# Tripal 4.x Documentation

***Release 4.x.alpha.1***

**Tripal Project Management Committee with the help of the Tripal C**

**Apr 12, 2024**

# CONTENTS:

This documentation is meant to guide users, admin and developers in the usage and customization of Tripal 4.

CHAPTER

ONE

# INSTALL TRIPAL

The recommended method to get involved in Tripal 4 development is through a Tripal Docker installation. This most closely mimics the environment automated tests are run on.

## 1.1 Requirements

- Drupal (see supported versions below)
- Drupal core modules: Search, Path, Views, and Field.
- PostgreSQL 12+
- PHP 8 (tested with 8.1, 8.2, 8.3)
- Apache 2+
- Composer 2+
- UNIX/Linux

**Note:** Apache 2+ is recommended as the webserver but Drupal is also known to work well with Nginx. You may have limited support for Nginx from the Tripal community.

PostgreSQL is required by Chado to function properly, rather than MySQL or any other database management system.

### 1.1.1 Supported Drupal Versions

The following table shows the current status of automated testing on the versions of Drupal we currently support.

| Drupal | 10.0.x | 10.1.x | 10.2.x |
|---|---|---|---|
| **PHP 8.1** | | | |
| **PHP 8.2** | | | |
| **PHP 8.3** | | | |

## 1.2 Traditional Installation

These instructions assume that your webserver's root directory is `/var/www` and that your site will be installed into a directory called `tripal4`. Please change these in the commands below if your configuration is different.

They also assume that your system meets all the prerequisites for running a Drupal site. Refer to the Requirements page for more information.

### 1.2.1 Install Prerequisites

1. If you are starting from a clean operating system installation, you will first need to install some needed programs. First make sure the operating system is up to date.

   ```
   sudo apt-get update
   sudo apt-get dist-upgrade
   ```

   And restart if requested.

2. Install PHP

   ```
   sudo apt-get install php-dom php-gd php-curl php-cli
   php --version
   ```

3. Install Apache2 web server. Note the PHP version from step 2 above, and adjust the version of PHP here if necessary.

   ```
   sudo apt-get install apache2 libapache2-mod-php
   sudo a2enmod rewrite php8.1
   sudo systemctl restart apache2
   ```

   At this point, you should be able to open a browser on your installation system and view the Apache2 Default Page at http://localhost

4. Install Composer

   ```
   sudo apt-get install composer
   ```

5. Install PostgreSQL database engine

   ```
   sudo apt-get install postgresql php-pgsql
   sudo su - postgres
   createuser -P drupal
   ```

   and supply a password. Now, while we are still the `postgres` user, give the drupal user the permissions it will need

```
psql
alter role drupal with login replication createdb;
ALTER DATABASE "template1" SET bytea_output = 'escape';
\q
exit
```

### 1.2.2 Install Drupal

1. Install Drupal using Composer. Composer is now the recommended way to install and manage Drupal, extension modules, and other dependencies. Detailed information can be found on Drupal's documentation: Using composer to install Drupal and manage dependencies.

   A. Navigate to your webserver's root directory and prepare a directory. You may not have permission to create the directory here, so set it up first using sudo

   ```
   cd /var/www
   sudo mkdir tripal4
   sudo chown $USER:$USER tripal4
   ```

   B. Run the composer command to install a fresh copy of Drupal 10 into the `tripal4` directory:

   ```
   composer create-project drupal/recommended-project /var/www/tripal4
   ```

   (If you are presented with this question, you can respond with a *y*: *Do you want to remove the existing VCS (.git, .svn..) history? [Y,n]?*)

   This should leave you with a `tripal4` directory that looks something like this:

   ```
   tripal4
   ├── composer.json
   ├── composer.lock
   ├── vendor
   └── web
   ```

   The `vendor` directory is where many of the dependencies like drush (see below) are installed.

   The `web` directory is the actual webroot for Drupal. This should be the directory that is served by your webserver. **The two composer files and the `vendor` directory should not be publicly accessible.**

2. Install Drush and other required modules, also with composer, ensuring that you are within your new `tripal4` directory:

   ```
   cd /var/www/tripal4/
   composer require drush/drush drupal/field_group drupal/field_group_table
   ```

3. Drupal may complain about permissions on certain files, as well as generating a configuration file from the template provided by Drupal. The files in question must be readable and writable by the webserver's user, as well as yourself. If you're using Apache, this is typically `www-data` and for Nginx, it is commonly `nginx`. Read more about Drupal's requirements here: Administering a Drupal site - security in Drupal, or run the following commands to satisfy them:

```
# Make sure you are in the web root:
cd /var/www/tripal4/web

# Create the files directory:
mkdir sites/default/files

# Copy Drupal's configuration template:
cp sites/default/default.settings.php sites/default/settings.php

# Set permissions, assuming www-data is your web user (Apache). If
# necessary, you can determine the Apache username as follows:
apachectl -S | grep User
# example output is  User: name="www-data" id=33 not_used

# Using the user name `www-data` or whatever it may be, change ownership as␣
↪follows:
sudo chown www-data:$USER sites/default/files
sudo chown www-data:$USER sites/default/settings.php
```

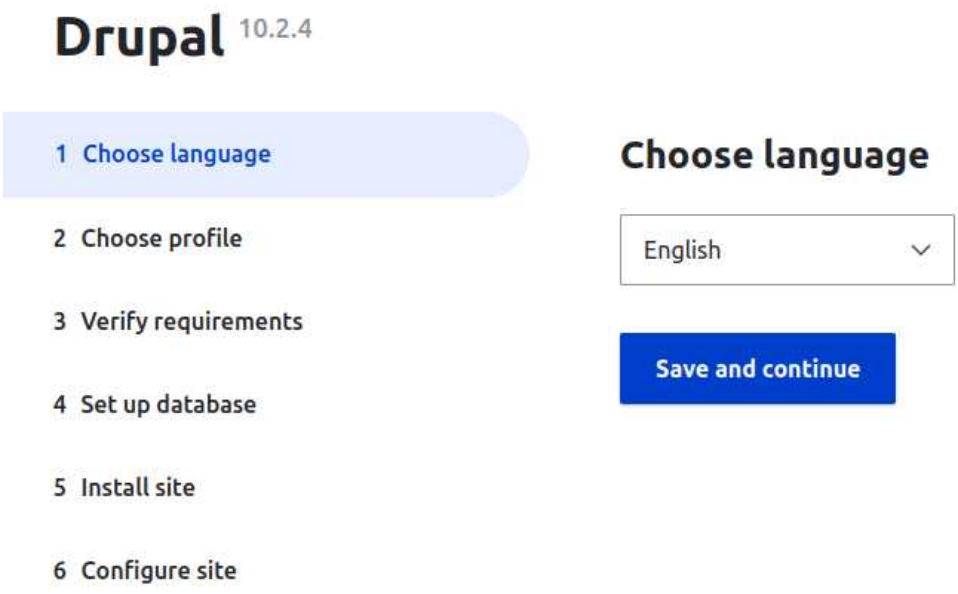
4. Configure Apache to allow access to our install location `/var/www/tripal4` so that it will show up as `http://localhost:/tripal4`. Use your preferred editor and, with sudo, edit `/etc/apache2/sites-available/000-default.conf` and make the following additions somewhere inside the `<VirtualHost *:80>` section.

```
Alias /tripal4 "/var/www/tripal4/web"

<Directory /var/www/tripal4/web>
   AllowOverride All
</Directory>
```

   After saving these changes, restart Apache

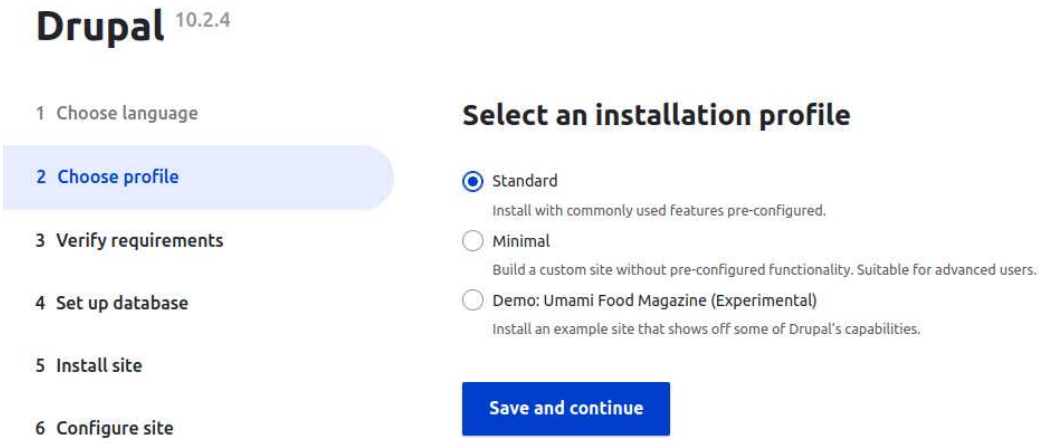
```
sudo systemctl restart apache2
```

5. Navigate to your new site in your browser: `<siteaddress.com>/tripal4/core/install.php` and follow the instructions for setting up a Drupal site. The first page you should appear similar to this:

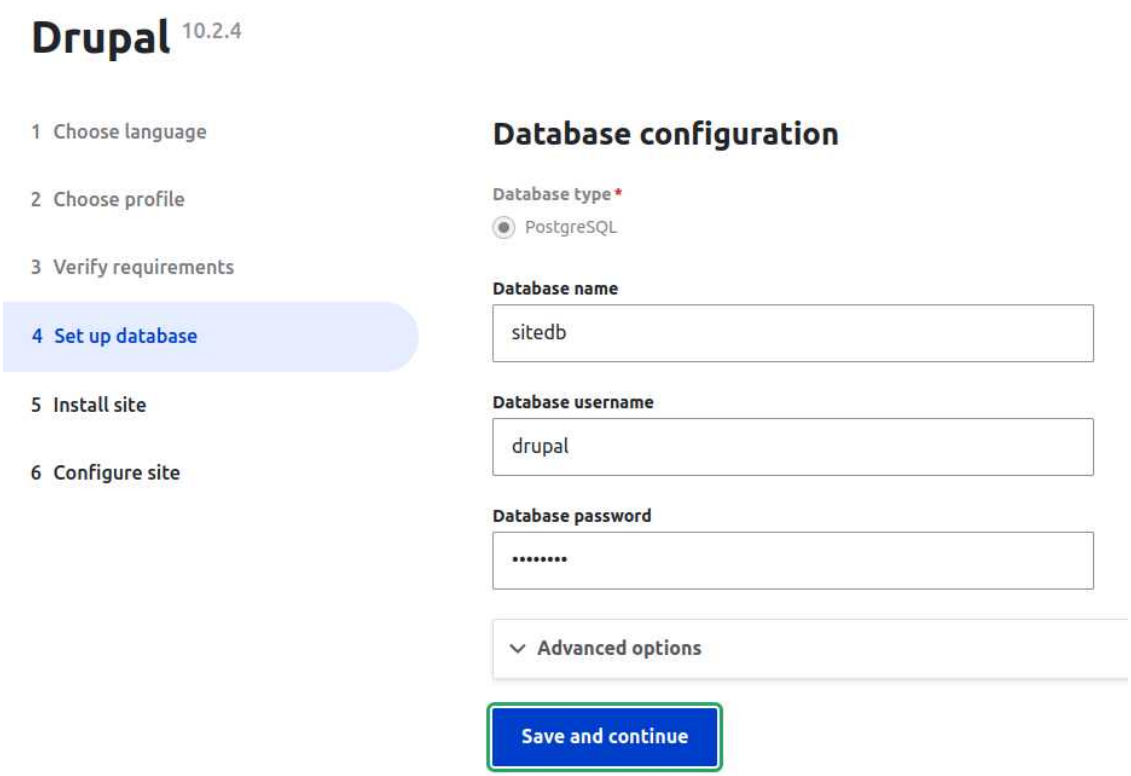



Select your preferred language and continue.

