▪ <i>Hungarian*</i> ▪ Hungarian with preshuffling

23.2 The big picture

To try to achieve position balance, Tabbycat treats the allocation of teams to debates as an [assignment problem](#). That is, it computes the «cost» of assigning each team to each position in each debate, and finds an assignment of all teams to a position in a debate that minimises the total cost (the sum over all teams).

23.2.1 A simple example

Here's a small example, to illustrate the idea. Say you have a tournament with 16 teams, and you're about to draw round 4. There are sixteen «places» in the draw: four positions in each of four rooms. Tabbycat calculates the «cost» of putting each team in each place, and puts them in a matrix, like this:

Room	Top				Second				Third				Bottom			
Position	OG	OO	CG	CO	OG	OO	CG	CO	OG	OO	CG	CO	OG	OO	CG	CO
A (8)	16	16	16	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
B (7)	16	0	16	16	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
C (7)	16	16	0	16	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
D (6)	16	0	16	16	16	0	16	16	∞	∞	∞	∞	∞	∞	∞	∞
E (6)	0	16	16	16	0	16	16	16	∞	∞	∞	∞	∞	∞	∞	∞
F (6)	16	16	0	16	16	16	0	16	∞	∞	∞	∞	∞	∞	∞	∞
G (5)	∞	∞	∞	∞	16	0	16	16	∞	∞	∞	∞	∞	∞	∞	∞
H (5)	∞	∞	∞	∞	16	0	16	16	∞	∞	∞	∞	∞	∞	∞	∞
I (4)	∞	∞	∞	∞	∞	∞	∞	∞	16	16	0	16	∞	∞	∞	∞
J (4)	∞	∞	∞	∞	∞	∞	∞	∞	16	16	16	0	∞	∞	∞	∞
K (3)	∞	∞	∞	∞	∞	∞	∞	∞	0	16	16	16	0	16	16	16
L (3)	∞	∞	∞	∞	∞	∞	∞	∞	16	16	0	16	16	16	0	16
M (3)	∞	∞	∞	∞	∞	∞	∞	∞	16	16	16	0	16	16	16	0
N (3)	∞	∞	∞	∞	∞	∞	∞	∞	0	16	16	16	0	16	16	16
O (1)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	16	16	16	0
P (1)	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	0	16	16	16

Each «16» is the cost of putting a team in a position it's seen once; each «0» is the cost of putting a team in the position it hasn't. (Details of how this is calculated are [below](#).) For example, team A (on 8 points) has been in every position except CO. The ∞'s indicate places where the team isn't allowed to go, because the room isn't in their bracket. For

example, the three teams on 6 points (D, E, F) can go in either the top or second room, because any of them can be the pullup team.

The algorithm then chooses entries so that one is selected from each row and one is selected from each column, in a way that minimises the sum of the selected entries. In this case, the selected entries are highlighted in blue. For example, the top room comprises teams E (OG), B (OO), C (CG) and A (CO).

Sometimes, particularly in round 4, it simply isn't possible to «satisfy» everyone. For example, among the top eight teams, five haven't been in OO, but only two can be accommodated within those brackets. In this case, teams B and G got lucky; there are also many other draws that would have incurred the same total cost.

More generally, in most cases, there will be many optimal solutions. To randomise the selection among them, Tabbycat (under default settings) randomly permutes the rows and columns of the matrix before starting the assignment algorithm.

23.3 Explanations of options

23.3.1 Pullup distribution

If the number of teams in a bracket is not a multiple of four, it pulls up teams from the next bracket down. The pullup distribution then governs how those teams are paired into the upper bracket.

The available options are as follows:

- **Anywhere in bracket:** The pullup teams are treated as if they were any other team in their new bracket. For example, if there are 17 teams in a 10-point bracket, then the three 9-point teams that get pulled up may be paired anywhere in the 10-point bracket, independently of each other. Chance might put them in the same room, but more likely, they will not all be in the same room, so there will be multiple pullup rooms in the 10-point bracket.
- **All in the same room:** All of the pullup teams will be paired into the same room. This means that there will be at most one pullup room per bracket, effectively creating an «intermediate bracket».

Nota: While it can be argued that the *All in the same room* setting is fairer, it is prohibited by the WUDC constitution. If your tournament follows WUDC rules, you cannot use this setting.

The teams that get pulled up aren't specifically chosen—they're just assigned as part of the algorithm described [above](#), which optimises for position balance. Tabbycat doesn't support taking anything else into account when choosing pullup teams. (WUDC rules wouldn't allow it, either.)

23.3.2 Position cost options

The *position cost function* is a function that indicates how «bad» it would be if a team were to be allocated a certain position (OG, OO, CG, CO) in a debate. When generating a draw, Tabbycat chooses from among the draws that minimise the sum of the position costs for each team.

More formally:

- A *position history* or just *history* \mathbf{h} is a 4-tuple where each element is the number of times a team has already been in the corresponding position. For example, $\mathbf{h} = (0, 2, 1, 1)$ means that a team has been in OO twice, CG and CO once each, and hasn't been in OG.
- A cost function $C(\mathbf{h}, s)$ is a function specifying how «bad» it would be if a team with position history \mathbf{h} were assigned the position s in the next round.

Tabbycat allows you to choose from a number of different **position cost functions**, as well as a **position cost exponent** β . Then, when allocating teams to debates, Tabbycat allocates teams to positions $(s_t, t \in \mathcal{T})$ to minimise

$$\sum_{t \in \mathcal{T}} [C(\mathbf{h}_t, s_t)]^\beta$$

where \mathcal{T} is the set of all teams, \mathbf{h}_t is the position history of team t and s_t is the position to which team t would be allocated.

Position cost exponent

The **position cost exponent** β controls how different teams trade off with each other.

- The **larger** β is, the more concerned it is with preventing *very* bad situations. That is, it will give more teams some slight unevenness in order to prevent one team from getting a *very* uneven history.
- The **smaller** β is, the more concerned it is with preventing *any* unevenness. That is, it will try to keep more teams from being uneven *at all*, at the cost of possibly letting just one team get a very uneven history.
- At the large extreme, as $\beta \rightarrow \infty$, it will do everything it can to minimise the plight of the *worst-off* team, and it won't care for *any* team other than the worst-off.
- At the small extreme, as $\beta \rightarrow 0$, it will do everything it can to minimise the number of teams with a non-optimal profile—but if it's impossible to protect a team from sub-optimality, it won't care *how* uneven the unlucky team gets.

The «balanced» approach would be $\beta = 1$, which just takes the cost function as-is. This doesn't mean that this is the best idea, however—you'd typically want to bias towards preventing very uneven histories a bit more. Most tournaments will probably want β to be somewhere between 2 and 5. (Note that β need not be an integer.)

Position cost functions

Tabbycat allows you to choose between three position cost functions $C(\mathbf{h}, s)$: **Simple**, **Rényi entropy** and **Population variance**.

In the descriptions that follow, $\mathcal{S} = \{\text{OG}, \text{OO}, \text{CG}, \text{CO}\}$, the set of all BP positions.

Simple

The simple cost function $C_{\text{simple}}(\mathbf{h}, s)$ returns the number of times the team has already been in position s , less the number of times the team has been in its least frequent position. That is,

$$C_{\text{simple}}(\mathbf{h}, s) = \mathbf{h}[s] - \min_{s' \in \mathcal{S}} \mathbf{h}[s']$$

where $\mathbf{h}[s]$ is the element of \mathbf{h} corresponding to position s .

Rényi entropy

Informally speaking, the **Rényi entropy** is a measure of the diversity of the positions in a team's history. A history consisting only of one position has *low* entropy, while a history that is perfectly evenly distributed has *high* entropy. The **Rényi entropy cost function** reverses this intuition, so that an even hypothetical history has low cost, while an uneven hypothetical history has high cost.

The Rényi entropy takes one parameter, known as its *order*, α , which will be further discussed below.

More formally, the Rényi entropy cost function $C_{\text{Rényi}}(\mathbf{h}, s)$ is defined as

$$C_{\text{Rényi}}(\mathbf{h}, s) = n_{\mathbf{h}}[2 - H_{\alpha}(\hat{p}_{\mathbf{h},s})]$$

where

- $n_{\mathbf{h}} = \sum_{s'} \mathbf{h}[s']$ is the number of rounds the team has competed in so far.
- $\hat{p}_{\mathbf{h},s}$ is the *normalised hypothetical* position history that would arise if a team with history \mathbf{h} were to be allocated position s in the next round; that is,

$$\hat{p}_{\mathbf{h},s}[s'] = \begin{cases} \frac{1}{n_{\mathbf{h}}+1}(\mathbf{h}[s'] + 1), & \text{if } s = s', \\ \frac{1}{n_{\mathbf{h}}+1}\mathbf{h}[s'], & \text{if } s \neq s'. \end{cases}$$

Note that $\hat{p}_{\mathbf{h},s}$ is a probability distribution (that is, its elements sum to 1).

- $H_{\alpha}(\cdot)$ is the *Rényi entropy* of order α of a probability distribution, defined as

$$H_{\alpha}(p) = \frac{1}{1-\alpha} \log_2 \left(\sum_{s \in \mathcal{S}} (p[s])^{\alpha} \right), \quad \alpha \neq 1.$$

In the special (limiting) case where $\alpha = 1$, it reduces to the *Shannon entropy*,

$$H_1(p) = - \sum_{s \in \mathcal{S}} p[s] \log_2 p[s].$$

Note that for all α , $0 \leq H_{\alpha}(p) \leq \log_2(4) = 2$ (since there are four positions in BP).

The **Rényi order** is the parameter α above, and it controls *what it means to be «even among positions»* for a team. Note that «evenness» is not easily defined. After round 8, which position history is more even: (0, 2, 3, 3) or (1, 1, 1, 5)? The Rényi order allows us to tune this definition.

- The **smaller** α is, the more it cares that teams compete in every position *at least* once, favouring (1, 1, 1, 5) over (0, 2, 3, 3): it's worse to have never Oged, than it is to have COed five times.
- The **larger** α is, the more it cares that teams do not compete in *any* (one) position too many times, favouring (0, 2, 3, 3) over (1, 1, 1, 5): it's worse to have COed five times, than it is to have never Oged.
- At the small extreme, as $\alpha \rightarrow 0$, it *only* counts how many positions a team has seen at least once, and doesn't care about the distribution among them so long as a team has been in each position once.
- At the large extreme, as $\alpha \rightarrow \infty$, it *only* looks at how many times each team has seen its *most frequent* position, and tries to keep this number even among all teams.

The «balanced» approach would be $\alpha = 1$ (the *Shannon entropy*), though of course it's arguable what «balanced» means. Tabbycat defaults to this value.

To give some intuition for the useful range: In round 9, a strict ordering by number of positions seen at least once occurs for approximately $\alpha < 0.742$. A strict ordering by number of times in the most frequent position occurs for $\alpha > 3$. Changing α outside the range $[0.742, 3]$ will still affect the relative (cardinal) weighting *between teams*, but will not affect the *ordinal* ranking of possible histories.

The purpose of weighting costs by $n_{\mathbf{h}}$ is to prioritise those teams who have competed in every round over those who have competed in fewer rounds.

Population variance

The **population variance** cost function is just the population variance of the history 4-tuple,

$$C_{\text{popvar}}(\mathbf{h}, s) = \frac{1}{4} \sum_{s' \in \mathcal{S}} \left(\hat{\mathbf{h}}_s[s'] - \mu_{\hat{\mathbf{h}}_s} \right)^2,$$

where $\hat{\mathbf{h}}_s$ is the *hypothetical* position history that would arise if a team with history \mathbf{h} were to be allocated position s in the next round; that is,

$$\hat{\mathbf{h}}_s[s'] = \begin{cases} \mathbf{h}[s'] + 1, & \text{if } s = s', \\ \mathbf{h}[s'], & \text{if } s \neq s'. \end{cases}$$

and where $\mu_{\hat{\mathbf{h}}_s}$ is the mean of $\hat{\mathbf{h}}_s$,

$$\mu_{\hat{\mathbf{h}}_s} = \frac{1}{4} \sum_{s' \in \mathcal{S}} \hat{\mathbf{h}}_s[s'].$$

At the extremes, a team that has seen all positions evenly will have a population variance of zero, while a team that has seen just one position n times will have a population variance of $\frac{3n^2}{16}$.

23.3.3 Assignment method

Tabbycat uses the [Hungarian algorithm](#) to solve the [assignment problem](#) of assigning teams to positions in debates. This can be run with or without preshuffling:

- **Hungarian algorithm** just runs the Hungarian algorithm as-is, with no randomness. This probably isn't what you want.
- **Hungarian algorithm with preshuffling** also runs the Hungarian algorithm on the position cost matrix, but shuffles the input so that the draw is randomised, subject to having optimal position allocations.

