Setting up the plugin using auto setup

Steps:

1. You will need the latest version of TSWVote. Some parts of this guide may not hold true with older versions.
2. Install the TSWVote plugin. To do this, you will need to place the plugin DLL file into the `ServerPlugins` folder.
3. Enable the REST API. To do this, you will need to manually edit the TShock config file, named `config.json`, located in the `tshock` folder. Locate these lines: `"RestApiEnabled": false, "RestApiPort": 7878`. Make sure `"RestApiEnabled"` is set to `true`, with no quotes around `true`. Make sure `"RestApiPort"` is set to a port of your liking (again with no quotes around the value), but not your server's main port. Beware that as far as we know, TShock will not automatically create a port-forward rule for this port. Making sure your networking environment allows you to use this port on the open internet is outside the scope of this guide. If you're trying to host your server at home, search the internet for a guide on `port-forwarding`. If you're with a hoster that specializes in Terraria servers or game servers, consult their documentation or tech support on whether you can use an extra port for REST API, which port it can be, and whether any extra action is needed to make it open to the internet.
4. You can now (re-)start your server. Once you load into a world, type `/tswautosetup` into the server's console, or as superadmin on the server. If you neither have access to the console nor to a superuser account, then your account's group must have the `vote.autosetup` permission.
5. If the command ran smoothly, you will be told that the token was generated and can be found in the `tshock` folder, which is the same folder that contains tshock's config file that you edited earlier. You will need to restart your server again for the changes to take effect. The text file containing the token is named `tswtoken.txt`. Whether to delete the file after you collected the token is entirely up to you, but beware that running the command again will create a new token and destroy the old one. If you're somehow obstructed from accessing this file, the only other place you can find this token is in the tshock config file, under the `ApplicationRestTokens` section.
6. If you didn't register your server on TServerWeb yet or didn't register at all, do that before continuing. Log into tserverweb.com, proceed to your account, and for the server you're working on, select `Action` -> `Edit Server`. Go to the `tShock Config` tab. Here, you will first need to enter the REST API port that you've specified in your tshock config file earlier. Then, proceed to put the token from the `tswtoken.txt` file into the `Current Server Token` field. Leave the Username and Password fields blank. Press `Save All Changes`.
7. This step is not required if you only plan to use TSWConsole and not TSWVote. You may remove the TSWVote plugin if in-game voting is not something you want. To use in-game voting functionality, you also need to run the `/tserverweb` command, specifying your server's ID on TServerWeb. There are two ways to get the ID of your server. The first way is to make sure your server is currently running, then navigate to the `Manage server` page of the server you're working on. If TServerWeb says that your server is offline, try pressing `Force Server Status Update`. If that didn't help, you may need to check your network settings, as mentioned in step 2. Then navigate to the `Server Plugins` tab, press `Manage` under `In-game Voting`. Once the page loads, you will be able to find a tiny writing on the right, saying `Your Server ID: XXXX`. Take this number, and then run `/tserverweb XXXX` (`XXXX` being the number) as console or superadmin. With no console and superuser access, your account's group must have the

`vote.changeid` permission. After all that, you can also take some time on this same page to set up what happens upon voting. The other way of retrieving server ID is to navigate to either `Edit Server` or `Manage Server` and then just click onto your browser's address bar, then locate the digits after `?sid=`. Then you just run the command, as described above. If all of this seems too complicated, you can instead run `/group addperm tserverweb vote.changeid`. This will hand this step over to TServerWeb to handle, but if somebody steals your token, they will be able to break in-game voting on your server. You will also have to navigate to the `In-game voting` page anyway.

8. *(Optional)* By default, TServerWeb's access to your server will be limited to the minimum required for the operation of our two plugins and not beyond that. This mostly includes the ability to see info about users (their group and IP address), and the ability to execute commands that require no additional permissions. If you wish to use the advanced administrative capabilities of TServerWeb, you will need to manually add the required permissions to the `tserverweb` group, by running the `/group addperm` command. Refer to TShock's documentation on permissions.
9. *(Optional)* Try out our TSWConsole plugin. It will enable TServerWeb to present you with a live view of your server's console. After installation, it will require no further setup beyond what you already did in this guide.