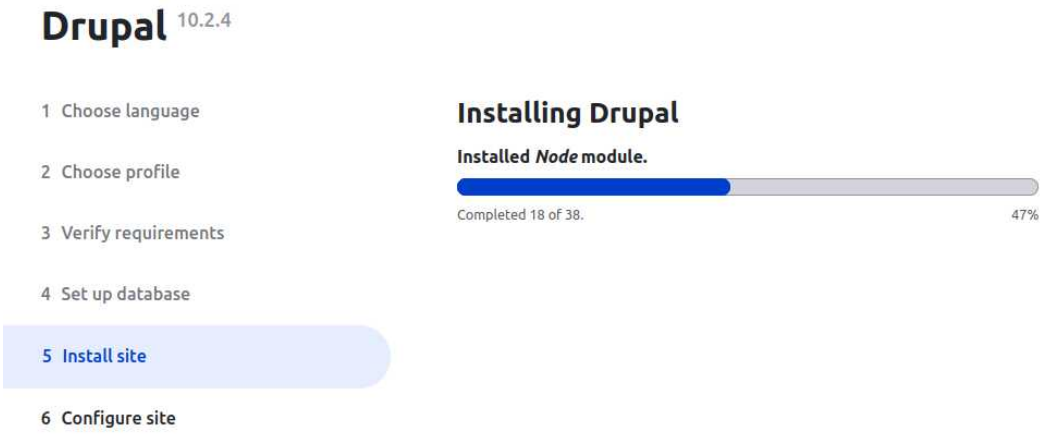
6. For the installation profile select **Standard**, and continue.



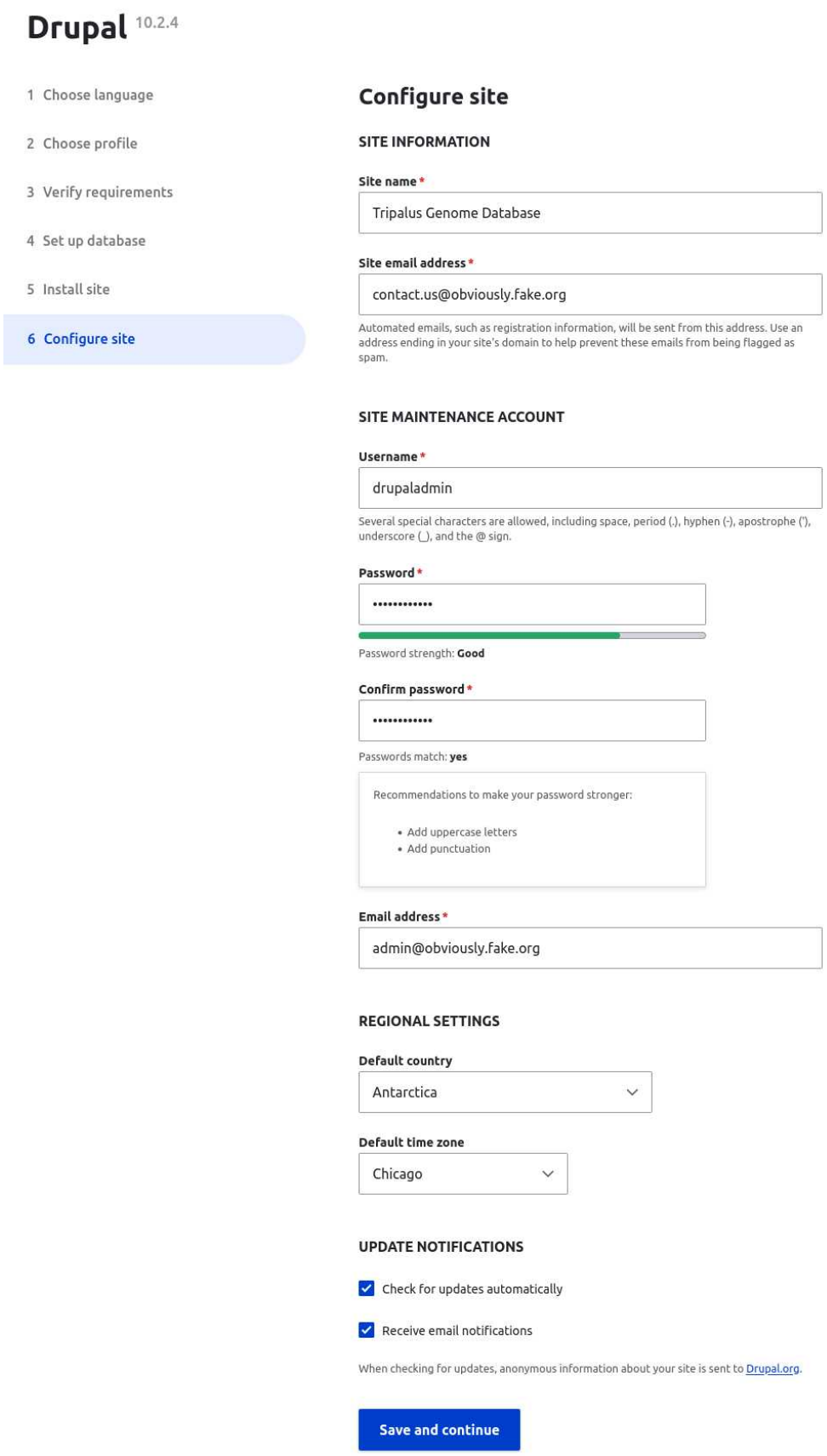

7. If all requirements are met, step 3 should be skipped automatically.
8. **In step 4, you will be asked to provide credentials for a database user. Postgres is required for Chado, and therefore it is strongly recommended to use a Postgres database for Tripal.**
   Detailed information on creating a Postgres database and user account can be found here: Getting started - installing Drupal. For the **Database name** you can use whatever you like. For example `sitedb`. The **Database username** `drupal` and **Database password** must be the same as the ones you provided earlier in prerequisite step #5.



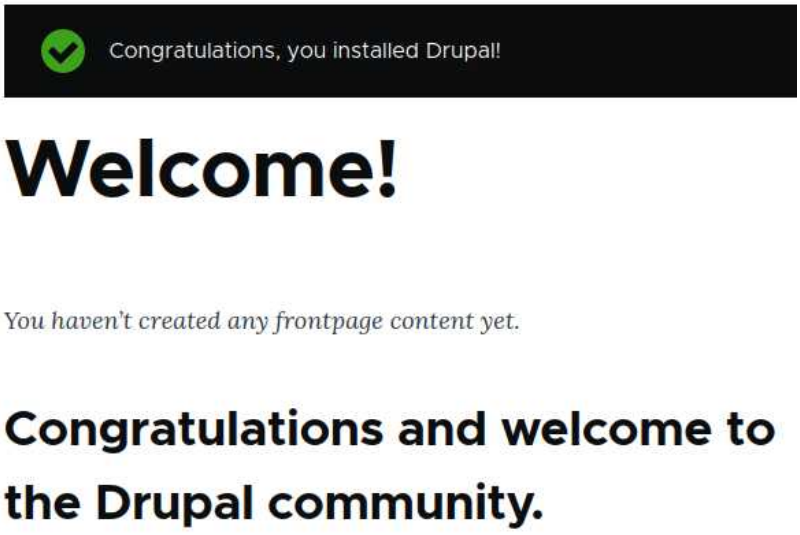

9. For step 5, installation of Drupal should begin, with progress shown similar to this.




10. For step 6, you will need to configure your site. An example is presented below, enter appropriate information for your site.


Drupal 10.2.4
1 Choose language
2 Choose profile
3 Verify requirements
4 Set up database
5 Install site
6 Configure site
Configure site
SITE INFORMATION
Site name *
Tripalus Genome Database
Site email address *
contact.us@obviously.fake.org
Automated emails, such as registration information, will be sent from this address. Use an address ending in your site's domain to help prevent these emails from being flagged as spam.
SITE MAINTENANCE ACCOUNT
Username *
drupaladmin
Several special characters are allowed, including space, period (.), hyphen (-), apostrophe ('), underscore (_), and the @ sign.
Password *
Password strength: Good
Confirm password *
Passwords match: yes
Recommendations to make your password stronger:
Add uppercase letters
Add punctuation
Email address *
admin@obviously.fake.org
REGIONAL SETTINGS
Default country
Antarctica
Default time zone
Chicago
UPDATE NOTIFICATIONS
Check for updates automatically
Receive email notifications
When checking for updates, anonymous information about your site is sent to Drupal.org.
Save and continue

12. You should then see a screen similar to this.




### 1.2.3 Install Tripal

1. We need to first add the Tripal module. There are two options, depending on how you will use your site. If you are installing a production site, or are just trying out Tripal, use method "**A**". If you are a developer you should use method "**B**".