Preshuffling doesn't compromise the optimality of position allocations: It simply shuffles the order in which teams and debates appear in the input to the algorithm, by randomly permuting the rows and columns of the position cost matrix. The Hungarian algorithm still guarantees an optimal position assignment, according to the chosen position cost function.

Nota: Running the Hungarian algorithm *without* preshuffling has the side effect of grouping teams with similar speaker scores in to the same room, and is therefore prohibited by WUDC rules. Its inclusion as an option is mainly academic; most tournaments will not want to use it in practice.

No other assignment methods are currently supported. For example, Tabbycat can't run fold (high-low) or adjacent (high-high) pairing *within* brackets.

Preformed Panels

Preformed panels, also known as a “shadow draw”, allow for adjudicator panels to be created *before* a round has been drawn and then applied once its draw is ready. This means that the task of panel formation can be largely performed during periods that sit outside of the normal time pressure that comes with trying to finalise a draw for release. This process can save significant amounts of time at large tournaments, or at tournaments where the adjudication core wants to very carefully control the specific combination of adjudicators within panels.

Tabbycat’s implementation of preformed panels is more powerful, but less intuitive, than many other implementations. The chief difference is that our workflow does not simply transpose a linear set of preformed panels atop a draw. Instead we employ Tabbycat’s existing allocation tools, primarily the notion of a debate’s *priority*, to allow for a non-linear matching of preformed panels that will avoid creating conflicts and can better adapt to a given draw and — particularly when the most important debates do not strictly follow the highest debate brackets.

The central concept here is that each preformed panel has a specific priority value. When applying preformed panels to a draw, the allocator ties to best match the priority value of each preformed panel to the priority of each actual debate. This is approximately equivalent how Tabbycat’s normal auto-allocator matches the strength of each potential panel (as measured by adjudicator’s ratings) to the priority (or as a fallback: the bracket) of each debate.

24.1 Step 1: Creating Preformed Panels

The link to the preformed panels section is available under the **Setup** menu. Preformed panels are formed for specific rounds, and this page links to all of the rounds available in your tournament. The page for creating preformed panels for a specific round is essentially the same as that of the normal adjudicator allocation.

Nota: The **Draw** page of each individual round also contains direct linked to the preformed panels of that specific round and the next round.

Initially, the preformed panels page will have no panels available. The **Create Panels** button in the top-left will let you make some. Note that the panels it creates are based upon a projection of that round’s general results using the results of the previous round. As a result, each preformed panel will have a bracket-range and a liveness range.

<

No

Create Panels

Prioritise

Allocate

?

Seen

Institution

Conflict

Missing

Unavailable

👁 Break

👁 Gender

👁 Rank

👁 Region

The bracket range of the hypothetical debate

11-12	4	<div></div>	N/A	👁			ⓘ
9-10	4	<div></div>	N/A	👁			ⓘ
8	4	<div></div>	N/A	👁			ⓘ
8-9	4	<div></div>	N/A	👁			ⓘ
8-9	4	<div></div>	N/A	👁			ⓘ

Sort By Drag Order

Sort By Name

Sort By Score

Show Available (77)

Show All (80)

4.0

Camille L

DORWINION

1.0

Chukwubuike I

DORWINION

3.0

Krešimir M

IRON HILLS

2.0

Davi C

CARAS GALADHON

3.0

Diego B

EREBOR

3.0

Merry B

UPBOURN

3.0

Gabriel C

ITHILLEN

1.0

Bella H

SOUTHERN MIRKWOOD

3.0

Jiří M

RI ROHAN

3.0

Neven B

UPBOURN

3.0

Artin M

EDORAS

3.0

Khaza D

SOUTHERN MIRKWOOD

2.0

Oliver M

STADDLE

4.0

Ruairidh K

IRON HILLS

3.0

Fareeq S

MITHLOD

1.0

Elmurza C

4.0

Maria A

2.0

Nail D

1.0

Rós G

4.0

Ellen K

2.0

Kim G

2.0

Lian T

4.0

Hoàn Đ

Nota: Like the normal adjudicator allocation interface, the preformed panel interfaces will indicate when an adjudicator has not been marked as available. If using preformed panels, you may want to set adjudicator availability earlier than you would otherwise.

24.2 Step 2: Assign Priorities to Preformed Panels

By default the priority slider for all preformed panels is in the neutral position. You can use the «Prioritise» button in the top left to assign each preformed panel an priority value automatically based upon their brackets or liveness. Before or after this step you can alter the priorities as normal — even after you have allocated adjudicators.

It is, important, to remember to assign a range of priorities to the panels. Without distinct priority values, the application of your preformed panels to the actual draw will be essentially random. If allocating priorities manually, it is a good idea to keep a relatively even distribution of preformed panel priorities — use the range!

Nota: In Round 1, each debate has a liveness and bracket of 0. If you are using preformed panels in this instance you may need to manually-differentiate their priorities.

24.3 Step 3: Allocate Adjudicators to Preformed Panels

< No Create Panels Prioritise Allocate
 Seen Institution Conflict Missing Unavailable
Break Gender Rank Region

Successfully auto-allocated adjudicators to preformed panels. ×

Successfully auto-prioritised preformed panels. ×

11-12	4		3.8 4.5	5.0 Sonny B PELARGIR	4.0 Semere A PELARGIR	3.0 Zac G UG LEBENNIN	3.0 Lian T UG RINGLO VALE	①
9-10	4		4.0 4.5	5.0 Facino L ISENGARD	4.0 Marina B IRON HILL 3 ago	4.0 Camille L DORWINION	3.0 Diego B EREBOI 3 ago	①
8	4		4.7 5.0	5.0 Marvin L UG ANFALAS	5.0 Jana F UG ANFALAS	4.0 Czesława K DOL AMROTH		①
8-9	4		4.3 5.0	5.0 Urszula G IRON HILLS 2 ago	5.0 Loránd H ISENGARD	3.0 Park P UG LEBE 2 ago		①
8-9	4		4.0 4.5	5.0 Lea B ITHILIN 2 ago	4.0 Thindra J ARNOR	4.0 Klara S UG RINGLO VALE	3.0 Cintio A UG LOSSAR 2 ago	①
7	4		3.7 4.0	4.0 Uchechu... EREBOI	4.0 Franco M RIVENDELL	3.0 Crispus U EOTHEOD		①

Now that your panels have an priority, you can begin allocating adjudicators. This can be done entirely manually; however note that the normal “auto” allocator also functions in this context. Even if you want to tweak your panels extensively, the auto-allocator can provide a good first-pass collection of panels because it will give stronger adjudicators to the panels that you have marked as important. It will also avoid creating conflicts in forming its panels.

The created panels all autosave, so you can leave the page as needed. Like the main allocation interface, changes should appear “live” across different computers and the sharding system is available to divide up each person’s view of the draw.

24.4 Step 4: Create the Draw

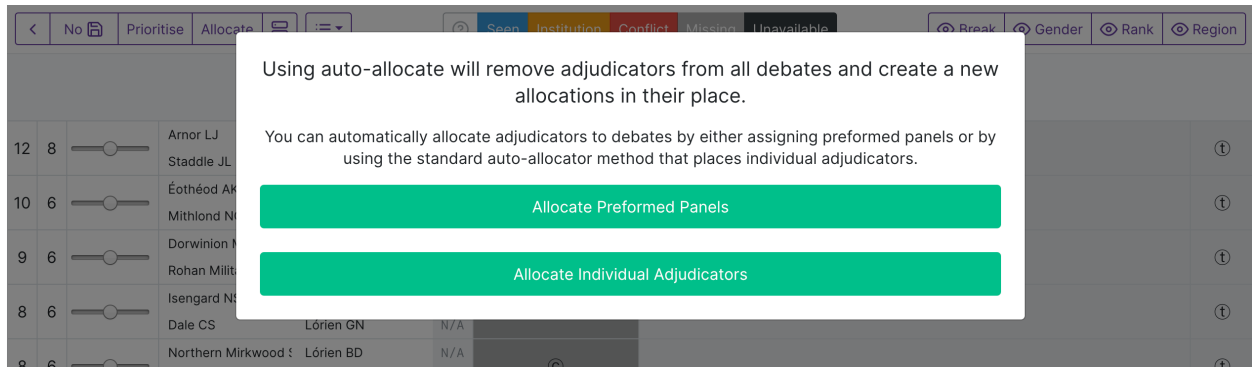
Proceed with the creation of the draw as per normal. Open up the normal adjudicator allocation page for that round.

24.5 Step 5: Assign Priorities to the Debates

When allocating preformed panels, the system uses priority is the interface between the preformed panels and the actual debates. It is thus crucial that you assign priorities to the debates in the actual draw using automatic prioritisation or the manual sliders. Because the automatic prioritiser does not employ the highest priority value, it is worth having a look at the draw and seeing if any debates justify this before proceeding.

24.6 Step 6: Allocate Preformed Panels to Debates

To allocate preformed panels to your debates you click the “normal” Allocate button and then select the *Preformed Panels* option.



This will then automatically apply those preformed panels.

Successfully auto-allocated preformed panels to debates.

Debate	Adjudicator	Score	Panel	Score	Panel	Score	Panel	Score	Panel
12 8	Arnor LJ	3.8	Upbourn GB	3.8	5.0 Sonny B	4.0 Semere A	3.0 Zac G	3.0 Lian T	
	Staddle JL	4.5	Southern Mirk	4.5					
10 6	Éothéod AK	4.0	Mithlond SR	4.0	5.0 Lea B	4.0 Klara S	4.0 Thindra J	3.0 Cintio A	
	Mithlond NC	4.5	Bree HC	4.5					
9 6	Dorwinion MI	4.3	Erebor BP	4.3	5.0 Urszula G	5.0 Loránd H	3.0 Park P		
	Rohan Military	5.0	Dale TM	5.0					
8 6	Isengard NS	5.0	Upbourn VS	5.0	5.0 Marvin L	5.0 Jana F			
	Dale CS	5.0	Lórien GN	5.0					
8 6	Northern Mirkwood	4.3	Lórien BD	4.3	5.0 Lan L	5.0 Yesenia A	3.0 Khaza D		
	Iron Hills AD	5.0	Ithilien CM	5.0					
8 4	Dol Amroth CW	4.0	UG Ringló Vale PN	4.0	5.0 Facino L	4.0 Marina B	3.0 Diego B	4.0 Camille L	
	Southern Mirkwood I	4.5	Staddle RB	4.5					
7 5	Ri Rohan MM	3.7	Rivendell BA	3.7	4.0 Ruairidh K	3.0 Caitlin I	4.0 Isak J		
	Caras Galadion BL	4.0	Edoras SW	4.0					
7 6	Pelargir GS	3.7	Caras Galadion LD	3.7	4.0 Eleanor W	4.0 Sofia R	3.0 Kathrin F		
	Rivendell CL	4.0	Minas Tirith BV	4.0					
7 8	Dorwinion SS	3.7	UG Anfalas BL	3.7	4.0 Tea van d...	3.0 Boubker E	4.0 Hana H		
	Lórien SB	4.0	Bree MN	4.0					
7 5	Minas Tirith EN	3.7	Arnor II	3.7	4.0 Rashidah B	4.0 Rô N	3.0 Olivia L		
	Arnor AC	4.0	Iron Hills HS	4.0					
6 5	Erebor OT	3.7	Edoras NB	3.7	4.0 Maria A	4.0 Ellen K	3.0 Fareeq S		
	Iron Hills RM	4.0	Ithilien TN	4.0					
6 6	Rivendell DD	4.0	Rohan Military	4.0	5.0 Laila F	3.0 Masa F	4.0 Kosisochukwu C		
	Rivendell IS	4.5	Hobbiton LW	4.5					
6 6	UG Lossarnaci	3.8	UG Ringló Vale RM	3.8	4.0 Uchechu...	4.0 Franco M	3.0 Crispus U	4.0 Czesława K	
	Minas Tirith CB	4.0	Minas Tirith HC	4.0					
5 6	The Shire NE	3.0	Iron Hills LE	3.0	4.0 Theodor M	3.0 Evelyn P	2.0 Tim B		
	Dorwinion DG	3.5	Northern Mirkwood I	3.5					

Sort By Drag Order Sort By Name Sort By Score

2.0 Abraham W 2.0 Anzor D 3.0 Addison M

You can the edit the allocation as normal. If needed, you can redo the allocation of the preformed panels at any point.

Printing Ballots and Feedback

Tabbycat offers the ability to print scoresheets and feedback sheets on a per-round basis. This means that each scoresheet is customised to match the draw, and so will display:

- The motion or possible motions that could have been chosen from
- The teams, and a list of speakers for each team
- The room number, and name of the adjudicators present

Feedback sheets will also be customised in this manner; displaying:

- A “from” source. Sheet will be printed for each possible piece of feedback in that room: ie adjudicators will each get a sheet for their panellists and trainees while teams and panellists will get sheets for the chair.
- The specific questions you established when configuring the *adjudicator feedback questions*
- The room number

To print the ballots for each round, you would go to the **Display** page for that round, then click one of the two relevant print links. This opens a new window. In each window, you then use your browser’s print function (CTRL-P). In the resulting print interface (from your browser) you’ll want to turn background graphics on, header/footers off, and set the layout to portrait (feedback sheets) or landscape (score sheets).