The config file

When you start your server with the plugin installed, it will immediately create a config file under the name of `TSWVote.json` in the `tshock` folder. The contents of this file will look something like this:

```
{
  "ServerID": 0,
  "NumberOfWebClients": 30,
  "Timeout": 10000,
  "RequirePermission": false,
  "PermissionName": "vote.vote"
}
```

Let's break down the available options.

- `ServerID` is the ID of your server at TServerWeb. You can set it up in-game or in the console, using the `/tserverweb` command. You can read more about this in the auto-setup guide.
- `NumberOfWebClients` is the maximum number of connections that your server will make to TServerWeb at any given time. Under normal circumstances, this will not need to exceed the number of player slots on your server, and lower values might work fine too. The default is `30`. The value does *not* need quotation marks `""` around it.
- `Timeout` is the number of milliseconds (one second is 1000 milliseconds) that the plugin will wait for to get a response from TServerWeb. If this is too low, you will get errors even when a vote gets placed successfully. If you're getting any errors mentioning something timing out, try increasing this value. The default is `10000` (which is 10 seconds). The value does *not* need quotation marks `""` around it.
- `RequirePermission` is whether voting on your server will require the player's group to have a certain permission. The default is `false`. The other option is `true`. The value does *not* need quotation marks `""` around it.
- `PermissionName` is the permission that players will need to vote. This only takes effect if `RequirePermission` is set to `true`. The default is `vote.vote`. The value *does* need quotation marks `""` around it.

If you're creating the config file ahead of time even before you've installed the plugin, be aware of the following things:

- On non-Windows operating systems, filenames are often case-sensitive. Make sure the filename is exactly `TSWVote.json`, with the first 4 letters being upper-case.
- The names of every config option do need quotation marks `""` around them, as they're considered a string-type value in JSON. It would be best on your part to just copy the example presented above and fill your values in.

Commands and permissions

The TSWVote plugin has the following commands:

- `/tserverweb` is the command used to setup the server ID and for TServerWeb's back-end to verify that you've installed the plugin. The two different permissions to access this command are `vote.changeid` and `vote.ping` with the latter only having access to displaying the server ID and not actually changing it. You can read more on this here.
- `/tswversioncheck` is the command that displays the current version of the plugin. You may use it to check your plugin's version against the latest available update. The permission to use this command is `vote.checkversion`.
- `/voteclear` is the command that can be used to clear vote-states of specific players, as well as all players. In other words, it can allow players to attempt placing their votes earlier than the plugin would otherwise let them, which could happen if their previous attempt failed. Usage examples: `/voteclear` will clear your own vote-state; `/voteclear all` will clear everyone's vote-states; `/voteclear Alice` will clear the vote-state of the player named `Alice`; `/voteclear 127.0.0.1` will clear the vote-state associated with the `127.0.0.1` IP address. The permission to use this command is `vote.admin`.
- `/tswautosetup` is the command used in the process of setting up the plugin. Read more on that here. The permission to use this command is `vote.autosetup`.

You can find TShock's guide on permissions here.

Manual setup of a REST Application token

This guide is here to provide an alternative to steps 3 and 4 of the TSWVote auto-setup guide. It demonstrates how you can set up a REST Application token without using the `/tswautosetup` command or indeed the TSWVote plugin at all. You may refer to it as a general guide on creating Application REST tokens to use with other plugins as well. If you are setting up TSWVote or TSWConsole, you should still follow the auto-setup guide, but replace steps 3 and 4 with the steps from this guide.