   A. **Production or testing installation.** To just use the most recent **stable** version of tripal install this way:

   ```
   cd /var/www/tripal4/
   composer require tripal/tripal
   ```

   To install the most recent **development** version:

   ```
   cd /var/www/tripal4/
   composer require tripal/tripal:4.x-dev
   ```

   To install a specific released version, find the tag in the Tripal release page, and install it like this:

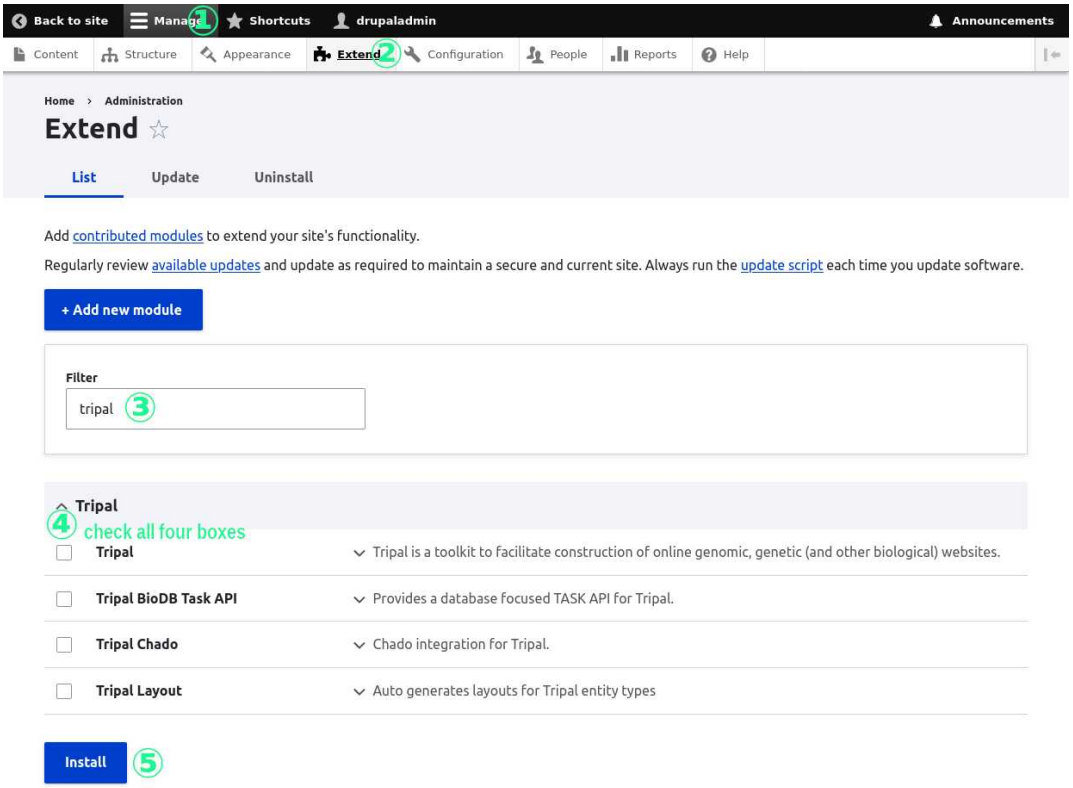
   ```
   cd /var/www/tripal4/
   composer require tripal/tripal:4.0-alpha2
   ```

   B. **Developer installation.** Clone the Tripal repository in your `web/modules` directory.

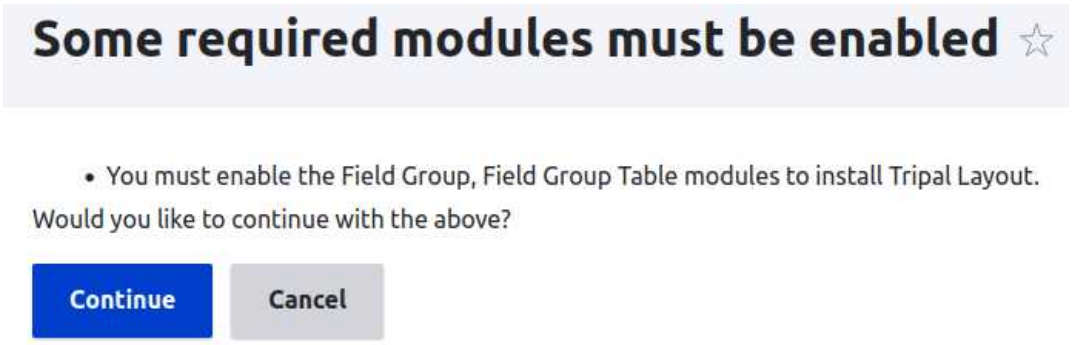
   Note: Within the `modules` directory, you may create your own custom directory to store other extension modules.

   ```
   cd /var/www/tripal4/web/modules/
   git clone https://github.com/tripal/tripal.git
   # or if you have a GitHub account configured
   git clone git@github.com:tripal/tripal.git
   ```

2. Enable Tripal in your site using the Administration Toolbar: Manage > Extend




Select "Continue" to also install "Field Group" and Field Group Table"




If successful you will see:

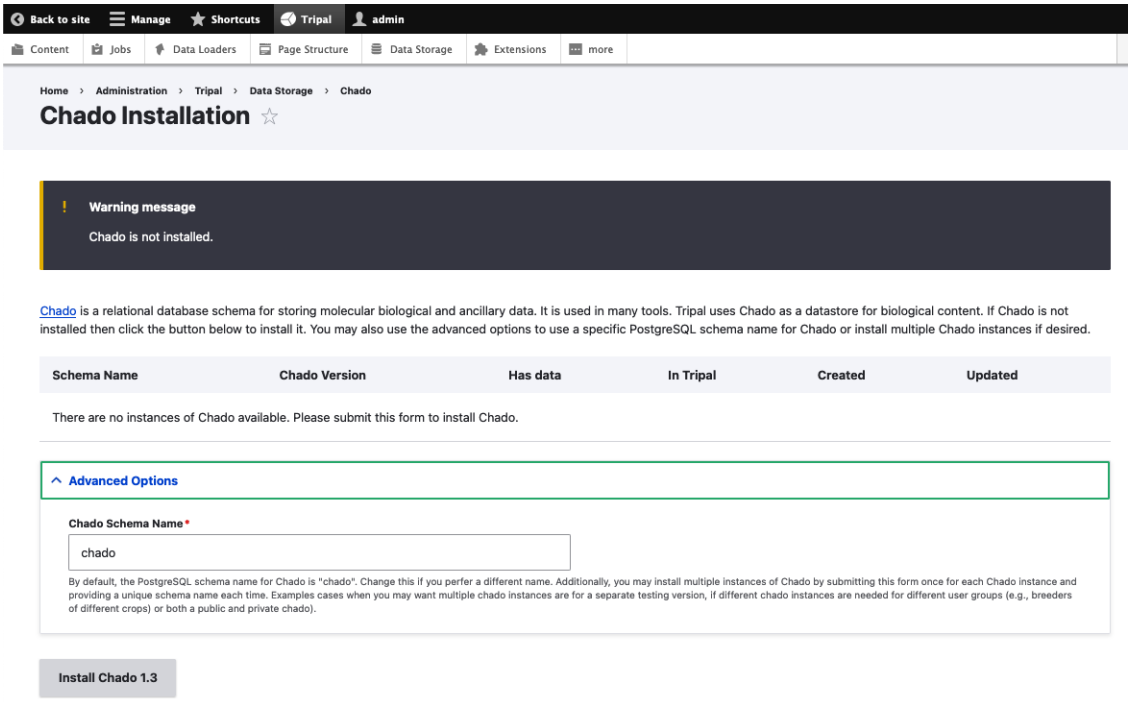

3. Use Drush to rebuild the cache so that Tripal menu items appear correctly.

```
/var/www/tripal4/vendor/bin/drush cache-rebuild
```

### 1.2.4 Install and Prepare Chado

The site is not quite ready to use yet! The Chado schema must be installed and the site must be prepared to use the installation.

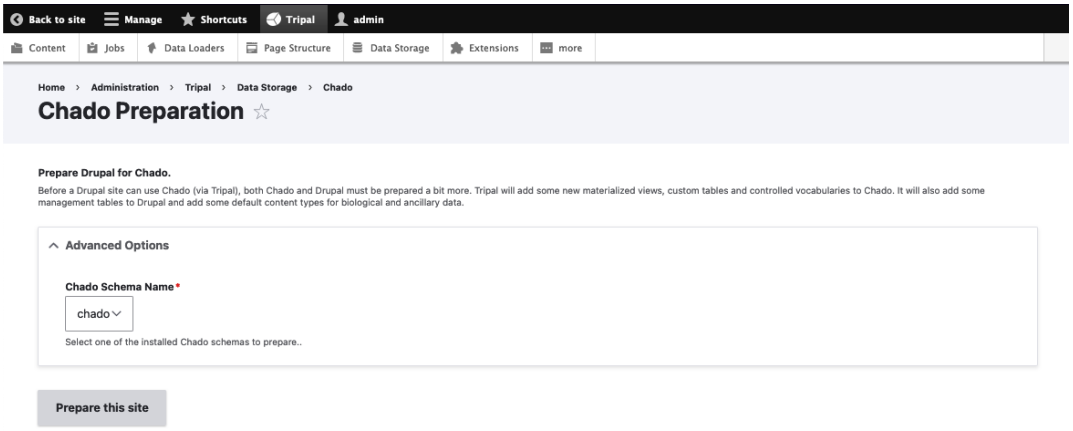
1. **On your site, navigate to Tripal → Data Storage → Chado → Install Chado**
   The page should warn you that Chado is not installed. Use this form to install it. If you wish, you can provide a custom name to your Chado schema:




2. Click "Install Chado 1.3". You will be prompted to use Drush to trigger the installation of Chado. This must be done on the command line:

```
/var/www/tripal4/vendor/bin/drush trp-run-jobs --username=drupaladmin --
↪root=/var/www/tripal4/web
```

3. Once Chado is installed, the site must be further prepared. Navigate to **Tripal → Data Storage → Chado → Prepare Chado**



4. Click "Prepare this site", and like before, run the supplied Drush command:

```
/var/www/tripal4/vendor/bin/drush trp-run-jobs --username=drupaladmin --
↪root=/var/www/tripal4/web
```

Congratulations, you now have a freshly installed Tripal 4 site with Chado as the storage back end. The next step is *Building your Site*

## 1.3 Tripal Docker

Tripal Docker is currently focused on Development, Debugging, and Unit Testing. There will be a production focused Tripal Docker soon.

### 1.3.1 Software Stack

**Currently we have the following installed:**

- Debian Bullseye(11)
- PHP 8.2.17 with extensions needed for Drupal (Memory limit 1028M)
- Apache 2.4.56
- PostgreSQL 16.2 (Debian 16.2-1.pgdg110+2)
- Composer 2.7.2
- Drush 12.5.1.0
- Drupal 10.2.5-dev downloaded using composer (or as specified by drupalversion argument).
- Xdebug 3.2.1

### 1.3.2 Quickstart

1. Run the image in the background mapping its web server to your port 9000.

   a) Stand-alone container for testing or demonstration.

```
docker run --publish=9000:80 --name=t4 -tid tripalproject/
↪tripaldocker:latest
```

   b) Development container with current directory mounted within the container for easy edits. Change my_module with the name of yours.

```
docker run --publish=9000:80 --name=t4 -tid --volume=$(pwd):/var/www/drupal/
↪web/modules/contrib/my_module tripalproject/tripaldocker:latest
```

2. Start the PostgreSQL database.

```
docker exec t4 service postgresql start
```

#### Development Site Information:

| | |
|---|---|
| URL | http://localhost:9000 |
| Administrative User | drupaladmin |
| Administrative Password | some_admin_password |

### 1.3.3 Usage

- Run Drupal Core PHP Unit Tests:

```
docker exec --workdir=/var/www/drupal/web/modules/contrib/tripal t4 phpunit
```

- Open PSQL to query the database on the command line. The password is docker.

```
docker exec -it t4 psql --user docker sitedb
```

- Run Drush to generate code for your module!

```
docker exec t4 drush generate module
```

- Run Drush to rebuild the cache

```
docker exec t4 drush cr
```

- Get version information:

```
docker exec t4 drush status
docker exec t4 php -v
docker exec t4 psql --version
docker exec t4 apache2 -v
```

- Run Composer to upgrade Drupal

```
docker exec t4 composer up
```

### 1.3.4 Detailed Setup for Core Development

If you want to contribute to Tripal development, you will likely want to create a branch to work on and create a Docker using this branch, or on another contributor's branch. For instructions on how to do this, see the section *Creating a Docker for Testing*

### 1.3.5 Troubleshooting

#### The provided host name is not valid for this server.

On my web browser, I got the message "The provided host name is not valid for this server".