Typically you’d save these to PDF for printing, although you should be able to print straight from this dialogue box.

Prudencia: Printing works best in Safari and Firefox; use those if possible.

Sending Notifications

Tabbycat offers integrations with email delivery services to send notifications to participants on certain enumerated events. For sending these emails, [SendGrid](#) is readily available as an add-on in Heroku. It may be necessary to install the [SendGrid add-on](#) manually. Other integrations may also be used in its place by changing the relevant environment variables.

26.1 Events

Tabbycat includes a number of templated notifications that can be sent in various times. Variables which are included between curly brackets which are substituted for personalized information passed by email. Links to email will redirect to a page where the message can be changed and the participants selected.

All emails have the `{{ USER }}` and `{{ TOURN }}` variables to indicate who the email is sent to, and the tournament it relates to. The «From» in the emails will also be the tournament's name.

Email content and description	Variables
Adjudicator draw notification Email to adjudicators indicating their room assignment. Available through the admin draw page.	<ul style="list-style-type: none"> ■ {{ ROUND }}: The round name ■ {{ VENUE }}: The venue of the assigned debate ■ {{ PANEL }}: A list of all the adjudicators assigned to the venue (with positions) ■ {{ DRAW }}: A list of the team matchup with their roles ■ {{ POSITION }}: The target adjudicator's position in the panel ■ {{ URL }}: A link to the adjudicator's private URL page
Team draw notification Email to teams indicating their pairing. Available through the admin draw page.	<ul style="list-style-type: none"> ■ {{ ROUND }}: The round name ■ {{ VENUE }}: The venue of the assigned debate ■ {{ PANEL }}: A list of all the adjudicators assigned to the venue (with positions) ■ {{ DRAW }}: A list of the team matchup with their roles ■ {{ TEAM }}: The team's code or short name ■ {{ SIDE }}: The team's side
Private URL distribution Email to participants giving them their private URL for electronic forms. Available through the private URLs page.	<ul style="list-style-type: none"> ■ {{ URL }}: The personalized URL ■ {{ KEY }}: The private code in the URL
Ballot submission receipt Email to adjudicators of their ballot after tabroom confirmation. Sent automatically when their ballot's result status becomes confirmed, if enabled in the «Notifications» section of the tournament options.	<ul style="list-style-type: none"> ■ {{ DEBATE }}: The name (with round & venue) of the relevant debate ■ {{ SCORES }}: The submitted ballot with speaker scores and team names
Current team standings Email to speakers with their point total. Available through the «Confirm Round Completion» page.	<ul style="list-style-type: none"> ■ {{ URL }}: The URL of the team standings page (if public) ■ {{ TEAM }}: The team's name ■ {{ POINTS }}: The team's number of points
Motion release Email to speakers with the motion(s) of the current round. Available through the admin draw page.	<ul style="list-style-type: none"> ■ {{ ROUND }}: The name of the round ■ {{ MOTIONS }}: A list of the motions released
Team information Email to speakers with information pertaining to their team, such as eligibility and codes. Available through the Participants page.	<ul style="list-style-type: none"> ■ {{ SHORT }}: The team's short name ■ {{ LONG }}: The team's long name ■ {{ CODE }}: The team's code name ■ {{ EMOJI }}: The team's assigned emoji ■ {{ DIVISION }}: The team's division if applicable ■ {{ BREAK }}: Break categories which the team is a member ■ {{ SPEAKERS }}: A list of the speakers in the team ■ {{ INSTITUTION }}: The team's affiliation
144	<div>Capítulo 26. Sending Notifications</div>

26.2 Event Webhook

With SendGrid, the status of sent emails can be sent to Tabbycat to be displayed, giving an indication of failures and whether participants have opened the messages. To activate this feature, setup must be done both in your SendGrid account and in Tabbycat.

1. Set a secret key in the Email section of the tournament's preferences. This key must be alphanumeric without any spaces.
2. Copy the «web-hook» link presented in a box at the top of the «email participants» page.
3. Go to https://app.sendgrid.com/settings/mail_settings and select «Event Notifications»
4. Enable the feature and paste the Tabbycat URL under «HTTP POST URL».
5. Select the notifications to keep track (or all of them).

Prudencia: Each email and change in status sent to Tabbycat will add a row to the database. If the number of rows is limited, as is for free Heroku apps, enabling the webhook may use up a significant number of rows. Be selective in the events to keep track.

27.1 Team standings rules

In Tabbycat, you can choose how teams are ranked in the team standings. For example, at Australs, teams are ranked first on the number of wins, and second on their total speaker score. The setting that specifies how teams are ranked is called the **team standings precedence**. The team standings precedence is used:

- When displaying the team tab,
- Whenever a power-paired draw is generated, and
- When computing which teams are in the break.

When you choose the team standings precedence, you choose from a list of *metrics*. Then, in the standings, teams will be sorted first by the first metric, then by the second metric, and so on. You must choose at least one metric, and you can choose up to eight. Teams tied on all metrics will have the same rank.

If you like, you can also choose **team standings extra metrics**, which are metrics that will be shown in the team standings, but not used to rank teams.

Metric	Description
Wins	How many debates the team has won.
Points	How many points the team has. For two-team formats, this is just a synonym for wins, and differs only in column labelling. For BP, this is 3 points for a first, 2 for a second, 1 for a third and 0 for a fourth.
Points (2/1/0)	How many points the team has, where teams earn 2 points for a win, 1 point for a loss and 0 points for a forfeit.
Total speaker score	The sum of all speaker scores attained in all debates.
Average total speaker score	The average total speaker score over all debates the team has had, not counting debates where they or their opponents forfeited.
Average individual speaker score	The total substantive speaker score, over all debates the team has had and the number of speakers. Provides an equivalent metric to average total speaker score in no-reply formats, but within the substantive speech scoring range.
Speaker score standard deviation	The standard deviation of total speaker scores over all debates the team has had, not counting debates where they or their opponents forfeited. This metric is ranked in ascending order (smaller standard deviations ranked higher).
Sum of margins	The sum of all margins. Wins are positive, losses are negative.
Average margin	The average margin over all debates the team has had, not counting debates where they or their opponents forfeited.
Draw strength	The sum of the number of wins of every team this team has faced so far. This is also known in some circuits as <i>win points</i> , <i>opp wins</i> or <i>opp strength</i> .
Votes/ballots carried	The number of adjudicators that gave this team a win across all of their debates. Also known as the number of <i>ballots</i> or <i>judges</i> a team has. In cases where the panel is smaller or larger than 3, this number is normalised to be out of 3. For example, if a panel of five splits 3–2, then the winning team is recorded as gaining 1.8 votes, and the losing team is recorded as gaining 1.2. This also means that solo adjudicators are always worth three votes.
Number of firsts	The number of debates in which the team came first. Only makes sense for British Parliamentary.
Number of seconds	The number of debates in which the team came second. Only makes sense for British Parliamentary.
Who-beat-whom	If there are exactly two teams tied on all metrics earlier in the precedence than this one, then check if the teams have faced each other. If they have, the team that won their encounter is ranked higher. If they have seen each other more than once, the team that has won more of their encounters is ranked higher. If there are more than two teams tied, this metric is not applied. This metric can be specified multiple times. Each time who-beat-whom occurs, it applies to all the metrics earlier in the precedence than the occurrence in question.
Who-beat-whom (in divisions)	As for who-beat-whom, but only compares for teams in the same division. That is, the metric applies whenever there are exactly two teams from the same division exactly tied.

27.2 Speaker standings rules

The speaker standings precedence is only used in speaker standings (*i.e.*, it doesn't affect the operation of the tournament). As for team standings, the **speaker standings precedence** specifies which metrics are used to rank speakers, with the second metric tie-breaking the first, the third tie-breaking the second, and so on. The **speaker standings extra metrics** are metrics that will be shown in the speaker standings, but won't be used to rank speakers.

Metric	Description
Total	The sum of all speaker scores attained by the speaker. Note that if a speaker misses a round, they'll typically be relegated to the bottom of the speaker standings by this metric.
Average	The average of all speaker scores attained by the speaker.
Trimmed mean	The average speaker score after excluding their highest and lowest speaker scores. Also known as the <i>high-low drop</i> , <i>truncated mean</i> or <i>Olympic average</i> . If the speaker has only one or two scores, this metric just returns the average of those scores, without excluding any.
Standard deviation	The standard deviation of all speaker scores attained by the speaker. This metric is ranked in ascending order (smaller standard deviations ranked higher).
Average speaker score	The average total speaker score over all debates the team has had, not counting debates where they or their opponents forfeited.
Number of speeches given	The number of speaker scores associated with the speaker. (In tournaments where teams can rotate speakers, this may not be all rounds.) This metric is normally used as an «extra» (unranked) metric, because it'd be weird to rank by number of speeches given, but you can if you want to.

27.3 Motion balance

The motion balance page applies a statistical test to estimate the degree to which a motion is imbalanced. This is calculated by first making an underlying assumption that a motion is generally fair. This will be our null hypothesis: that, for a given motion, affirmative teams won the same number of times as negative teams.

Our chi-squared test will then be centred around disproving this hypothesis. If we disprove the hypothesis, we say that, in the context of this tournament and this draw, the motion ended up being unbalanced. However (technically speaking) if we fail to reject the null hypothesis, we would conclude that there is insufficient evidence to suggest that the motion was unbalanced in the context of this tournament.

The test proceeds by [calculating the chi-squared stat, then running a series of tests](#). The tests are where we go a little off-book with respect to statistical methodology. Normally we would test at a single «level of significance» (ie. with a certain degree of certainty), but that's insufficient in telling us how bad a motion ended up being. So, instead, we conduct a range of tests with a range of levels of significance, and calculate the minimum level of significance that causes our null hypothesis to be rejected. Using the minimum level of significance that rejects our null hypothesis, we can then grade the fairness of the motion on a scale. Motions whose tests fall below a certain threshold will be considered fair, while others will be graded based on the minimum.

For formats with topic selection, the same test is applied using the number of affirmative and negative vetoes in place of wins. The assumption here is that, during the time allotted for motion selection, teams estimate how appealing a motion is from their position, and then veto the topic that they feel is least favourable. Thus, the null hypothesis is that a motion that is perceived of as fair would be vetoed by affirmative and negative teams to an equal degree.

Team Code Names

Some tournaments use «code names» to obscure the institutional affiliations of teams. For example, rather than calling a team «Harvard DK», they would be presented in the draw as «Butterfly». A natural way to do this would be just to change the name of every team, but then the team's «real name» would be hidden from tournament staff, too.

Instead, Tabbycat allows you to assign code names to teams, alongside their real names. This way, you can have code names show to participants, while real team names show in administrative views (*e.g.* allocating adjudicators). It also allows you to «decode» team names for elimination rounds or final tab release easily, without having to actually change every team's name.

Advertencia: While the most frequently-used public views have been checked to ensure that they show only code names, not all views have been checked thoroughly. Please check views using demonstration data on a test site, configured in the same way that you would use it at your tournament, before using code names at a real tournament.

28.1 Assigning code names

Most methods of importing teams, including the simple importer and the `importtournament` command, automatically assign code names to teams. The code name is the name of the emoji that is automatically assigned at the same time. For example, the team assigned `will` will be code-named «Butterfly».

If you wish to use your own code names, you need to set the «code name» field of each team. Here are two ways to do this:

- **Edit Database area:** Enter the Edit Database area, and under **Participants > Teams**, click **Change**. Modify each team one by one, entering a new code name then saving.
- **`importtournament` command:** If you imported a tournament from CSV files, you can just add a `code_name` column to your teams CSV file.

28.2 Displaying code names

Code names are disabled by default; to enable then, go to **Setup > Configuration > Public Options**, and change the **Team code names** option. You can choose between the following options:

- Do not use code names
- Use real names everywhere, and show code names in tooltips
- Use code names for public; real names with code names in tooltips for admins
- Use code names for public; code names with real names in tooltips for admins
- Use code names everywhere; do not use tooltips (real names show in some admin views)

«Code names in tooltips» means that the code name will display in the details box that appears when you roll over a team's name, and similarly for real names.

One typical use is as follows:

- Before the tournament, set the team code names setting to *Use code names for public; real names with code names in tooltips for admins*. This hides real names from anything participants would see, but continues to refer to teams in administrative views by real names.
- After the break is announced, set it to *Use real names everywhere, and show code names in tooltips*. This basically decodes all team names, while still allowing people to look up the (now former) code name of a team.