1. Make sure you run with TShock's console or superadmin permissions. Running the commands below under a restricted group will likely be impossible.
2. First, you will need to start your server and create a group that will have all the permissions you plan to give to your REST API application token. You can do so with the `/group add` command. Like so: `/group add tserverweb tshock.rest.useapi,tshock.rest.users.info,tshock.rest.command,vote.ping,AdminRest.allow`. The permissions required for the basic operation of our two plugins are included in the above command, where `tserverweb` is the new group's name and the rest are permissions separated by commas. It is at this stage that you may give TServerWeb extra permissions, but you can always add more permissions using `/group addperm` as well. Refer to TShock's guide on permissions to learn more.
3. The second step is to create a user and make it a member of the new group. This user's credentials will not be used or exposed to any outside application. You may give this user a randomized username and must give this user a secure randomized password to avoid any abuse or other security concerns. The password's recommended length would be 16 characters. A convenient source of randomness would be random.org's password tool. If you're in a Linux terminal environment, you may also try this command `< /dev/urandom tr -dc A-Za-z0-9 | head -c16;echo;` instead. You must keep the username, and you can shred or discard the password after you've used it in the command below. Because the password won't be used for anything, you need not bother to make it mnemonic, easy to remember, or otherwise human-readable. To create this user, you run `/user add` like this: `/user add tserverwebXXXX PASSWORD GROUP`, where `XXXX` is a randomized postfix/appendix to the username, `PASSWORD` is your random password, and `GROUP` is the group that you've created in step 1 (which was `tserverweb` in our example). Beware that if you intend to use TSWVote's autosetup command later, it may destroy any users with username beginning with `tserverweb` and belonging to group `tserverweb`, as well as delete the group `tserverweb` if no users belonging to it are found. It will also destroy any API tokens that you create in step 3 if their (the token's, not the user's) group is `tserverweb`.
4. Now, you will need to generate a secure token and then manually add it to TShock's config file. You may take one of the passwords that random.org generated for you and use that as a token, but make sure it's different from the user's password in the previous step. Alternatively, if you're a developer, you may find it convenient to use a mnemonic sequence when you're working with the API manually. Open the `config.json` file in the `tshock` folder and locate the `ApplicationRestTokens` section: `"ApplicationRestTokens": {}`. If there's nothing between the brackets, you will need to make some space for your token by pressing Enter after the opening bracket a couple of times. If there are already tokens present, add a comma after the closing bracket of the last token, and then make some space. Then, copy and paste this:

```
 "TOKEN": {
   "Username": "USERNAME",
   "UserGroupName": "GROUP"
 },
```

- Replace `TOKEN` with the token value you decided to use, `USERNAME` with the username you made in step 2, and `GROUP` with the group you made in step 1. In this step, you may also enable REST API and select the port for it. For more info on this, refer to step 2 of this other guide.

4. Now you can save the file and attempt to (re-)start your server. JSON is a pretty sensitive data structure and if you did anything incorrectly, you may have to come back and fix it before TShock can start properly. Make sure your config looks something like this:

```
"RESTRequestBucketDecreaseIntervalMinutes": 1,
"ApplicationRestTokens": {
  "TOKEN": {
    "Username": "USERNAME",
    "UserGroupName": "GROUP"
  }
}
```

After this step, your REST application token is finally set up and you may continue following the autosetup guide, resuming at step 5, and assuming that any reference to the token acquired from the `tswtoken.txt` file is actually referring to the token you've just created.

Using the VoteSuccess hook (for plugin developers)

`VoteSuccess` is the hook that allows your plugins to listen for successful in-game votes.

Add `TSWVote.dll` to your project's references.

Add to the top of your .cs file: `using TSWVote;`

Assuming that `TSWVote()` will be your event handler function, add to your `Initialize` or `OnInitialize`:

```
try
{
        VoteHooks.VoteSuccessRegister(TSWVote);
}
catch
{

}
```

Add to your Dispose:

```
try
{
        VoteHooks.VoteSuccessDeregister(TSWVote);
}
catch
{

}
```

Your function will need to accept a single argument of type `VoteSuccessArgs`:

```
private void TSWVote(VoteSuccessArgs e)
{

}
```

`VoteSuccessArgs` has two child fields: `Player` is the `TSPlayer` object for the player that voted successfully, and `Container` is of type `object` and can be used in cases when your two or more separate event handlers need to communicate.