**Solution:** It is most likely because you tried to access the site through a URL different from `localhost` or `127.0.0.1`. For instance, if you run docker on a server and want to access your d8t4 site through that server name, you will have to edit the settings.php file inside the docker (at the time writing this, it would be every time you (re)start the docker)

and change the last line containing the parameter `$settings[trusted_host_patterns]`. This file by default is read-only, so you will first need to change permissions to allow editing:

```
docker exec -it t4 chmod +w /var/www/drupal/web/sites/default/settings.php
docker exec -it t4 vi /var/www/drupal/web/sites/default/settings.php
```

For instance, if your server name is `www.yourservername.org`:

```
$settings[trusted_host_patterns] = [ '^localhost$', '^127\.0\.0\.1$', '^www\.
↪yourservername\.org$', ];
```

#### Not seeing recent functionality or fixes.

As Tripal 4 is currently under rapid development, this could be due to not using the most up to date docker image available. The following instructions can be used to confirm you are using the most recent image.

```
docker rm --force t4
docker rmi tripalproject/tripaldocker:latest
docker pull tripalproject/tripaldocker:latest
```

At this point, you can follow up with the appropriate `docker run` command. If your run command mounts the current directory through the `--volume` parameter then make sure you are in a copy of the t4 repository on the main branch with the most recent changes pulled.

### 1.3.6 Debugging

#### Xdebug: Overview

There is an optional Xdebug configuration available for use in debugging Tripal 4. It is disabled by default. Currently, the Docker ships with three modes available:

**Develop**
: Adds developer aids to provide "better error messages and obtain more information from PHP's built-in functions".

**Debug**
: Adds the ability to interactively walk through the code.

**Profile**
: Adds the ability to "find bottlenecks in your script and visualize those with an external tool".

To enable Xdebug, issue the following command:

```
docker exec --workdir=/var/www/drupal/web/modules/contrib/tripal t4 xdebug_toggle.sh
```

This will toggle the Xdebug configuration file and restart Apache. You should use this command to disable Xdebug if it is enabled prior to running PHPUnit Tests as it seriously impacts test run duration (approximately 8 times longer).

There is an Xdebug extension available for most modern browsers that will let you dynamically trigger different debugging modes. For instance, profiling should only be used when you want to generate profiling data, as this can be quite compute intensive and may generate large files for a single page load. The extension places an interactive Xdebug icon in the URL bar where you can select which mode you'd like to trigger.

### Xdebug: Step debugging

Step debugging occurs in your IDE, such as Netbeans, PhpStorm, or Visual Studio Code. There will typically already be a debugging functionality built-in to these IDEs, or they can be installed with an extension. Visual Studio Code, for example, has a suitable debugging suite by default. This documentation will cover Visual Studio Code, but the configuration options should be similar in other IDEs.

The debugging functionality can be found in VS Code on the sidebar, the icon looks like a bug and a triangle. A new configuration should be made using PHP. The following options can be used for basic interaction with Xdebug: .. code:

```
{
  "version": "0.2.0",
  "configurations": [
      {
          "name": "Listen for Xdebug",
          "type": "php",
          "request": "launch",
          "port": 9003,
          "pathMappings": { "/var/www/drupal/web/modules/contrib/tripal": "~/Dockers/t4"␣
↪}
      }
  ]
}
```

The important parameter here is *pathMappings* which will allow Xdebug and your IDE know which paths on the host and in the Docker VM coorespond to eachother. The first path listed is the one within the Docker and should point to the Tripal directory. The seocnd path is the one on your local host machine where you installed the repo and built the Docker image. If you followed the instructions above, this should be in your user folder under *~/Dockers/t4*.

9003 is the default port and should only be changed if 9003 is already in use on your host system.

With this configuration saved, the Play button can be pressed to enable this configuration and have your IDE listen for incoming connections from the Xdebug PHP extension.

More info can be found for VS Code's step debugging facility in VS Code's documentation.

### Xdebug: Profiling

Profiling the code execution can be useful to detect if certain functions are acting as bottlenecks or if functions are being called too many times, such as in an unintended loop. The default configuration, when profiling is enabled by selecting it in the Xdebug browser extension, will generate output files in the specified directory.

To view these files, we recommend using Webgrind. It can be launched as a separate Docker image using the following command:

```
docker run --rm -v ~/Dockers/t4/tripaldocker/xdebug_output:/tmp -v ~/Dockers/t4:/host -p␣
↪8081:80 jokkedk/webgrind:latest
```

You may need to adjust the paths given in the command above, similar to when setting up the pathMappings for step debugging earlier.

CHAPTER

TWO

# BUILDING YOUR SITE

## 2.1 Anatomy of a Tripal Site

### 2.1.1 Content Types

Tripal sites host data such as organisms, analyses, genes and mRNA, and publications. In Tripal, these are known as **Content Types**. Tripal comes with a number of content types out-of-the-box, but also provides the ability to create custom content types. These content types extend the standard Drupal content types such as Article and Basic Page.

In Tripal, all content types are defined by Controlled Vocabulary (CV) terms. This has a number of advantages:

1. Facilitates sharing between Tripal sites.
2. Provides a clear indication of what content is available on your site.
3. Makes content creation more intuitive from Tripal v2 (add a "Gene" rather then a "feature").
4. Allows complete customization of what data types your site provides.
5. Integrates tightly with web services allowing Tripal to adhere to RDF specifications

#### Examples

This is a working list of content types that are currently built-in to Tripal. Some of them are not enabled by default but come in bundled modules.

- **General**
  - Analysis
  - Contact
  - Organism
  - Project
  - Protocol
  - Publication
  - Study
- **Expression**
  - Array Design
  - Assay
  - Biological Sample

- **Germplasm**
  - Breeding Cross
  - Germplasm Accession
  - Germplasm Vareity
  - Recombinant Inbred Line
- **Genomic**
  - DNA Library
  - Gene
  - Genome Annotation
  - Genome Assembly
  - Genome Project
  - mRNA
  - Phylogenetic Tree
  - Physical Map
- **Genetic**
  - Genetic Map
  - Genetic Marker
  - Heritable Phenotypic Marker
  - QTL
  - Sequence Variant

### 2.1.2 Fields

Each content type is composed of a number of datapoints, in Tripal these are **Fields**. By default, Tripal uses the Chado database schema to store data, and each field is linked to a specific table and column in Chado.

For example, the Organism content type comes by default with the following fields, and each one represents a property from Chado's definition of an organism:

| Field Name | Chado "organism" table column |
|---|---|
| Abbreviation | abreviation |
| Common Name | common_name |
| Description | comment |
| Genus | genus |
| Infraspecies | infraspecific_name |
| Infraspecific Type | infraspecific_name |
| Species | species |

Just like with Content Types, each field must also have its own Controlled Vocabulary term associated to it. If we look at the Organism example again, we have the following terms that are drawn from ontologies and their identifier:

| Field Name | Chado "organism" table column | PUMPKIN |
|---|---|---|
| Abbreviation | abreviation | local:abbreviation |
| Common Name | common_name | NCBITaxon:common_name |
| Description | comment | schema:description |
| Genus | genus | TAXRANK:0000005 |
| Infraspecies | infraspecific_name | TAXRANK:0000045 |
| Infraspecific Type | infraspecific_type | local:infraspecific_type |
| Species | species | TAXRANK:0000006 |

## 2.2 Creating Content Types

**Note:** Prior to creating a new content type you should understand the structure of Chado and how others use Chado to store similar types of data.

As mentioned in the Anatomy of a Tripal Site, all content types are defined by Controlled Vocabulary (CV) terms.

### 2.2.1 Find a Controlled Vocabulary (CV) Term

Before creating a new content type for your site you must identify a CV term that best matches the content type you would like to create. CVs are plentiful and at times selection of the correct term from the right vocabulary can be challenging. If there is any doubt about what term to use, then it is best practice to reach out to others to confirm your selection. The Tripal User community is a great place to do this by posting a description of your content type and your proposed term on the Tripal Issue Queue. Confirming your term with others will also encourage re-use across Tripal sites and improve data exchange capabilities.

The EBI's Ontology Lookup Service is a great place to locate terms from public vocabularies. At this site you can search for terms for your content type. If you can not find an appropriate term in a public vocabulary or via discussion with others then you create a new **local** term within the **local** vocabulary that comes with Tripal.

**Warning:** Creation of **local** terms is discouraged but sometimes necessary. When creating local terms, be careful in your description.

### 2.2.2 How to Add a CV Term

#### Loading From an OBO File

Once you've chosen a term to describe your content type, you may need to add the term to Tripal if it is not already present. Many CVs use the OBO file format to define their terms. If the term belongs to a controlled vocabulary with a file in OBO format then you can load all the terms of the vocabulary using Tripal's OBO Loader at **Tripal** → **Data Loaders** → **Chado Vocabularies** → **Chado OBO Loader**.

#### Manually Adding a Term

Alternatively, you can add terms one at a time. To add a single term either from an existing vocabulary or a new local term, navigate to **Tripal** → **Data Loaders** → **Chado Vocabularies** → **Manage Chado CVs** and search to see if the vocabulary already exists. If it does you do not need to add the vocabulary. If it does not exist, click the **Add Vocabulary** link to add the vocabulary for your term. Then navigate to **Tripal** → **Data Loaders** → **Chado Vocabularies** → **Manage Chado CV Terms** then click the **Add Term link** to add the term.

## 2.3 Example Genomic Site Setup

The following tutorial will walk you through creating content and loading genomic data. This is a good introduction to Tripal 4 Content Types and the new Administrative User Interface regardless of whether you intend to store genomic data in your particular Tripal 4 site.