Truco: If you're enabling team codes, you probably want to disable the **Show team institutions** option too.

Tournament Data Importers

This page describes how to write your own tournament data importer. It is aimed at an audience that is familiar with programming in Python, and may be willing to get their head around the Django model if necessary.

The **tournament data importer** is the class that imports data from one or more files (usually CSV files) into the database. A base class `BaseTournamentDataImporter` is in [importer/base.py](#). An example of a data importer is in [importer/anorak.py](#).

Por hacer: This page is incomplete. If you're finding this information insufficient, please contact Chuan-Zheng using the contact details in the [Authors & Acknowledgements](#) section.

29.1 Why write your own?

While Tabbycat has standard import formats, you might find that none of them fit the data that you need to import.

It's not possible to devise a single, universally-convenient import file format. Tabbycat supports way too many permutations of configurations for this to be workable. Instead, we provide the ones that have been useful before and are therefore likely to be useful again—but if your tournament has different needs, you might decide that it's easier to write an importer to conform to you, rather than conform to the importer.

A base importer class abstracts away most of the nitty-gritty of parsing files, allowing new importers to focus on their interpretation with as little code as possible.

To allow new importers to be written with as little code as possible, most of the work is abstracted to the base class. The flipside of this abstraction is that it induces a learning curve.

29.2 Basic workflow

1. Choose a name. We name importers after items of clothing in alphabetical order (starting at “Anorak”).
2. Write a subclass of `BaseTournamentDataImporter`.

3. Write the front-end interface. This will probably be a [Django management command](#).

29.3 A basic example

It's easiest to start with an example. Here's a basic importer with just one import method, which imports adjudicators.

```
from .base import BaseTournamentDataImporter, make_lookup, make_interpreter
from participants.models import Person, Adjudicator

class ExampleTournamentDataImporter(BaseTournamentDataImporter):

    lookup_gender = make_lookup("gender", {
        ("male", "m"): Person.GENDER_MALE,
        ("female", "f"): Person.GENDER_FEMALE,
        ("other", "o"): Person.GENDER_OTHER,
    })

    def import_adjudicators(self, f):
        """Imports adjudicators. `f` is a file object."""
        interpreter = make_interpreter(
            institution=Institution.objects.lookup,
            gender=self.lookup_gender,
            tournament=self.tournament
        )
        counts, errors = self._import(f, Adjudicator, interpreter)
        return counts, errors
```

Let's break this down. The method `import_adjudicators()` takes a single argument, a file object representing the CSV file. Most of the work is passed off to `self._import()`. This helper method is defined in `BaseTournamentDataImporter` and is where most of the intelligence lies.

When called, `self._import(f, model, interpreter)` does the following:

1. It reads the CSV file using a `csv.DictReader`. A `DictReader` iterates through the CSV file, yielding a dict for each line, whose keys are given by the column header names in the first row of the file.
2. On each line:
 - a. It passes the dict given by the `DictReader` to `interpreter`. The interpreter modifies the dict (or creates a new one) to prepare it for the model constructor, and returns it.
 - b. The dict returned by `interpreter` is then passed as keyword arguments to the `model` constructor.

So in very simplified form, `self._import(f, model, interpreter)` does this:

```
def _import(self, f, model, interpreter):
    reader = csv.DictReader(f)
    for line in reader:
        kwargs = interpreter(line)
        inst = model(**kwargs)
        inst.save()
```

(There's a lot more to it than that, but that's the basic idea.)

Importante: A consequence of relying on column headers to identify fields is that the header names in CSV files must match model field names exactly, unless they are deleted by the interpreter using the `DELETE` keyword (see below).

29.4 Interpreters

The main task of an importer, then, is to provide interpreters so that `self._import` knows how to interpret the data in a CSV file. An interpreter takes a dict and returns a dict. For example:

```
def interpreter(line):
    line['institution'] = Institution.objects.lookup(line['institution'])
    line['gender'] = self.lookup_gender(line['gender'])
    line['tournament'] = self.tournament
    return line
```

This interpreter does the following:

- Replaces `line['institution']` with an `Institution` object, by looking up the original value by name.
- Replaces `line['gender']` with a `Person.GENDER_*` constant. We'll come back to how this works later.
- Adds a new `line['tournament']` entry to the dict, being the `Tournament` object represented by `self.tournament`, the tournament that was passed to the importer's constructor.
- Leaves all other entries in the dict unchanged.

This looks simple enough, but it's very robust. What if a cell in the CSV file is blank, or what if the file omits a column? (For example, some tournaments might not collect information about participant gender, so Tabbycat doesn't require it.) We could deal with these scenarios on a case-by-case basis, but that's cumbersome.

Instead, we provide a `make_interpreter` method that returns an interpreter method which, in turn, takes care of all these details. This way, all you have to do is provide the functions that transform fields. So the following is equivalent to the above, but better:

```
interpreter = make_interpreter(
    institution=Institution.objects.lookup,
    gender=self.lookup_gender,
    tournament=self.tournament
)
```

Notice that we provided a callable in two of these keyword arguments, and a (non-callable) `Tournament` object to the third. `make_interpreter` is smart enough to tell the difference, and treat them differently. What it does with each field depends on (a) whether a value exists in the CSV file and (b) what transformation function was provided, as summarised in the following table:

Value in CSV file	Transformation	Action
	provided and not callable	populate model field with interpreter value
does not exist or blank	callable or not provided	do not pass to model constructor
exists and not blank	callable	call interpreter on column value, pass result to model constructor
exists and not blank	not provided	pass column value directly to model constructor

Truco:

- If a transformation isn't an existing method, you might find [lambda functions](#) useful. For example: `lambda x: Speaker.objects.get(name=x)`.
- You shouldn't check for mandatory fields. If a mandatory field is omitted, the model constructor will throw an error, and `self._import()` will catch the error and pass a useful message on to the caller. On the other

hand, if it's an optional field in the model, it should be optional in the importer, too. Similarly, interpreters generally shouldn't specify defaults; these should be left to model definitions.

- You don't need to include interpreter transformations for things like converting strings to integers, floats or booleans. Django converts strings to appropriate values when it instantiates models. So, for example, adding `test_score=float` to the above interpreter would be redundant.
-

29.4.1 More complicated interpreters

If you have a column in the CSV file that shouldn't be passed to the model constructor, you can tell the interpreter to remove it by using the special `DELETE` argument:

```
interpreter = make_interpreter(
    institution=Institution.objects.lookup,
    DELETE=['unwanted_column_1', 'unwanted_column_2']
)
```

The `make_interpreter` can only deal with modifications where each field is modified separately of the others (or not at all). If you want to combine information from multiple fields, you need to write your interpreter the long way (perhaps calling a function returned by `make_interpreter` to do some of the work).

On the other hand, if you don't need to do any transformations involving some sort of object or constant lookup, then you can just omit the `interpreter` argument of `self._lookup()`, and it'll just leave the fields as-is.

29.5 Lookup functions

In the above example, we used a function `self.lookup_gender` to convert from the text in the CSV file to a `Person.GENDER_*` constant. To make this easier, the importer provides a convenience function to define such lookup functions. Let's look at the relevant lines again:

```
lookup_gender = make_lookup("gender", {
    ("male", "m"): Person.GENDER_MALE,
    ("female", "f"): Person.GENDER_FEMALE,
    ("other", "o"): Person.GENDER_OTHER,
})
```

This should be a member of your subclass, in our case, `ExampleTournamentDataImporter`. It generates a function that looks something like:

```
@staticmethod
def lookup_gender(val):
    if val in ("male", "m"):
        return Person.GENDER_MALE
    elif val in ("female", "f"):
        return Person.GENDER_FEMALE
    elif val in ("other", "o"):
        return Person.GENDER_OTHER
    else:
        raise ValueError("Unrecognised value for gender: %s" % val)
```

The `make_lookup` function takes two arguments. The first is a text description of what it's looking up; this is used for the error message if the value in the CSV file isn't recognised. The second is a dict mapping tuples of valid strings to constants.

29.6 Debugging output

The `BaseTournamentDataImporter` constructor accepts a `loglevel` argument:

```
importer = MyTournamentDataImporter(tournament, loglevel=logging.DEBUG)
```

If `loglevel` is set to `logging.DEBUG`, the importer will print information about every instance it creates.

You can also pass in a logger for it to use (instead of the default one) with the `logger` argument.

CAPÍTULO 30

User Accounts

For obvious reasons, user logins are required to data entry and administrative functions. Conceptually, there are four levels of access:

Access	Should be used by	Grants access to	Is capable of
Public	The public	Publicly available information.	Viewing things, and submitting new ballots/feedback if electronic submission is permitted by the tournament.
Assistant	Data entry helpers	The assistant area	Entering, confirming and printing ballots and feedback, checking in ballots and people, and displaying the draw.
Superuser	Chief adjudicators	The administrator and assistant areas	Generating draws, allocating adjudicators and venues, and editing ballots, feedback and adjudicator scores.
Staff and superuser	Tab director	The administrator, assistant and Edit Database areas	Editing the database directly.

If a user account on the tab system belongs to someone who is also a participant in the tournament (*e.g.*, a chief adjudicator), these two capacities are completely separate. User accounts are only used to regulate access to administrative functions. Tabbycat doesn't know about any relationship between user accounts, and who is participating in the tournament.

30.1 Account roles

You should create an account for each person who needs to access the tab system. When you create an account in the Edit Database area, there are checkboxes for **Superuser status** and **Staff access**. Superusers have access to the administrator area, and staff have access to the Edit Database area. You should grant permissions as follows:

- Tab directors should get both superuser and staff status.

- Chief adjudicators and their deputies should get superuser status, but not staff status.
- Tab assistants (helping only with data entry) should get neither superuser nor staff status.

Tournament participants (other than tab staff) do not need an account. Everything they need to know can be accessed without an account. If you're using electronic ballots or electronic feedback, they access these using a URL that only they know (see *Private URLs*).

When doing data entry, users with superuser status should use the **assistant area**. The administrator area is intended for managing the tournament, and doesn't include someand should **not** in general be used for data entry. Specifically, the administrator area lacks checks that are important for data integrity assurance. It should be used only to override the normal *data entry* procedure, for example, to unconfirm or modify a ballot.

The **Edit Database** interface should not be used except where it is actually necessary. There are a few functions which require this, but as a principle, it shouldn't be used as a matter of course.

Nota: In theory, you could grant an account staff status but not superuser status. But then they'd be allowed to edit the database, but not run the tournament, which would be weird.

30.2 Adding accounts

To add an account:

1. Go to the Edit Database area of the site.
2. Under **Authentication and Authorization**, click the **Add** link next to **Users**.
3. Ask the user to enter a username, password and possibly email address.
 - Only they should know what the password is.
 - If you're hosting on the internet, all passwords should be at least moderately strong!
 - Passwords are not stored as raw passwords, so you can't figure out what their password is.
 - The email address is optional.
 - This email address is only used to reset their password if they forget it, and has nothing to do with the email address that Tabbycat uses to send emails to tournament participants (*e.g.* private URL links).
4. If they are being assigned superuser and/or staff status, check the relevant boxes.
5. Click «Save» or «Save and add another».

Venue Constraints

Tabbycat supports a basic form of venue constraints. A **venue constraint** is a requirement that a particular **team**, **adjudicator**, **institution** or **division** be assigned to a venue in a particular **venue category**. Typical uses would include:

- Meeting venue accessibility requirements of particular teams (*e.g.* step-free access)
- Placing adjudication core and tab team members close to the tab room
- Keeping all debates in a division in one location

Constraints apply to **venue categories**, not individual venues. That is, you specify that (say) a team should be given a venue from a particular *list* of venues. Of course, it's permissible for a venue category to have only one venue in it.

The algorithm used to satisfy venue constraints is not guaranteed to be optimal. In some rare cases, it may propose an allocation that fails some constraints, even though some other allocation would have satisfied all (or more) constraints. In almost all practical circumstances, however, it should work, and save human effort (and time) in specially allocating rooms.

31.1 Adding venue categories

Before you add venue constraints, you first need to add venue categories. Each venue category is a list of venues, typically satisfying a particular need. For example, you might have a category for each of the following:

- Venues with step-free access
- Venues that are close to general assembly (the briefing room)
- Venues that are close to the tab room
- Venues that are, or venues that are not, being live-streamed

Each venue can be in as many categories as you like (or none at all).

Name

(Accessible)

Name of category, e.g., "Purple", "Step-free access", "Close to tab room". This name is shown when the category is prefixed or suffixed to a venue name in the draw, e.g., "Purple – G05".

Description

Venues we are sure have step-free access

Description, as the predicate of a sentence, e.g. "has step-free access", "is close to the briefing hall". This description follows "This venue" when shown in tooltips, e.g., "This venue is close to the briefing hall."

Display in venue name

Display as suffix

Prefix: "Purple – G05", Suffix: "G05 – Purple"

Display in public tooltip

☐

Displays the description in the tooltip for the venue on public pages. The description, if not blank, will always show on admin pages.

Venues

Coconut G04
Coconut G05
Coconut G06
Coconut G08
Main Hall (Accessible)
Outside
Sub Hall (Accessible)
Z08
Z09
Z10

To add or edit venue categories, go to the **Import Data** area (under Setup) then select **Add/Edit Venue Categories**. Note that this page will show all existing Venue Categories first before showing the blank forms that allow you to create new categories. Give your category a name (like «Step-free access»), assign it some venues, then click the «Save Venue Categories» button at the bottom of the page.

Alternately you can add or edit a venue category by going to the **Edit Database** area (under Setup), scroll down to «Venues» and click «Venue categories». Then click the + **Add venue category** button in the top-right of the page or click an existing item.

31.2 Adding venue constraints

To add or edit venue constraints, go to the **Import Data** area (under Setup) then select **Add/Edit Venue Constraints**. Note that this page will show all existing Venue Constraints first before showing the blank forms that allow you to create new categories. Note that the «Constraine ID» field should let you select from a dropdown or type in the name of an adjudicator, institution, or team (rather than having to lookup the exact ID).

Constraineed Type

institution

Constraineed ID

15

Delete the existing number and start typing the name of the person/team/institution you want to constrain to lookup their ID.

Venue Category

(Accessible)

Priority

100

Alternately you can add or edit a venue category by going to the **Edit Database** area (under Setup), scroll down to «Venues» and click «Venue constraints». Then click the **+ Add venue category** button in the top-right of the page or click an existing item.

For each constraint, you need to specify four things:

Cate-gory	The venue category to which the subject of this constraint should be locked.
Prio-ri-ty	This is a number used to resolve conflicts between constraints. Constraints with higher priority (greater number) take precedence over those with lower priority. If none of your constraints will ever conflict, then the priority is arbitrary (but it must still be specified).
Sub-ject con-tent type	The type of subject to which this constraint relates: adjudicator, team, institution or division.
Sub-ject ID	Which adjudicator, team, institution or division the constraint relates to. The textbox takes a number (the ID of the object in the database), but you can search for the subject by clicking on the search icon next to it. This will bring up a table of objects of the type specified in «subject content type» for you to choose from. (You need to select the subject content type first.)

31.3 Applying venue constraints

If you don't have any venue constraints for adjudicators, venue constraints are applied automatically when the draw is generated.

However, if you have one or more venue constraints for adjudicators, it's not possible to take adjudicator venue constraints into account during draw generation, because the adjudicator allocation isn't known then. You'll need to run the venue allocation yourself after you've allocated adjudicators.

To run venue allocation, go to **Edit Venues** (while looking at the draw), then in the screen where you can edit venues, click the **Auto Allocate** button. You can also do this at any other point (say, after adding a new venue constraint) if, for whatever reason, you would like to re-run the venue allocation algorithm.

If a venue constraint couldn't be met, a message will show in the «conflicts/flags» column of the draw. A constraint might not be met for a number of reasons:

- It could be that constraints of different parties (say, one team and one adjudicator) conflicted, so only one could be fulfilled.
- It could be that all available rooms in the relevant category were already taken by other, higher-priority constraints.
- It could just be one of those edge cases that's too hard for the naïve algorithm to handle.

Currently, Tabbycat doesn't tell you which of these happened, so if the venue allocation fails to meet all your constraints, it's on you to figure out why. In most scenarios, we imagine you'll have few enough constraints that this will be obvious; for example, if the chief adjudicator is judging a team with accessibility requirements, it might be obvious that the latter's constraint took priority. We might in future add support for more useful guidance on conflicting constraints, but we currently consider this to be of low priority.

There are a number of ways to report bugs, ask for help, or submit feedback.

32.1 Facebook

Our [Facebook group](#) is a good place to ask for help. It's also a good place to keep up with new releases and participate in more general discussions of features and ideas.

32.2 GitHub

Adding an issue to our [GitHub repository](#) is a great way to let us know about bugs or writeup suggestions for how to improve Tabbycat. Pull requests are also encouraged!

When submitting bugs or reporting errors please let us know your site address (if installed online) or operating system (if local) along with a complete description of the problem along with any error messages.

32.3 Email

Feel free to *contact the maintainers directly* if you are not able to access Facebook or GitHub.

Authors & Acknowledgements

33.1 Authors

Tabbycat was authored by Qi-Shan Lim for Auckland Australs 2010. The current active maintainers are:

- Philip Belesky ([pb-e-mail](#))
- Chuan-Zheng Lee ([cz-e-mail](#))

Please don't hesitate to contact us with any suggestions, expressions of interest or generally anything relating to Tabbycat.

33.2 Contributors

- Étienne Beaulé has contributed many features, fixes, and suggestions across many aspects of Tabbycat.
- Thevesh Theva contributed the algorithm for calculating the liveness of teams within a particular break category.
- Viran Weerasekera contributed the statistical tests used to estimate the degree to which a motion's results and vetoes are balanced.
- Viran Weerasekera, Valerie Tierney, Molly Dale, Madeline Schultz, and Vail Bromberger contributed to the «Tournament Logistics» section of our documentation.

33.3 Sponsors

- Thanks to the Western Australian Debating League for sponsoring various features related to organising division-based tournaments and round-robin based draw methods.

34.1 2.3.3

Release date: 26 April 2020

- Fixed issue where the ballot graph would ignore draft ballots getting confirmed
- Fixed team draw notifications failing due to an unexpected variable
- Fixed ballot receipts not showing decimal speaker points
- Fixed issue where Docker installs would compile without css/javascript; breaking many pages

34.2 2.3.2

Release date: 19 October 2019

- Fixed issue where teams would appear to be unavailable in break rounds
- Other minor fixes

34.3 2.3.1

Release date: 6 October 2019

- Fixed issue where the institutions list would count teams/adjudicators outside of the tournament
- Fixed issue where a rejected ballot form would crash rather than providing an error message
- Fixed issue where the javascript bundle would not build on a local windows install
- Fixed issue where the adjudicator record pages would show an unreleased motion if that round's draw was released

34.4 2.3.0 (LaPerm)

Release date: 27 September 2019

- **Added a preformed panel system which provides a powerful take on a “shadow draw” workflow**

- Shadow draw systems allow an adjudication core to form panels prior to a round being drawn. For example, the panels for Round 4 could be formed while Round 3 is taking place. Most implementations do so by having the tab system create a copy of the Round 3 draw, form new panels on top of it, and then transpose these panels onto Round 4. In large tournaments this workflow allows an adjudication core much more time to consider panel formation
- Tabbycat’s preformed panels are formed per-round under a section available under the Setup menu. This interface looks like the standard Edit Adjudicators interface, but the “debates” shown are based on a simulation of that round’s results. These fake debates can then be prioritised
- Adjudicators can then be allocated to those fake debates in order to create a pre-formed panel. When the real draw is ready to be created, the priority of each preformed panel will be matched to the priority of the real debates
- By using the existing per-debate priority system, and by giving pre-formed panels their own priority, this workflow allows for very fine amounts of control over exactly how preformed panels are allocated as compared to a more simple top-down transposition of panels. Adjudication cores can easily target general areas of the draw (e.g. break-threshold brackets); control adjudicator strength within and across panels; and still account for special cases where a debate requires a particularly strong panel. At the same time, our existing options for automatic prioritisation and automatic allocation apply to all steps of this process so that preformed panels can be created and deployed rapidly

- **Rewrote the Edit Adjudication, Venues, and Teams pages to enable a number of enhancements**

- These pages now live-update changes that were made on other instances of that page. As a result, users on different computers can each open the Edit Adjudicators page and see the changes made by the other users. This feature, along with sharding, should make it easier than ever to distribute the task of adjudicator allocation across an entire adjudication core
- A new interface layout should better maximise space, particularly in BP settings, while also increasing the font size of key information
- The unused panel is now able to sort adjudicators by name, score, or drag order
- Average scores for all adjudicators, and a voting majority, are now shown next to the panel
- Various allocation-relevant settings, such as the minimum feedback score needed for a voting position, are now available inline on the allocation page itself. This should enable much faster tweaks/iterations of these values

- **The ballot entry page will now indicate which teams have currently or recently given “iron person” speeches so that these can be easily tracked, audited, and confirmed. It does so by showing both a text-highlight/icon in the table and in a dedicated modal window. Thanks to Étienne Beaulé for contributing this feature!**

- **Split up the Django settings files. Note that this means if you are upgrading a local install of Tabbycat to this version you v**

- Copy `tabbycat/settings/local.example` to become `local.py` (and fill in your original database details)
- Optional: repeat the same copying procedure for `development.example` and set the `LOCAL_DEVELOPMENT` environmental variable to `True` if you would like to use the settings designed to aid local development

- **A range of improvements to the email notifications contributed by Étienne Beaulé:**

- Ballot receipt emails now provide more information about team scores/points
 - Emails are now in a rich-text format
 - Custom emails may be sent out to select participants through the web-interface
 - Participants can be specifically included or excluded from receiving a notification before sending with checks for duplicate messages
 - Teams can be sent emails with their draw details
 - Emails can be tracked to determine if sent or read (SendGrid-specific)
- **Expanded the use of private URLs (Encore Étienne Beaulé):**
 - QR codes are now included in addition to the URL when printing private URLs
 - Private landing pages will now display check-in status (if check-ins are used) along with further details regarding break categories, regions, etc.
 - Current and former draw assignments will display along with submitted ballots (for adjudicators) on landing pages
 - **Reworked how conflicts are determined to support double-past institutional conflicts:**
 - Added a «team-institution conflict» model
 - Like adjudicator-institution conflicts, team-institution conflicts are automatically created if you use the simple importer or the command-line importer; but if you edit the database, it's your responsibility to add/edit them
 - Institutional affiliations no longer matter for determining conflicts for either teams or adjudicators; only institutions listed in the team's or adjudicator's conflicts matter
 - An adjudicator/team now conflicts with an adjudicator if *any* institution appears as an institutional conflict for both parties
 - When printing scoresheets you can now edit the motions display just on that printing page. This allows you to use placeholder motions in Tabbycat (in order to prevent leaks) while still producing ballots with the correct motions
 - Tabbycat no longer tracks which round is the “current” round and instead tracks the completion of individual rounds. This change does not alter any existing workflows, but makes it easier to run simultaneous draws in out-rounds
 - Info-slides can now be split into paragraphs
 - Check-in pages now differentiate between teams with 1 and 2 checked-in people in two-team formats
 - Institutional caps in breaks can be based on the number of teams in the break. Thanks to Viran Weerasekera for this feature!
 - Several Tabbycat functions, adjudicator/venue allocation and email notifications, have been shifted to worker processes to help make them more reliable. If you are upgrading a Tabbycat instance that you will continue to use for new tournaments you will need to install the Heroku toolbelt and run `heroku ps:scale worker=1`
 - Upgraded to Python 3.6, dropped support for Python 3.5 and below. Note that this will require you to upgrade your python versions if running locally.

34.5 2.2.10

Release date: 10 February 2019

- Fixed the display of feedback quantities on the Feedback Overview Page
- Fixed issue where “ignored” feedback would hide the result from the feedback graph but not affect an adjudicator’s current score. Thanks to Étienne for this fix

34.6 2.2.9

Release date: 24 January 2019

- Fixed an issue that could cause errors for tournaments when using an atypical number of rounds and break sizes. Thanks to Étienne for this fix
- Fixed an issue where the display of adjudicator’s record links would display their name twice

34.7 2.2.8

Release date: 14 December 2018

- Fix issue where the check-in buttons were always disabled on admin and assistant pages
- Other minor fixes. Thanks to Étienne for these and for the check-in button fix!

34.8 2.2.7

Release date: 16 November 2018

- Lock redis-py version to 2.10.6, as workaround for [compatibility issue between django-redis and redis-py](#)
- Fix login link on page shown after a user logs out

34.9 2.2.6

Release date: 14 November 2018

- Fix issue where check-ins could not be revoked
- Fix issue where the standings overview “dashboard” included scores from elimination rounds. Thanks to Étienne for this fix
- Fix issue where the Average Individual Speaker Score metric would fail to calculate in some circumstances. Thanks to Étienne for this fix
- Fix issue where draw emails would crash if venues were unspecified. Thanks, again, to Étienne for this fix!