### 2.3.1 Setup Tripal Content Types

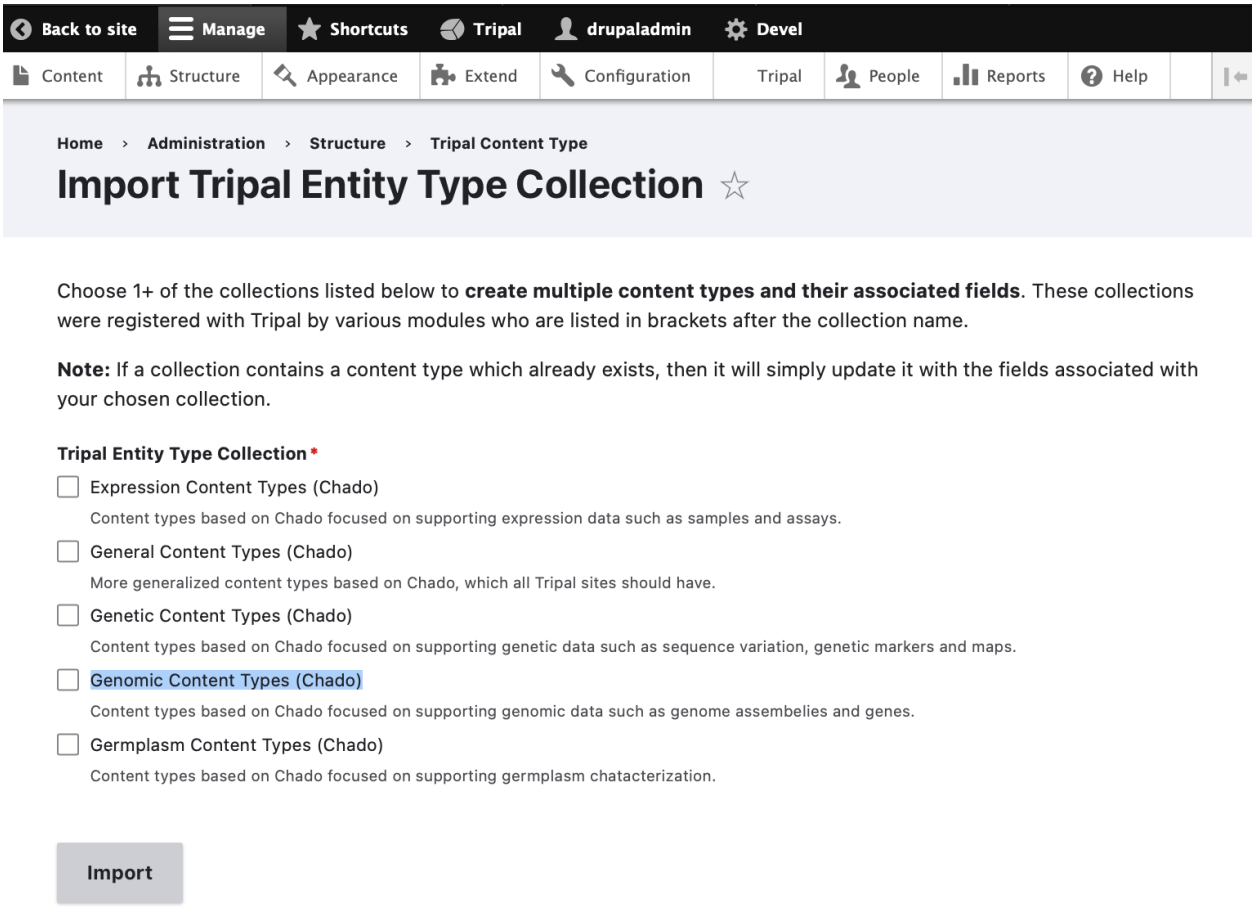
When you first install Tripal, you do not yet have any content types created. This is to provide you with flexibility to only add the content types you need for your data.

For a site containing genome assemblies, genes and associated content, you will want to import the *Genomic* content type collections. This is done by navigating to **Admin > Tripal > Page Structure** and then clicking "Import type collection" button. You want to select **Genomic Content Types (Chado)** and click on **Import**

---

**Note:** We expect in this tutorial that you already have the "General" content types for Tripal. If you don't have a number of content types already listed that say "General" in the first column when you go to the Page Structure listing then you will also want to select General in the following form.

---




Now run the submitted Tripal job from command line as follows if Drupal/Tripal is running as a web application:

```
drush trp-run-jobs --username=drupaladmin --root=/var/www/drupal/web
```

If Tripal is running from a docker container named $cntr_name, run:

```
docker exec -it $cntr_name drush trp-run-jobs --username=drupaladmin --root=/var/www/
↪drupal/web
```

You will see the following output:

```
2024-02-14 21:34:50
Tripal Job Launcher
Running as user 'drupaladmin'
-------------------
2024-02-14 21:34:50: Job ID 1.
2024-02-14 21:34:50: Calling: import_tripalentitytype_collection(Array)
[notice] Creating Tripal Content Types from: Genomic Content Types (Chado)
[notice] Content type, "Gene", created.
[notice] Content type, "mRNA", created.
[notice] Content type, "Phylogenetic Tree", created.
[notice] Content type, "Physical Map", created.
[notice] Content type, "DNA Library", created.
[notice] Content type, "Genome Assembly", created.
```

(continues on next page)

(continued from previous page)

```
[notice] Content type, "Genome Annotation", created.
[notice] Content type, "Genome Project", created.
[notice] Attaching fields to Tripal content types from: Chado Fields for Genomic Content␣
↪Types
:::
:::
:::
```

Now, when you go to **Admin > Tripal > Page Structure** you will see a Genomic Category that includes Gene and other content required for the Example Genomic site.

### 2.3.2 Create an Organism Page

Before we can load our data we must first have an organism to which the data will be associated. Chado v1.3 does not come preloaded with any organisms (although previous version of Chado do). For this tutorial we will import genomic data for Citrus sinesis (sweet orange), so we must first create the organism.

We can add the organism using the **Add Tripal Content** link in the top administrative menu or from **Content** -> **Add Tripal Content**. The Add Tripal Content page has several content types already available, including the **Organism** content type.

---

**Note:** Drupal provides its own content types such as **Article** and **Basic Page**. These content types are referred to as **nodes** in Drupal speak. You can add these content types via the **Add Content** page. Tripal v4 derived content types are separated from these Drupal content types.

---

To add a new organism click the **Organism** link and a form will appear with multiple fields. Fill in the fields with these values:

| Field Name | Value |
|---|---|
| Genus | Citrus |
| Species | sinensis |
| Abbreviation | C. sinensis |
| Common name | Sweet orange |
| Description | Sweet orange is the No.1 citrus production in the world, accounting for about 70% of the total. Brazil, Flordia (USA), and China are the three largest sweet orange producers. Sweet orange fruits have very tight peel and are classified into the hard-to-peel group. They are often used for juice processing, rather than fresh consumption. Valencia, Navel, Blood, Acidless, and other subtypes are bud mutants of common sweet orange varieties. Sweet orange is considered as an introgression of a natural hybrid of mandarin and pummelo; some estimates shows more mandarin genomic background than pummelo. The genome size is estimated at 380Mb across 9 haploid chromosomes. |

Leave all remaining fields empty and save the page. You should now have an organism page that appears as follows:

---

**Note:** The layout of the organism page is provided by the **tripal_ds** module that was enabled during Tripal installation. If you decided not to enable that module then your page will look quite different.

---


Home

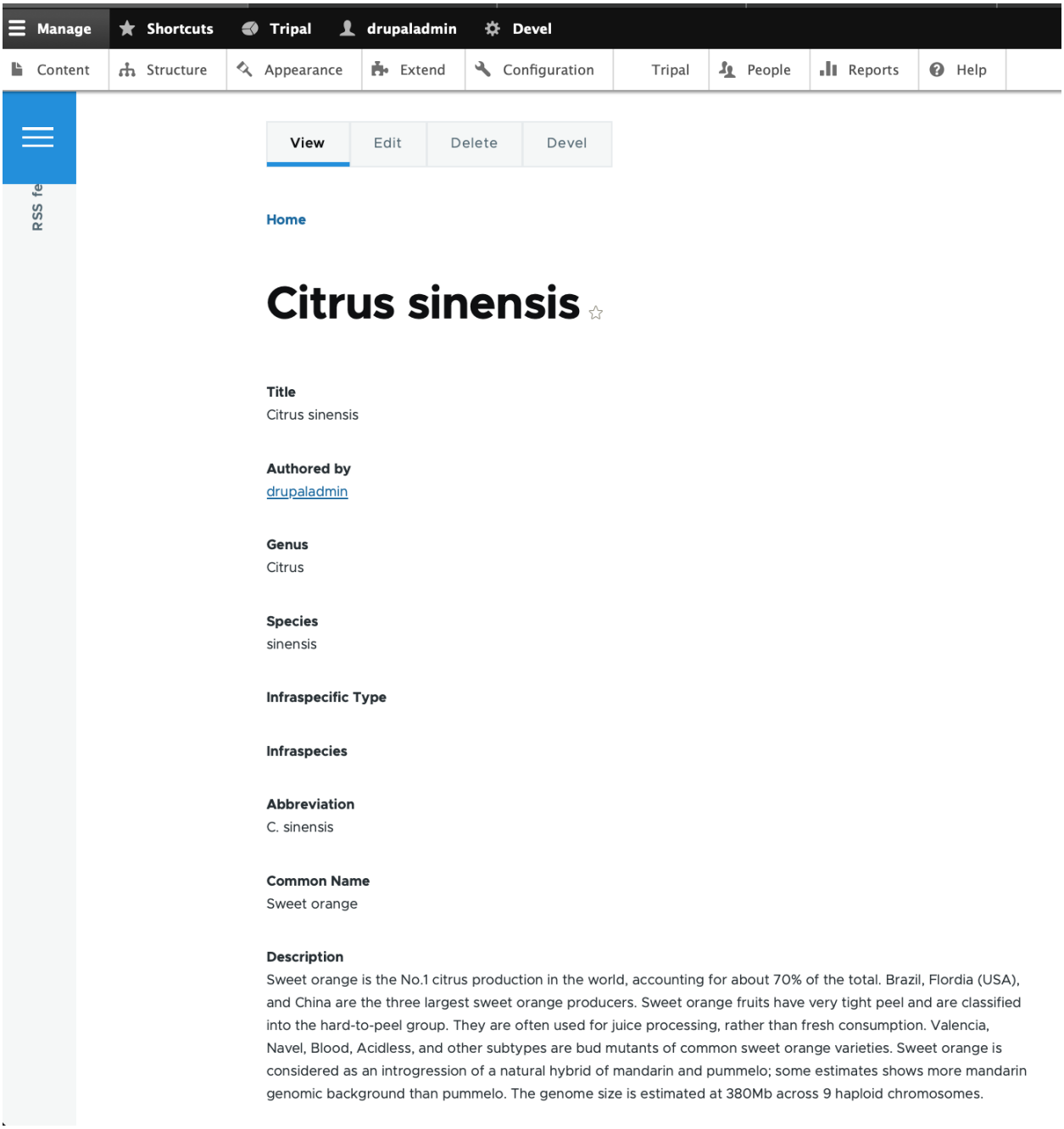
## Add Tripal Content ☆

### General

- Analysis
  Use the analysis page to for an individual analysis, workflow or pipeline that was performed using statistical or computational means.
- Contact
  Use the contage page a person or institution that can be linked as a responsible party for data or results.
- Organism
  Use the organism page for an individual living system, such as animal, plant, bacteria or virus,
- Project
  Use the project page to provide information about a project that many be linked to multiple sub components such as studies or analyses.
- Protocol
  Use the protocol page for a parameterizable description of a process.
- Publication
  Use the publication page for books, journal articles, or other citable work.
- Study
  Use the study page for a systematic investigation.

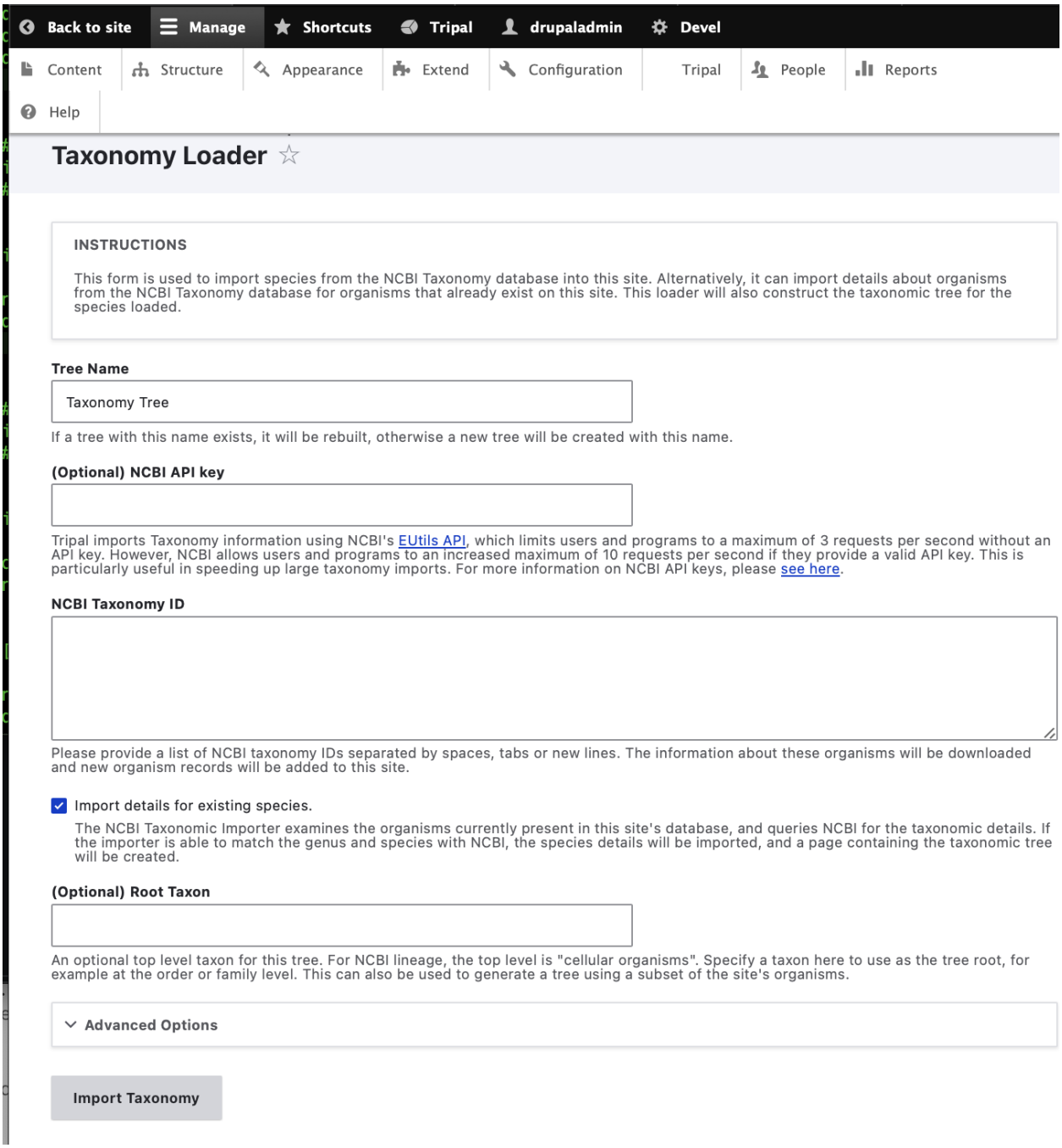
### Genomic

- DNA Library
  Use the DNA library page for a collection of DNA molecules that have been cloned in vectors.
- Gene
  Use the gene page for a region (or regions) that includes all of the sequence elements necessary to encode a functional transcript. A gene may include regulatory regions, transcribed regions and/or other functional sequence regions.
- Genome Annotation
  Use the genome annotation page for an analyses specifcally for genome annotation. Such an analysis typically involves one or more workflows of bioinormatics tools to generate the structural and functional annotations of the genome.
- Genome Assembly
  Use the genome assembly page for an analyses specifcally for genome assembly. Such an analysis typically involves one or more workflows of bioinormatics tools to generate the assembly.
- Genome Project
  Use the genome project page to provide information about a genome assembly and annotation project that many be linked to multiple sub analyses.
- mRNA
  Use the mRNA page for a messenger RNA whic is the intermediate molecule between DNA and protein. It includes UTR and coding sequences. It does not contain introns.
- Phylogenetic Tree
  Use the phylogenetic tree page for data or plotting of phylogenetic trees. Usually includes information such as topology, lengths (in time or in expected amounts of variance) and a confidence interval for each length.
- Physical Map
  Use the physical map page for a map of annotated with physical features or landmarks such as restriction sites, cloned DNA fragments, genes or genetic markers, along with the physical distances between them. Distance in a physical map is measured in base pairs. A physical map might be ordered relative to a reference map (typically a genetic map) in the process of genome sequencing.


Manage
Shortcuts
Tripal
drupaladmin
Devel
Content
Structure
Appearance
Extend
Configuration
Tripal
People
Reports
Help
View
Edit
Delete
Devel
Home
Citrus sinensis
Title
Citrus sinensis
Authored by
drupaladmin
Genus
Citrus
Species
sinensis
Infraspecific Type
Infraspecies
Abbreviation
C. sinensis
Common Name
Sweet orange
Description
Sweet orange is the No.1 citrus production in the world, accounting for about 70% of the total. Brazil, Flordia (USA), and China are the three largest sweet orange producers. Sweet orange fruits have very tight peel and are classified into the hard-to-peel group. They are often used for juice processing, rather than fresh consumption. Valencia, Navel, Blood, Acidless, and other subtypes are bud mutants of common sweet orange varieties. Sweet orange is considered as an introgression of a natural hybrid of mandarin and pummelo; some estimates shows more mandarin genomic background than pummelo. The genome size is estimated at 380Mb across 9 haploid chromosomes.

### Load data from NCBI Taxonomy

Tripal makes it easy to import additional information about any organisms within a Tripal site from the NCBI Taxonomy database. The importer will only import data for species that you currently have in the Tripal database. The taxonomic names must match those in the NCBI Taxonomy database. Currently, we only have a single organism (Citrus sinensis) and we will import additional properties for this organism from NCBI but we can return later to import data for new organisms we may add later. To import additional organism details, navigate to **Tripal** → **Data Loaders** → **Taxonomy Loader**. The following page appears:




Click the checkbox beside the 'Import taxonomy for existing species' and click Import Taxonomy. Now run the submitted job:

```
drush trp-run-jobs --username=administrator --root=/var/www/drupal/web
```

If Tripal is running from a docker container named $cntr_name,

```
docker exec -it $cntr_name drush trp-run-jobs --username=drupaladmin --root=/var/www/
↪drupal/web
```

You will see the following output:

```
024-02-14 18:58:58
Tripal Job Launcher
Running as user 'drupaladmin'
-------------------
2024-02-14 18:58:58: Job ID 2.
2024-02-14 18:58:58: Calling: tripal_run_importer(5)
Running 'Taxonomy Loader' importer
NOTE: Loading of this file is performed using a database transaction. If it fails or is␣
↪terminated prematurely then all insertions and updates are rolled back and will not be␣
↪found in the database
Initializing Tree...
Insert phylotree: Created phylotree with phylotree_id: <em class="placeholder">1</em>
Import phylotree summary: <em class="placeholder">0</em> nodes were successfully␣
↪associated to content, <em class="placeholder">0</em> nodes could not be associated
Rebuilding Tree...
Percent complete: 0%. Memory: 34,690,376 bytes.
Updating Existing...
Percent complete: 100.00 %. Memory: 34,722,768 bytes.
Percent complete: 100.00 %. Memory: 34,723,296 bytes.
Import phylotree: Associated <em class="placeholder">Citrus sinensis</em> to organism_
↪id: <em class="placeholder">1</em>
Import phylotree summary: <em class="placeholder">1</em> nodes were successfully␣
↪associated to content, <em class="placeholder">0</em> nodes could not be associated
Done.
```

Now, Click on **Tripal Content -> + Publish Tripal Content -> Content Type -> Organism** Run the jobs as mentioned earlier followed by :