34.10 2.2.5

Release date: 21 October 2018

- Remove the buttons from the public check-ins page (as these do nothing unless the user is logged in)
- Hopefully fixed error that could cause Team- and Adjudicator- Institutional conflicts to not show properly on Allocation pages

- Thanks to Étienne for pull requests fixing rare bugs in the user creation form and break generation when rounds are not present

34.11 2.2.4

Release date: 9 October 2018

- Small fixes for functions related to email sending, conflict highlighting, and certain configurations of standings metrics

34.12 2.2.3

Release date: 28 September 2018

- *Literally* fix the issue causing public views of released scoresheets to throw errors (thanks for the pull request Étienne)
- Fix minor spacing issues in printed ballots (thanks for the pull request Étienne)
- Fix issue where institution-less adjudicators would cause some draw views to crash (thanks for the pull request Étienne)

34.13 2.2.2

Release date: 22 September 2018

- *Actually* fix the issue causing public views of released scoresheets to throw errors

34.14 2.2.1

Release date: 21 September 2018

- Fix issue causing public views of released scoresheets to throw errors

34.15 2.2.0 (Khao Manee)

Release date: 20 September 2018

- **Implemented a new server architecture on Heroku along with other optimisation that should significantly improve the performance**
 - Add the <https://github.com/heroku/heroku-buildpack-nginx.git> build pack under the Settings area of the Heroku Dashboard and positioning it first
 - If your Heroku Stack is not «heroku-16» (noted under that same Settings page) it will need to be set as such using the Heroku CLI and the `heroku stack:set heroku-16 --app APP_NAME` command
- Added a page to the documentation that details how to scale a Tabbycat site that is receiving large amounts of traffic; and another page that documents how to upgrade a Tabbycat site to a new version

- Added support for Japanese and Portuguese. Let us know if you'd like to help contribute translations for either language (or a new one)!
- The results-entry page now updates its data live, giving you a more up to date look at data entry progress and reducing the cases of old data leading people to enter new ballots when they meant to confirm them
- **A huge thanks to Étienne Beaulé for contributing a number of major new features and bug fixes. Notably:**
 - Added a means to mark feedback as “ignored” so that it still is recorded as having been submitted, but does not affect the targeted-adjudicator’s feedback score
 - Added email notification to adjudicators on round release
 - Implemented participant self-check-in through the use of their private URLs
 - Gave all participants to a tournament a private URL key rather than being by team, and added a landing page for the participants using this key
 - Implemented templated email notifications with ballot submission and round advance with the messages in a new settings panel. Private URL emails are now also customizable
 - Added the «average individual speaker score» metric which averages the scores of all substantive speeches by the team within preliminary rounds. The old «average speaker score» metric has been renamed to to «average total speaker score»
 - Reworked the ballots status graph to be an area chart
- Added the ability to hide motions on printed ballots (even if they have been entered). Thanks to Github user 0zlw for the feature request!
- Added the ability to unconfirm feedback from any of the views that show it
- BP motion statistics now also show average points split by bench and half
- Added a warning when users are close to their free-tier database limit on Heroku that makes it clear not to create new tournaments
- Added `exportconfig` and `importconfig` management commands to export and import tournament configurations to a JSON file
- Upgraded `django-dynamic-preferences` to version 1.6

This won't affect most users, but advanced users previously having problems with a stray `dynamic_preferences_users_userpreferencemodel` table who are upgrading an existing instance may wish to run the SQL command `DROP TABLE dynamic_preferences_users_userpreferencemodel;` to remove this stray table. When this table was present, it caused an inconsistency between migration state and database schema that in turned caused the `python manage.py flush` command to fail. More information is available in the [django-dynamic-preferences changelog](#)

34.16 2.1.3

Release date: 21 August 2018

- Added an alert for British Parliamentary format grand-final ballots that explains the workaround needed to nominate a sole winner
- Improved display of images shown when sharing Tabbycat links on social media
- Optimised the performance of several commonly-loaded pages. Thanks to Étienne Beaulé for the pull request!

- Prevented the entry of integer-scale feedback questions without the required min/max attributes
- Provided a shortcut link to editing a round's feedback weight
- Prevented standings from crashing when only a single standings metric is set

34.17 2.1.2

Release date: 14 July 2018

- Fixed an error caused when calculating breaks including teams without institutions
- Improved display of long motions and info slides
- Fixed bug in feedback progress tracking with UADC-style adjudication
- Fixed bug where the public checks page would cause large amounts of failing requests
- Fixed visual issue with adjudicator lists wrapping poorly on mobile devices
- Limited the time it takes to serve requests to match Heroku's in-built limit; this may help improve the performance of sites under heavy load

34.18 2.1.1

Release date: 19 May 2018

- The Scan Identifiers page now orders check-ins from top to bottom
- The Scan Identifiers now plays different sounds for failed check-ins
- The Scan Identifiers now has a toggle for sounds; allowing it to work in browsers like Safari that block auto-playing audio

34.19 2.1.0 (Japanese Bobtail)

Release date: 7 May 2018

- Added an introductory modal for the adjudicator allocation page to help outline how the features and workflow operate
- Added an automated method for assigning importances to debates using their bracket or “liveness”. This should allow smaller tournaments to more easily assign importances and save time for larger tournaments that do so
- **Added the ability to switch between using “team codes” and standard team names**
 - By default team codes are set to match that team's emoji, but team codes are editable and can be imported like standard data
 - Team codes can be swapped in and out for standard team names at will, with precise control over the contexts in which either is used — i.e. in public-facing pages, in admin-facing pages, in tooltips, *etc.*
- **Added a range of “check-in” functionality**
 - This includes barcode assignment, printing, and scanning. Scanning methods are optimised both for manual entry, entry with barcodes scanners, and for a “live” scanning view that uses your phone's camera!

- This includes new people and venue status pages that show an overview of check-in status and allow for easy manual check-ins; ideal for a roll-calls!. This page can also be made public
- Ballot check-ins have been converted to this new method, and now all printed ballots will contain the barcodes needed to scan them
- Venue check-ins have been added alongside the standard “person” check-ins to allow you to track a room’s status at the start of the day or round-by-round
- **Added (partial) translations in French, Spanish and Arabic**
 - Users can now use a link in the footer to switch the site’s language into French, Spanish, or Arabic. By default Tabbycat should also match your browser’s language and so automatically apply those languages if it matches
 - Our translations are generously provided by volunteers, but (so far) do not cover all of the interface text within Tabbycat. If you’re interested in helping to translate new or existing languages please get in touch!
 - Thanks to the excellent team at QatarDebate for contributing the Arabic translations, and to Alejandro, Hernando, Julian and Zoe for contributing the Spanish translations.
- **Added a new (beta) feature: allocation “sharding”**
 - Sharding allows you to split up the Adjudicator Allocation screen into a defined subset of the draw. This has been designed so that you can have multiple computers doing allocations simultaneously; allowing the adjudication core to split itself and tackle allocations in parallel.
 - Shards can be assigned into defined fractions (i.e. halves or fifths) according to specific criteria (i.e. bracket or priority) and following either a top-to-bottom sorting or a mixed sorting that ensures each bracket has an even proportion of each criteria.
- Added an option to show a «Confirm Digits» option to pre-printed ballots that asks adjudicators to confirm their scores in a manner that may help clarify instances of bad handwriting. This can be enabled in the «Data Entry» settings area.
- Added a “liveness” calculator for BP that will estimate whether each team has, can, or can’t break in each of their categories (as previously existed for 2-team formats)
- Added draw pull-up option: pull up from middle
- Added new draw option: choose pull-up from teams who have been pulled up the fewest times so far
- Added the ability to have different “ballots-per-debates” for in/out rounds; accommodating tournaments like Australian Easters that use consensus for preliminary rounds but voting for elimination rounds.
- Added time zone support to the locations where times are displayed
- Administrators can now view pages as if they were Assistants; allowing them to (for example) use the data entry forms that enforce double-checking without needing to create a separate account
- Fixed ² test in motion statistics, and refactored the motion statistics page
- Teams, like adjudicators, no longer need to have an institution
- Added a page allowing for bulk updates to adjudicator scores
- Added break categories to team standings, and new team standings pages for break categories
- **Made speaker standings more configurable**
 - Second-order metrics can now be specified
 - Added trimmed mean (also known as high-low drop)
 - Added ability to set no limit for number of missed debates

- Standard deviation is now the population standard deviation (was previously sample), and ranks in ascending order if used to rank speakers.

■ Quality of life improvements

- Added a «» indicator to more obviously liveness in the adjudicator allocation screen
- Added WYSIWYG editor for tournament welcome message, and moved it to tournament configuration
- Added «appellant» and «respondent» to the side name options
- Added a two new columns to the feedback overview page: one that displays the current difference between an adjudicator's test score and their current weighted score; another the displays the standard deviation of an adjudicator's feedback scores
- Added an "important feedback" page that highlights feedback significantly above or below an adjudicator's test score
- Added a means to bulk-import adjudicator scores (for example from a CSV) to make it easier to work with external feedback processing
- Speakers and speaker's emails in the simple importer can now be separated by commas or tabs in addition to new lines
- The «shared» checkbox in the simple importer is now hidden unless the relevant tournament option is enabled
- Current team standings page now shows silent round results if «Release all round results to public» is set
- The Consensus vs Voting options for how ballots work has now been split into two settings: one option for preliminary rounds and one option for elimination rounds
- Speaker scores now show as integers (without decimals) where the tournament format would not allow decimals
- Added a page showing a list of institutions in the tournament
- On the assistant «enter results» page, pressing «/» jumps to the «Find in Table» box, so data entry can be done entirely from your keyboard

■ Switched to using a Websockets/Channels based infrastructure to better allow for asynchronous updates. This should also

- On Heroku: You should remove the Memcachier plugin and instead add "heroku-redis" to any instances being upgraded
- Locally: You should recreate your *local_settings.py* from the *local_settings.example* file

■ Upgraded to Django 2.0

- Converted most raw SQL queries to use the new `filter` keyword in annotations

34.20 2.0.7

Release date: 13 April 2018

- Fixed an issue preventing draws with pre-allocate sides generating

34.21 2.0.6

Release date: 20 March 2018

- Added reminder to add own-institution conflicts in the Edit Database area
- Other minor fixes

34.22 2.0.5

Release date: 7 February 2018

- Improved the printing of scoresheets and feedback forms on Chrome.
- Other minor fixes

34.23 2.0.4

Release date: 22 January 2018

- Add alert for users who try to do voting ballots on BP-configured tournaments
- Fixed issue where draws of the «manual» type would not generate correctly
- Fixed issue where a ballot's speaker names dropdown would contain both team's speakers when using formats with side selection
- Fixed issue where scoresheets would not show correctly under some configurations
- Improved display of really long motions when using the inbuilt motion-showing page
- Other minor fixes

34.24 2.0.3

Release date: 3 December 2017

- Fixed issue where the “prefix team name with institution name” checkbox would not be correctly saved when using the Simple Importer
- Removed the scroll speed / text size buttons on mobile draw views that were making it difficult to view the table
- Improved the display of the motions tab page on mobile devices and fixed an issue where it appeared as if only half the vetoes were made

34.25 2.0.2

Release date: 27 November 2017

- **Fixes and improvements to diversity overview**
 - Fixed average feedback rating from teams, it was previously (incorrectly) showing the average feedback rating from all adjudicators

- Gender splits for average feedback rating now go by target adjudicator; this was previously source adjudicator
- Persons with unknown gender are now shown in counts (but not score/rating averages); a bug had previously caused them to be incorrectly counted as zero
- Improved query efficiency of the page
- Improved the BP motions tab for out-rounds by specifying advancing teams as «top/bottom ½» rather than as 1st/4th and removed the average-points-per-position graphs that were misleading
- Improved handling of long motions in the motion display interface
- Fixed issue where creating BP tournaments using the wizard would create an extra break round given the size of the break specified
- Fixed auto-allocation in consensus panels where there are fewer judges than debates in the round
- Fixed reply speaker validity check when speeches are marked as duplicate
- Prohibit assignment of teams to break categories of other tournaments in Edit Database area

34.26 2.0.1

Release date: 21 November 2017

- Fixed issue where results submission would crash if sides are unconfirmed
- Fixed issue where scoresheets would not display properly for adjudicators who lack institutions
- Fixed issue where the round history indicators in the Edit Adjudicators page would sometimes omit the «rounds ago» indicator

34.27 2.0.0 (Iberian Lynx)

Release date: 13 November 2017

- **British Parliamentary support**
 - Full support for British Parliamentary format has been added and we're incredibly excited to see Tabbycat's unique features and design (finally) available as an option for those tabbing in the predominant global format
 - As part of the implementation of this format we've made significant improvements over existing tab software on how sides are allocated within BP draws. This means that teams are less likely to have "imbalanced" proportions of side allocations (for example having many more debates as Opening Government than Closing Opposition)
 - We've added a new «Comparisons» page added to the documentation to outline some of the key differences between Tabbycat and other software in the context of BP tabbing
- **Refreshed interface design**
 - The basic graphic elements of Tabbycat have had a their typography, icons, colours, forms, and more redesign for a more distinctive and clear look. We also now have an official logo!
 - The «Motions» stage of the per-round workflow has now been rolled into the Display area to better accommodate BP formats and consolidate the Draw/Motion "release" process