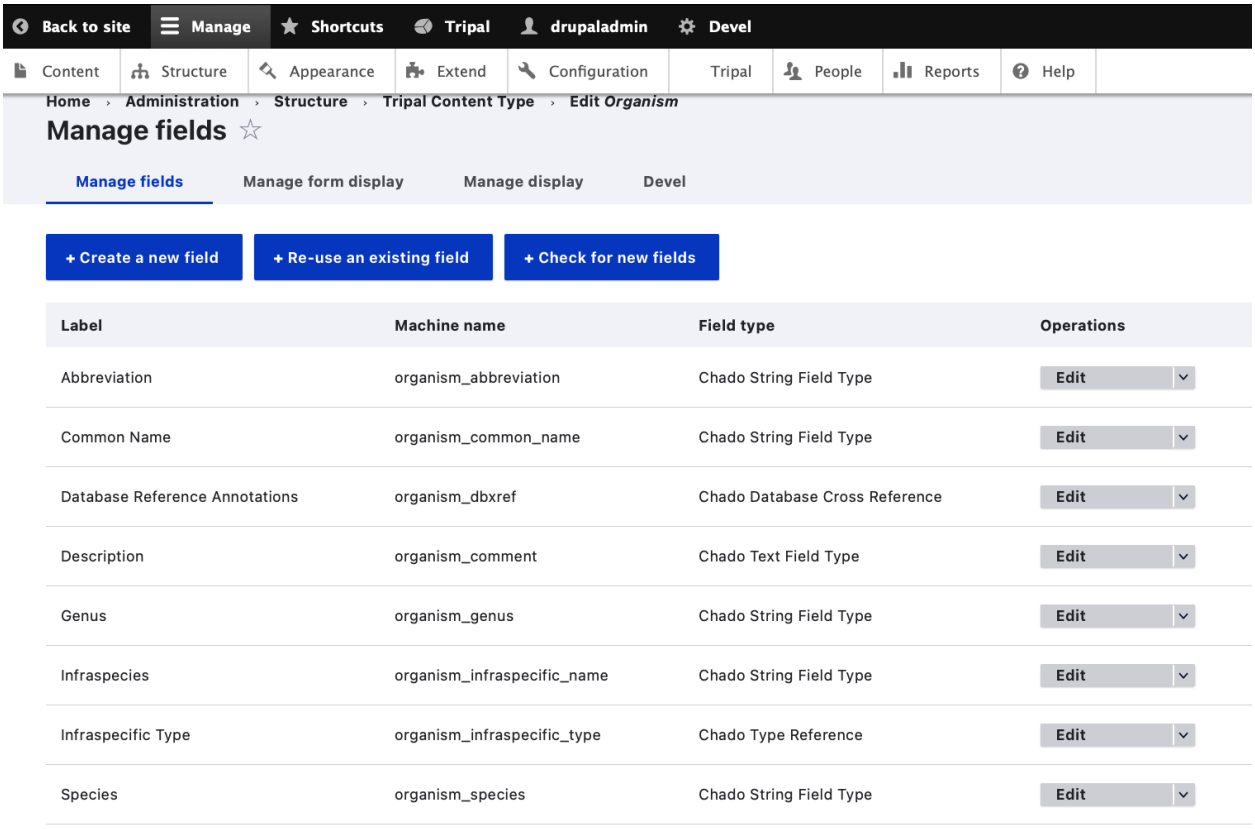
```
docker exec -it $cntr_name drush cr
```

to clear the drush cache. Now, clicking on **Tripal Content -> Organism=Citrus ** will show Taxonomy Reference Annotation of NCBI 2711 associated with Citrus sinensis. Editing of this organism also shows it.

### Adding New Fields

**Note:** This feature has not yet been implemented for Tripal v4, documentation will be added once this feature is available

We have now imported many new properties about the Citrus sinensis organism from NCBI Taxonomy. However, these properties won't show up on the page automatically. We need to tell Drupal that our organism pages now have new property fields for display. To do this, navigate to **Structure** → **Tripal Content Type** and in the row for the Organism content type, click Drop Down arrow and mouse over on **Manage Fields**. Here we see a list of fields that are associated with an Organism content type.



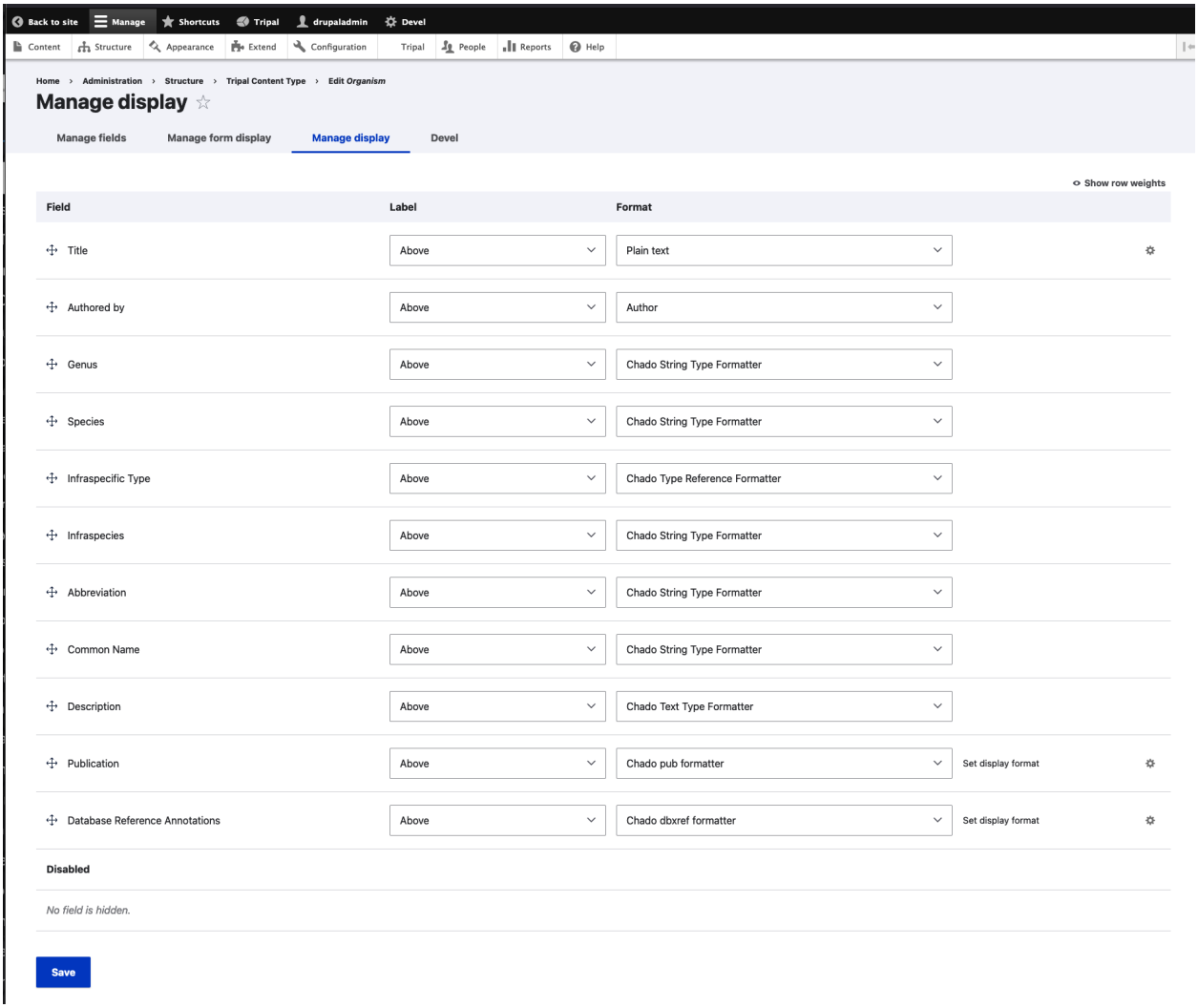

Click the link at the top of the page **+ Check for new fields**.

---

**Note:** The **Check for new fields** functionality has not yet been implemented for Tripal 4. Documentation will be added when it is available.

---

Drupal now knows about these new fields! But if we were to look at the Citrus sinensis page we would see that the new properties do not appear. Despite that Drupal knows about the fields it has disabled their display. To enable display of these fields click the **Manage Display** tab at the top right of the page. Here all of the fields are organized into the structure that they will be displayed on the page. Later in this tutorial a more formal description is provided about how you use this interface to change the way the page appears. For now, we simply need to get the new fields to be shown. Scroll to the bottom of the page and the new fields can be seen in the Disabled section.




We can move these newly created fields out of the Disabled section by clicking on the cross-hair icons to the left of the name and dragging the field into a section above. Drag these fields into the **Summary** section underneath the **Summary Table**. Notice in the screenshot below that the fields that were once in the **Disabled** section are now in the **Summary Table** section. Click the **Save** button at the bottom to make all changes final.

Now, if we return to the organism page we will see these new properties were added to the page inside of the Summary Table.

## Further Customizations

You may not like this arrangement of fields. You may prefer to place these extra fields inside of a new **pane** rather than inside of the **Summary pane**. Perhaps a pane named Additional Details. You can rearrange the order of these fields and create new panes, as desired by following the more details instructions on the Configure Page Display page of this tutorial. For example, the following shows these fields organized into a new pane named **Additional Details** which is separate from the **Summary** Pane. Note the table of contents sidebar now lists the **Summary** and **Additional Details** links. When clicked, the pane selected by the user migrates to the top of the page

Additional Resources:

- Tripal 3 reference for creating organism

**Note:** **Database Reference Annotations** that appear in the **Organism** creation page have not yet been implemented for Tripal v4, documentation will be added once this feature is available.

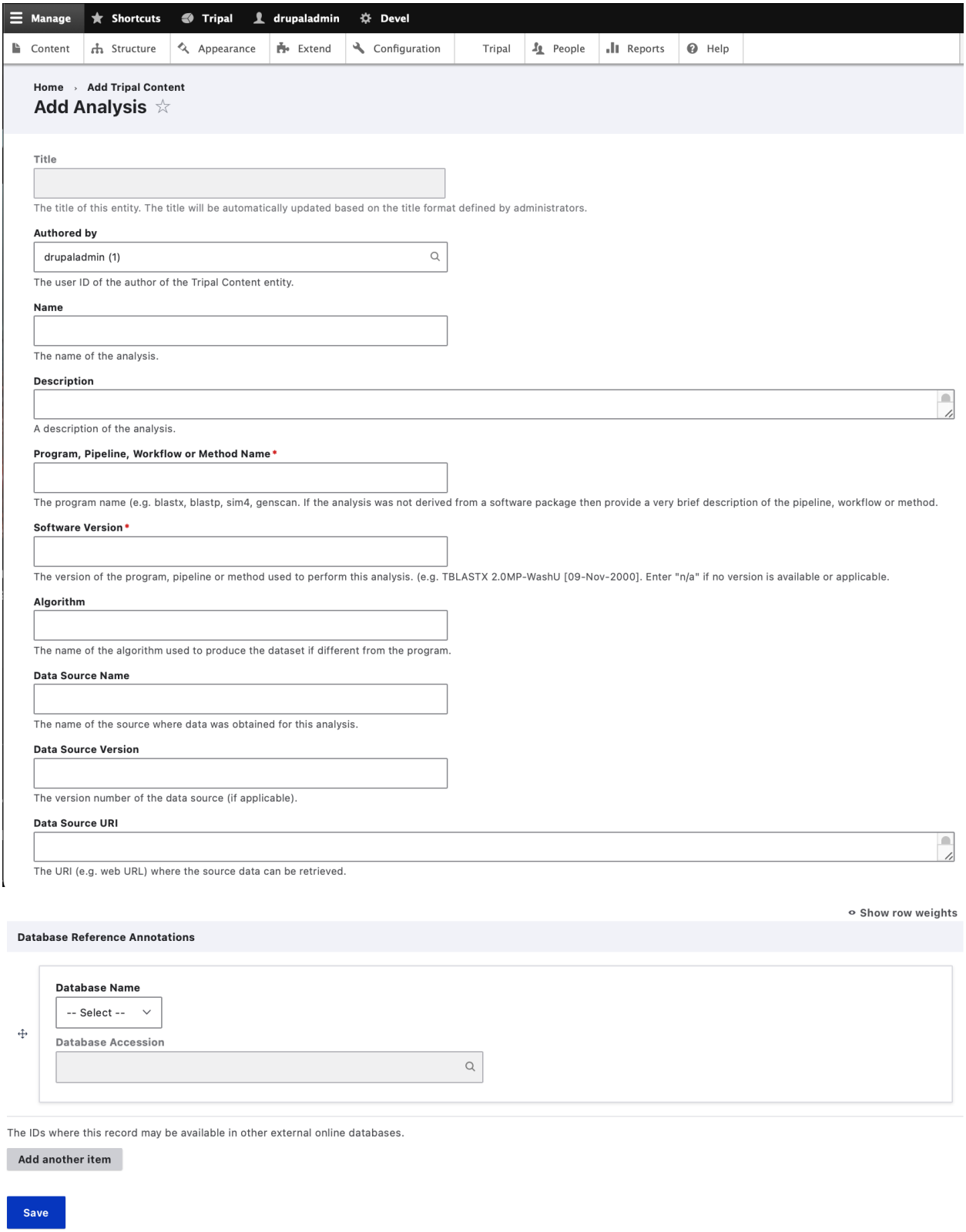
### 2.3.3 Create a Genome Assembly Page

For this tutorial we will later import a set of genes, and their associated mRNA, CDS, UTRs, etc. Tripal's Chado loader for importing genomic data requires that an analysis be associated with all imported features. This has several advantages, including:

- The source of features (sequences) can be traced. Even for features simply downloaded from a database, someone else can see where the features came from.
- Provenance describing how the features were created can be provided (e.g. whole genome structural and functional annotation description).
- The analysis associates all of the features together.