- Sidebar menu items now display all sub-items within a section, such as for Feedback, Standings, and Breaks
- Better tablet and mobile interfaces; including a fully responsive sidebar for the admin area that maximises the content area
- More explicit and obvious calls-to-action for the key tasks necessary to running a round, with better interface alerts and text to help users understand when and why to perform crucial actions
- Redesigned motions tab page that gives a better idea of the sample size and distribution of results in both two- and four- team formats
- **Improved handling of Break Rounds ballots and sides allocation**
 - The positions of teams within a break round are now created by the initial draw generation in an “unset” state in recognition that most tournaments assign these manually (through say a coin toss). This should help clarify when showing break rounds draws when sides are or are not finalised
 - Break rounds ballots for formats where scores are not typically entered (i.e. BP) will only specify that you nominate the teams advancing rather than enter in all of the speakers’ scores
- Now, like Break Categories, you can define arbitrary Categories such as “Novice” or “ESL” to create custom Speaker tabs for groups of Speakers
- You can now release an Adjudicators Tab showing test scores, final scores, and/or per-round feedback averages
- Information Slides can now be added to the system; either for showing to an auditorium within Tabbycat or for displaying alongside the public list of motions and/or the motions tab
- Teams and adjudicators are no longer required to have institutions; something that should be very useful when setting up small IVs and the like
- Private URLs can now be incrementally generated. Records of sent mail are now also kept by Tabbycat, so that emails can be incrementally sent to participants as registration data changes
- **Quality of life improvements**
 - After creating a new tournament you will now be prompted to apply a basic rules and public information preset
 - Better handling of errors that arise when a debate team is missing or where two teams have been assigned the same side
 - Fixed issue where the adjudicator feedback graphs would not sort along with their table
 - The Feedback Overview page now makes it more clear how the score is determined, the current distribution of scores, and how scores affect the distribution of chairs, panellists, and trainees
 - Speaker tabs now default to sorting by average, except for formats where we are certain that they must be sorted by total. The speaker tab page itself now prominently notes which setting is currently using
 - “Feedback paths” now default to a more permissive setting (rather than only allowing Chairs to submit feedback) and the Feedback Overview page will note that current configuration
 - Emails can be assigned to adjudicators and teams in the Simple Importer
 - More of the tables that allow you to set or edit data (such as the check-in tables for adjudicators, teams and venues) now automatically save changes
 - When adding/editing users extraneous fields have been hidden and the «Staff» and «Superuser» roles have new sub-text clarifying what they mean for users with those permissions
 - Team record pages now show cumulative team points, and if the speaker tab is fully released, speaker scores for that team in each debate

34.28 1.4.6

Release date: 23 October 2017

- Fixed issue where speaker standings with a large amount of non-ranking speakers would cause the page to load slowly or time-out.

34.29 1.4.5

Release date: 14 October 2017

- Added warning message when adjudicator scores are outside the expected range
- Fixed handling of uniqueness failure in simple importer for teams

34.30 1.4.4

Release date: 27 September 2017

- Fixed Vue dependency issue preventing Heroku installs after a dependency release
- Fixed issue with formatting non-numeric standings metrics
- Fixed behaviour of public tabs when all rounds are silent

34.31 1.4.3

Release date: 9 September 2017

- A number of improvements to error handling and logging
- Changed the «previous round» of an elimination round to point to the last one in the same break category
- Other minor bug fixes

34.32 1.4.2

Release date: 23 August 2017

- Minor bug fixes and error logging improvements

34.33 1.4.1

Release date: 2 August 2017

- Fixed bug that prevented edited matchups from being saved
- Added flag to prevent retired sites from using the database for sessions

34.34 1.4.0 (Havana Brown)

Release date: 26 July 2017

- **Overhauled the adjudicator allocation, venue allocation, and matchups editing pages, including:**
 - Upgraded to Vue 2.0 and refactored the code so that each page better shares methods for displaying the draw, showing additional information, and dragging/dropping
 - When dragging/dropping, the changed elements now “lock” in place to indicate that their saving is in-progress
 - Added conflicts and recent histories to the slideovers shown for teams/adjudicators
 - Added “ranking” toggles to visibly highlight adjudicator strengths and more easily identify unbalanced panels
 - Each interface’s table is now sortable by a debate’s importance, bracket, liveness, etc.
- Added a new «Tournament Logistics» guide to the documentation that outlines some general best practices for tabbing tournaments. Thanks to Viran Weerasekera, Valerie Tierney, Molly Dale, Madeline Schultz, and Vail Bromberger for contributing to this document
- Added (basic) support for the Canadian Parliamentary format by allowing for consensus ballots and providing a preset. However note that only some of the common draw rules are supported (check our documentation for more information)
- Added an ESL/EFL tab release option and status field
- Added a chi-squared test to measure motion balance in the motion standings/balance. Thanks to Viran Weerasekera for contributing this
- The Auto Allocate function for adjudicators will now also allocate trainees to solo-chaired debates
- Added a “Tab Release” preset for easily releasing all standings/results pages after a tournament is finished
- Added “Average Speaks by Round” to the standings overview page
- Fixed issue where the Auto Allocator was forming panels of incorrect strengths in debates identified as less important
- Fixed issue where toggling iron-person speeches on and off wouldn’t hide/unset the relevant checkboxes
- Fixed issue where VenueCategories could not be edited if they did not have Venues set
- Various other small fixes and improvements

34.35 1.3.1

Release date: 26 May 2017

- Fixed bug that allowed duplicate emoji to be occasionally generated

34.36 1.3.0 (Genetta)

Release date: 9 May 2017

- Added the ability to mark speeches as duplicates when entering ballots so that they will not show in speaker tabs, intended for use with “iron-man” speeches and swing speakers

- Reworked venue constraints and venue display options by streamlining «venue groups» and «venue constraint categories» into a single «venue category» type, with options for how they are used and displayed
- Relocated the Random (now renamed “Private”) URL pages to the Setup section and added pages for printing/emailing out the ballot submission URLs
- Reworked the simple data importer (formerly the visual importer) to improve its robustness
- Improved guards against having no current round set, and added a new page for manually overriding the current round (under Configuration)
- Added a preference for controlling whether assistant users have access to pages that can reveal draw or motions information ahead of their public release
- Added the ability to limit tab releases to a given number of ranks (*i.e.* only show the top 10 speakers)
- Added the ability to redact individual person’s identifying details from speaker tabs
- Added the ability for user passwords to be easily reset
- Added a minimal set of default feedback questions to newly created Tournaments
- When a tournament’s current round is set, redirect to a page where it can be set, rather than crashing
- A number of other minor bug fixes and enhancements

34.37 1.2.3

Release date: 17 March 2017

- Improved the display of the admin ballot entry form on mobile devices
- A number of other minor bug fixes

34.38 1.2.2

Release date: 4 March 2017

- Protected debate-team objects from cascaded deletion, and added warning messages with guidance when users would otherwise do this
- A number of other minor bug fixes

34.39 1.2.1

Release date: 25 February 2017

- Printable feedback forms will now display the default rating scale, any configured introduction text, and better prompt you to add additional questions
- A number of other minor bug fixes

34.40 1.2.0 (Foldex)

Release date: 15 February 2017

- Changed the core workflow by splitting display- and motion- related activities into separate pages to simplify each stage of running a round
- Added support for Docker-based installations to make local/offline installations much more simple
- Added a «Tabbykitten» version of Tabbycat that can be deployed to Heroku without a needing a credit/debit card
- Added button to load a demo tournament on the “New Tournament” page so it is easier to test-run Tabbycat
- Changed venue groups to be separate to venue constraint categories
- Modified the licence to clarify that donations are required for some tournaments and added a more explicit donations link and explanation page
- Added information about autosave status to the adjudicator allocations page
- Added configurable side names so that tournaments can use labels like «Proposition»/»Opposition» instead of «Affirmative»/»Negative»
- Started work on basic infrastructure for translations

34.41 1.1.7

Release date: 31 January 2017

- Yet more minor bug fixes
- The auto-allocation UI will now detail your minimum rating setting better
- Added guidance on database backups to documentation

34.42 1.1.6

Release date: 19 January 2017

- A number of minor bug fixes
- Added basic infrastructure for creating tabbycat translations

34.43 1.1.5

Release date: 12 January 2017

- A number of minor bug fixes and improvements to documentation

34.44 1.1.4

Release date: 25 November 2016

- Redesigned the footer area to better describe Tabbycat and to promote donations and related projects

- Slight tweaks to the site homepage and main menus to better accommodate the login/log out links
- A few minor bug fixes and improvements to error reporting

34.45 1.1.3

Release date: 15 September 2016

- Fixed bug affecting some migrations from earlier versions
- Made latest results show question mark rather than crash if a team is missing
- Fixed bug affecting the ability to save motions
- Fixed bug preventing draw flags from being displayed

34.46 1.1.2

Release date: 14 September 2016

- Allow panels with even number of adjudicators (with warnings), by giving chair the casting vote
- Removed defunct person check-in, which hasn't been used since 2010
- Collapsed availability database models into a single model with Django content types
- Collapsed optional fields in action log entries into a single generic field using Django content types
- Added better warnings when attempting to create an elimination round draw with fewer than two teams
- Added warnings in Edit Database view when editing debate teams
- Renamed «AIDA pre-2015» break rule to «AIDA 1996»

34.47 1.1.1

Release date: 8 September 2016

- Fixed a bug where the team standings and team tab would crash when some emoji were not set

34.48 1.1.0 (Egyptian Mau)

Release date: 3 September 2016

- Added support for the United Asian Debating Championships style
- Added support for the World Schools Debating Championships style
- Made Windows 8+ Emoji more colourful
- Fixed an incompatibility between Vue and IE 10-11 which caused tables to not render
- Minor bug fixes and dependency updates

34.49 1.0.1

Release date: 19 August 2016

- Fixed a minor bug with the visual importer affecting similarly named institutions
- Fixed error message when user tries to auto-allocate adjudicators on unconfirmed or released draw
- Minor docs edits

34.50 1.0.0 (Devon Rex)

Release date: 16 August 2016

Redesigned and redeveloped adjudicator allocation page

- Redesigned interface, featuring clearer displays of conflict and diversity information
- Changes to importances and panels are now automatically saved
- Added debate «liveness» to help identify critical rooms—many thanks to Thevesh Theva
- Panel score calculations performed live to show strength of voting majorities

New features

- Added record pages for teams and adjudicators
- Added a diversity tab to display demographic information about participants and scoring

Significant general improvements

- Shifted most table rendering to Vue.js to improve performance and design
- Drastically reduced number of SQL queries in large tables, *e.g.* draw, results, tab

Break round management

- Completed support for break round draws
- Simplified procedure for adding remarks to teams and updating break
- Reworked break generation code to be class-based, to improve future extensibility
- Added support for break qualification rules: AIDA Australs, AIDA Easters, WADL

Feedback

- Changed Boolean fields in AdjudicatorFeedbackQuestion to reflect what they actually do
- Changed «panellist feedback enabled» option to «feedback paths», a choice of three options
- Dropped «/t/» from tournament URLs and moved «/admin/» to «/database/», with 301 redirects
- Added basic code linting to the continuous integration tests
- Many other small bug fixes, refactors, optimisations, and documentation updates

34.51 0.9.0 (Chartreux)

Release date: 13 June 2016

- Added a beta implementation of the break rounds workflow

- Added venue constraints, to allow participants or divisions to preferentially be given venues from predefined groups
- Added a button to regenerate draws
- Refactored speaker standings implementation to match team standings implementation
- New standings metrics, draw methods, and interface settings for running small tournaments and division-based tournaments
- Improved support for multiple tournaments
- Improved user-facing error messages in some scenarios
- Most frontend dependencies now handled by Bower
- Static file compilation now handled by Gulp
- Various bug fixes, optimisations, and documentation edits

34.52 0.8.3

Release date: 4 April 2016

- Restored and reworking printing functionality for scoresheets/feedback
- Restored Edit Venues and Edit Matchups on the draw pages
- Reworked tournament data importers to use csv.DictReader, so that column order in files doesn't matter
- Improved dashboard and feedback graphs
- Add separate pro speakers tab
- Various bug fixes, optimisations, and documentation edits

34.53 0.8.2

Release date: 20 March 2016

- Fixed issue where scores from individual ballots would be deleted when any other panel in the round was edited
- Fixed issue where page crashes for URLs with «tab» in it but that aren't recognized tab pages

34.54 0.8.1

Release date: 15 March 2016

- Fixed a bug where editing a Team in the admin section could cause an error
- Added instructions on how to account for speakers speaking twice to docs
- Venues Importer wont show VenueGroup import info unless that option is enabled

34.55 0.8.0 (Bengal)

Release date: 29 February 2016

- Upgraded to Python 3.4, dropped support for Python 2
- Restructured directories and, as a consequence, changed database schema
- Added Django migrations to the release (they were previously generated by the user)
- Migrated documentation to [Read The Docs](#)
- New user interface design and workflow
- Overhauled tournament preferences to use [django-dynamic-preferences](#)
- Added new visual data importer
- Improved flexibility of team standings rules
- Moved data utility scripts to Django management commands
- Changed emoji to Unicode characters
- Various other fixes and refinements

34.56 0.7.0 (Abyssinian)

Release date: 31 July 2015

- Support for multiple tournaments
- Improved and extensible tournament data importer
- Display gender, region, and break category in adjudicator allocation
- New views for online adjudicator feedback
- Customisable adjudicator feedback forms
- Randomised URLs for public submission
- Customisable break categories
- Computerised break generation (break round draws not supported)
- Lots of fixes, interface touch-ups and performance enhancements
- Now requires Django 1.8 (and other package upgrades)

Contributions are welcome, and are greatly appreciated! Every little bit helps, and credit will be given. While at its core Tabbycat is a software project, you do not need to know how to code or use Git in order to help. We welcome feedback and ideas based on your tabbing experience and appreciate suggestions or proposals for how to improve the wording, translation, and design of our interface and documentation.

Feel free to [join our Facebook group](#) if you have any questions about how to get started.

35.1 Feedback and ideas

These can be added as issues in the [GitHub repository](#); posts in our [Facebook group](#); or as an *email to the developers*.

35.2 Bug reports

Please report bugs by opening a new issue in our [GitHub repository](#). It is most helpful if you can include:

- How Tabbycat was installed (on Heroku, locally on macOS, *etc.*)
- Any details about your tournament and setup that might be helpful in troubleshooting
- Detailed steps for how to reproduce the bug

35.3 Getting started with development

- To easily test your changes to Tabbycat you probably want a working *local install* (without using Docker)
- Please submit pull requests for features and bug fixes against *develop* (but not *master*).
- We broadly use the [git-flow workflow](#).
- We use Django's testing tools — adding unit tests to new features is greatly appreciated

- A number of our tests use [Selenium](#) and [ChromeDriver](#) to simulate in-browser functionality. They will fail if you do not have the Chrome browser and ChromeDriver installed.
- A number of extra dependencies are required for running tests, linting, and serving the documentation. These can be installed with:

```
$ pip install -r 'config/requirements_development.txt'
```

- Our `package.json` provides a convenience command that runs a standard set of development tools simultaneously, such as the Django server and the automatic recompilation with live injecting of javascript and CSS. Once you have set `USE_WEBPACK_SERVER=True` in your `settings_local.py` you can then run this with:

```
$ npm run serve
```

35.4 Generating test data

There are management commands to help developers quickly generate data for use in testing, including results and feedback. A list of all commands can be found from `django help`, but the most useful in this context are:

- `django importtournament (minimal8team | australis24team | bp88team)`, which imports participant data for the 8-team (`minimal8team`), 24-team *Australis* (`australis24team`) and 88-team *BP* (`bp88team`) demonstration tournaments respectively.
- `django simulaterounds ROUND [ROUND ROUND ...]`, which simulates all of the rounds specified, generating a draw, an adjudicator allocation and a complete set of random results (but not feedback).
- `django generatefeedback ROUND [ROUND ROUND ...]`, which randomly generates feedback for all existing debates in the specified rounds.
- `django generateresults ROUND [ROUND ROUND ...]`, which randomly generates results for all existing debates in the specified rounds. (You don't need to run this if you ran `simulaterounds`, because that already does it.)

Rounds can be specified by sequence number (`seq`) or abbreviation. You can find more information about each of them by adding `--help` after the command name.

35.5 Database schema changes

When adding new features, it may be necessary to modify the database schema to support these new additions. After the changes are made, the migration files made by `python manage.py makemigrations` must also be committed. The migration files should also contain methods fill the new fields based on existing data if possible.

Fixture files (found under `data/fixtures/`) may also need to be updated, which can be done by running the `migrate_fixtures.py` script under a unmigrated database, then committing the result.

```
$ python data/migrate_fixtures.py develop (your branch)
```

35.6 Style guide

For the front end interface design there is a style guide available at [«/style/»](#) once a tournament has been setup.

For python code, we use [flake8](#) to check for a non-strict series of style rules. Warnings will trigger a Travis CI build to fail. The entire codebase can be checked by using:

```
$ flake8 .
```

For stylesheets, we use [stylelint](#). The relevant code can be checked by using:

```
$ npm run lint-sass
```

For javascript, we use [eslint](#) to enforce the [standardJS](#) style and the standard recommendation of the vue plugin for eslint. The relevant code can be checked by using:

```
$ npm run lint-vue
```

35.7 Versioning convention

Our convention is to increment the minor version whenever we add new functionality, and to increment the major version whenever:

- the database can't be migrated forwards using `python manage.py migrate --no-input`, or
- there is a major change to how the tournament workflow goes, or
- we make some other change that is, in our opinion, significant enough to warrant a milestone.

We write [data migrations](#) to allow existing systems to be upgraded easily. However, we don't always support backward database migrations. Our expectation is that long-lived installations keep up with our latest version.

One day, we hope to have a public API in place to facilitate the integration with other debating tournament software, like registration or adjudicator feedback systems. If and when that happens, we'll probably revise this convention to be more in line with [Semantic Versioning](#).

Starting from version 0.7.0, we use cat breeds as the code names for major versions.

35.8 Documentation

Documentation is created using [Sphinx](#) and hosted at [Read The Docs](#). Pushes to `develop` will update the *latest* documentation set, while pushes to `master` will update the *stable* documentation set.

To preview the documentation locally, install the development dependencies and then start the server:

```
$ sphinx-autobuild docs docs/_build/html --port 7999
```

You should then be able to preview the docs at 127.0.0.1:7999.

35.9 Project structure

- `bin` contains a number of convenience scripts for starting/stopping Docker, and the webserver/asset pipeline.
- `data` contains the sample data sets and fixtures used to setup demo tournaments and in automated tests respectively
- `docs` contains our document source files and images (although some are linked from the root directory)
- **tabbycat** is the main directory containing the Django project

- `locale` contains translation strings for shared templates (others are in respective app directories)
 - `templates` contains shared html templates, stylesheets, javascript source files, and Vue.js components/mixins.
 - `utils` contains shared utilities
 - All other folders are the Django apps that contain specific views, models, and templates for functions such as draw generation/display, or recording results. Each has sub-folders for tests and templates.
- In the root directory there are a number of files defining our python and javascript dependencies, core configuration files, and key documents like the `README`

35.10 Internationalization/Localization

The `gettext` framework is used to enable the translation of strings in Python files and Django templates. Backend in this context signifies these types of files.

The backend's translation files can be updated from the `tabbycat` directory using one or more of the supporting language codes (see `settings.py`):

```
$ dj makemessages -l es
```

To do more than one language, just specify `-l` multiple times, *e.g.* `-les -lar`.

These can then be compiled using:

```
$ dj compilemessages -l es
```

As it stands Heroku needs the `.mo` files pre-compiled (see [issue in Heroku Python buildpack](#), so these are committed to Git. Note that the English (`en`) language files should not be compiled; their sole purpose is to provide a source language for [Crowdin](#).

Strings defined in Vue files must similarly be marked with `gettext` but must be added manually to `tabbycat/locale/LANGUAGE_CODE/djangojs.po`, for each language supported. These can then be compiled to javascript bundles using:

```
$ dj compilemessages -l es          # or whichever language(s) you want to update
$ dj compilejsi18n -l es
```

These are then also committed to git to save users needing to run `compilejsi18n` during setup. The resulting files are then bundled as part of the npm build task. Updating these translations in development (live-reload) requires the use of the `cp-i18n` npm task.

35.11 Release checklist

1. Check that all migrations have been generated and committed into Git
2. Bump version number in `docs/conf.py`
3. Bump version number and (if applicable) codename in `tabbycat/settings/core.py`
4. Update the main `CHANGELOG.rst` file (including release date)
5. **Check the major current deployment options, including:**
 1. The `deploy_heroku.py` script

2. The Tabbykitten version
3. Docker (macOS, Windows 10*) and Docker Toolbox (Windows 10 Home) methods
4. Using Bash and Powershell on Windows
5. Using Terminal on macOS (at least test out a fresh install of the npm/pip dependencies)
6. Check that the last Travis CI build passed and run the full local test suite (this will include the Selenium tests that are not on Travis)
7. Shift remaining issues from the Github Milestone
8. Create and finish the release branch as per git-flow
9. Ensure the tag is correct (vX.Y.Z) and published to GitHub
10. Back-merge `master` to the `kitten` branch
11. Back-merge `develop` to the in-progress feature branches
12. Issue a formal release with change notes on GitHub
13. Post change notes on the Facebook page/group

Licence Information

We haven't released Tabbycat under an open-source licence, so there is no formal and general right to use this software. Nonetheless, you're welcome to freely use Tabbycat to help run a debating tournament if it is a not-for-profit and not-for-fundraising activity. A voluntary donation of A\$1 (Australian dollar) per team would be greatly appreciated, and would help us meet costs and justify our ongoing work and support for Tabbycat users. (A donation link is in the footer of your Tabbycat site.)

36.1 Use at for-profit or fundraising tournaments

If you use Tabbycat at a tournament where any individual or organisation is making a profit or raising funds (other than to cover the direct costs of the tournament), you or your tournament **must** make a donation of A\$1 (Australian dollar) per team to the Tabbycat maintenance team, as outlined in each tournament's donation page (the link is in the footer of your Tabbycat site). This includes instances where organisations run tournaments partly as fundraising activities and applies even if the organisation itself is a non-profit.

While we suggest that non-profit, non-fundraising tournaments budget for a donation of A\$1 per team, it is not required in order for such tournaments to use Tabbycat.

36.2 Modifications to and redistributions of Tabbycat

We grant you permission to modify and/or redistribute Tabbycat, provided that

- you do not receive any payment for this modification and/or redistribution,
- if you modify Tabbycat, you add prominent notices stating that you modified it,
- all references and functionality relating to donations are kept intact, and
- this licence page and all attributions of authorship are included as-is.

Modifying Tabbycat for payment, or using a version of Tabbycat that has been modified for payment, is strictly prohibited without our express permission.

If you use a version of Tabbycat that has been modified by you or a third party to run a for-profit or fundraising tournament, the abovementioned donation of A\$1 per team is still required.

36.3 Disclaimer of warranty and liability

We work on this software in our spare time and make it available for not-for-profit use in the hope that it will benefit the debating community. **It is provided «as is», without any warranty of any kind, express or implied, including without limitation any warranties of merchantability and fitness for a particular purpose, and we disclaim all legal liability. By using Tabbycat, you agree that none of its contributors will be held liable for any loss or claim arising directly or indirectly from using the software, or for any loss or claim otherwise in connection with the software.**

Tournament History

A partial list of major national and international tournaments (that we know of) which have used Tabbycat.

37.1 2017

- APU Philosophy CHallenge
- ANU Spring IV
- Australian British Parliamentary Debating Championships
- Australasian Wom*ns Debating Championships
- Asia World Schools Debating Championship
- Asia British Parliamentary Championship
- Borneo British Parliamentary Championship
- Canadian British Parliamentary Championship
- Cambridge Asian Schools BP Championship
- Cambodia United Asian Debating Championship
- China British Parliamentary Championships
- Japan British Parliamentary Championships
- Kings College London IV
- Kuala Lumpur Open Challenge
- KPU Pro Ams
- Malaysian National Intervarsity Debating Championship
- Melbourne Mini
- New Zealand Easters 2017

- New Zealand British Parliamentary Debating Championship
- North East Asia Open
- Swords Mini-Gong
- Shanghai Asian Schools British Parliamentary Debate Championships
- UCD Law Society IV
- UCD Vice President's Cup
- UQ Australs
- UT MARA Open
- Victoria Cup IV
- Western Sydney BP Championship
- Wollongong Easters
- Yale IV

37.2 2016

- Sydney Easters
- Joynt Scroll 2016
- Malaysia Debate Open 2016
- New Zealand Easters 2016
- Perth Australs
- Thailand United Asian Debating Championships
- The National Law School Debate
- The Khazak National Schools Debating Championship

37.3 2015

- Bali United Asian Debating Championships
- Daejeon Australs
- Joynt Scroll
- Malaysia Debate Open 2015
- New Zealand Easters 2015
- The National Law School Debate
- UNSW/UTS Easters

37.4 2014

- Joynt Scroll 2014
- NTU United Asian Debating Championships
- Otago Australs

37.5 2012

- Wellington Australs

37.6 2010

- Auckland Australs