To create an analysis for loading our genomic data, navigate to the Add Tripal Content and click on the link: **Analysis**

The analysis creation page will appear:



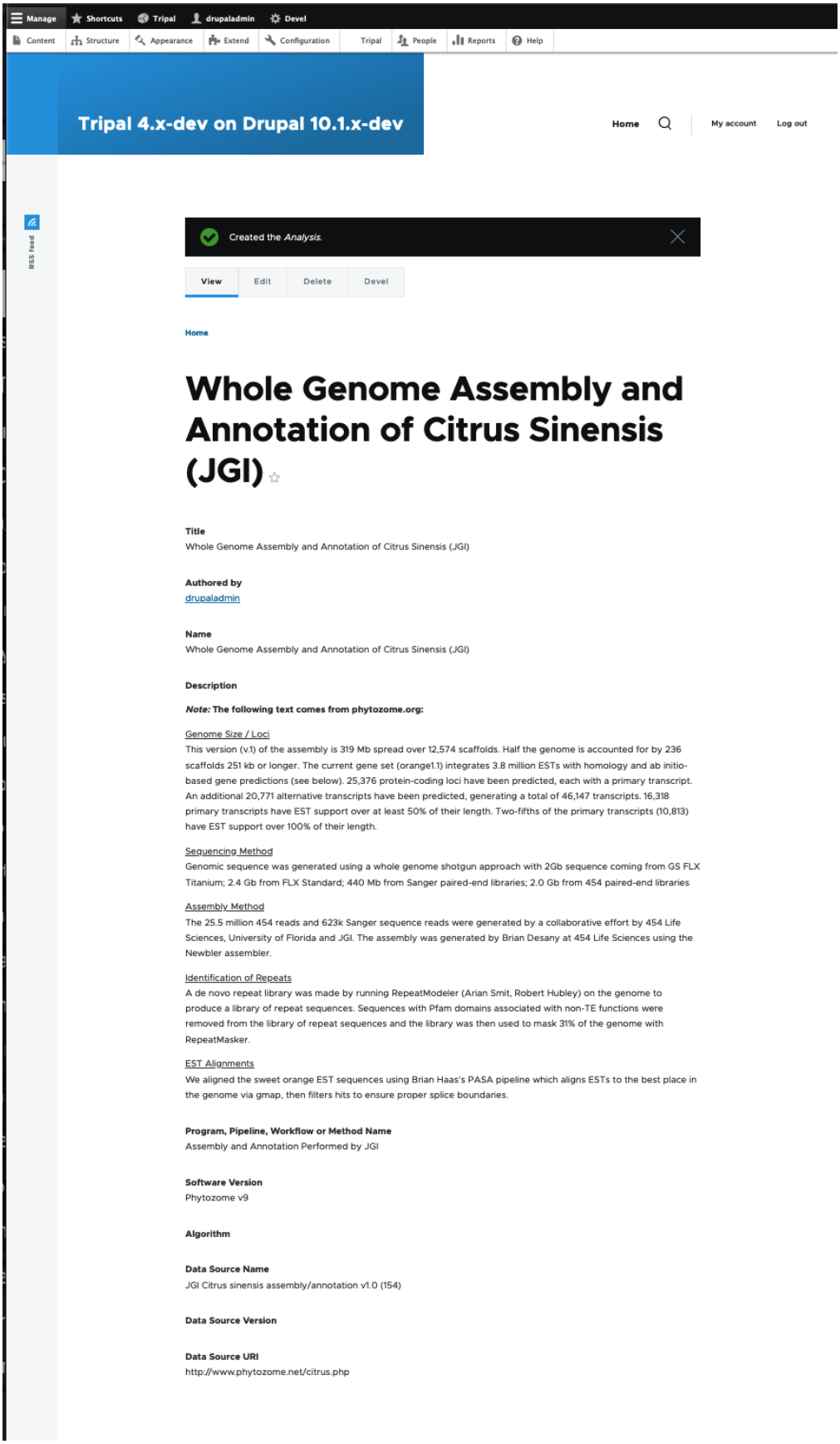

Here you can provide the necessary details to help others understand the source of your data. For this tutorial, enter the following:

| Form Element | Value |
|---|---|
| Name | Whole Genome Assembly and Annotation of Citrus Sinensis (JGI) |
| Description (Set to Full HTM |  Note: The following text comes from phytozome.org:  <u>Genome Size / Loci</u><br /> This version (v.1) of the assembly is 319 Mb spread over 12,574 scaffolds. Half the genome is accounted for by 236 scaffolds 251 kb or longer. The current gene set (orange1.1) integrates 3.8 million ESTs with homology and ab initio-based gene predictions (see below). 25,376 protein-coding loci have been predicted, each with a primary transcript. An additional 20,771 alternative transcripts have been predicted, generating a total of 46,147 transcripts. 16,318 primary transcripts have EST support over at least 50% of their length. Two-fifths of the primary transcripts (10,813) have EST support over 100% of their length.  <u>Sequencing Method</u><br /> Genomic sequence was generated using a whole genome shotgun approach with 2Gb sequence coming from GS FLX Titanium; 2.4 Gb from FLX Standard; 440 Mb from Sanger paired-end libraries; 2.0 Gb from 454 paired-end libraries  <u>Assembly Method</u><br /> The 25.5 million 454 reads and 623k Sanger sequence reads were generated by a collaborative effort by 454 Life Sciences, University of Florida and JGI. The assembly was generated by Brian Desany at 454 Life Sciences using the Newbler assembler.  <u>Identification of Repeats</u><br /> A de novo repeat library was made by running RepeatModeler (Arian Smit, Robert Hubley) on the genome to produce a library of repeat sequences. Sequences with Pfam domains associated with non-TE functions were removed from the library of repeat sequences and the library was then used to mask 31% of the genome with RepeatMasker.  <u>EST Alignments</u><br /> We aligned the sweet orange EST sequences using Brian Haas's PASA pipeline which aligns ESTs to the best place in the genome via gmap, then filters hits to ensure proper splice boundaries. |
| Program Pipeline Name or Method Name | Assembly and Annotation Performed by JGI |
| Software Version | Phytozome v9 |
| Data Source Name | JGI Citrus sinensis assembly/annotation v1.0 (154) |
| Data Source URI | http://www.phytozome.net/citrus.php |

**Note:** Above, the description is provided as HTML code. However if you enabled the **ckeditor** module (as instructed in the Tripal Prerequisites section), you should click the link **Switch to plain-text editor** found below the Description field before cut-and-pasting the code above. Normally, you would enter the text free-hand but for this tutorial it is fastest to cut-and-paste the HTML.

**Note:** Features **Publication**, **Project** and **Database Reference Annotations** have not yet been implemented for Tripal v4, documentation will be added once this feature is available.

After saving, you should have the following analysis page:


Manage
Shortcuts
Tripal
drupaladmin
Devel
Content
Structure
Appearance
Extend
Configuration
Tripal
People
Reports
Help
Tripal 4.x-dev on Drupal 10.1.x-dev
Home
My account
Log out
RSS feed
Created the Analysis.
View
Edit
Delete
Devel
Home
Whole Genome Assembly and Annotation of Citrus Sinensis (JGI)
Title
Whole Genome Assembly and Annotation of Citrus Sinensis (JGI)
Authored by
drupaladmin
Name
Whole Genome Assembly and Annotation of Citrus Sinensis (JGI)
Description
Note: The following text comes from phytozome.org:
Genome Size / Loci
This version (v.1) of the assembly is 319 Mb spread over 12,574 scaffolds. Half the genome is accounted for by 236 scaffolds 251 kb or longer. The current gene set (orange1.1) integrates 3.8 million ESTs with homology and ab initio-based gene predictions (see below). 25,376 protein-coding loci have been predicted, each with a primary transcript. An additional 20,771 alternative transcripts have been predicted, generating a total of 46,147 transcripts. 16,318 primary transcripts have EST support over at least 50% of their length. Two-fifths of the primary transcripts (10,813) have EST support over 100% of their length.
Sequencing Method
Genomic sequence was generated using a whole genome shotgun approach with 2Gb sequence coming from GS FLX Titanium; 2.4 Gb from FLX Standard; 440 Mb from Sanger paired-end libraries; 2.0 Gb from 454 paired-end libraries
Assembly Method
The 25.5 million 454 reads and 623k Sanger sequence reads were generated by a collaborative effort by 454 Life Sciences, University of Florida and JGI. The assembly was generated by Brian Desany at 454 Life Sciences using the Newbler assembler.
Identification of Repeats
A de novo repeat library was made by running RepeatModeler (Arian Smit, Robert Hubley) on the genome to produce a library of repeat sequences. Sequences with Pfam domains associated with non-TE functions were removed from the library of repeat sequences and the library was then used to mask 31% of the genome with RepeatMasker.
EST Alignments
We aligned the sweet orange EST sequences using Brian Haas's PASA pipeline which aligns ESTs to the best place in the genome via gmap, then filters hits to ensure proper splice boundaries.
Program, Pipeline, Workflow or Method Name
Assembly and Annotation Performed by JGI
Software Version
Phytozome v9
Algorithm
Data Source Name
JGI Citrus sinensis assembly/annotation v1.0 (154)
Data Source Version
Data Source URI
http://www.phytozome.net/citrus.php

### 2.3.4 Setup Cross References to external sites

**Note:** This feature has not yet been implemented for Tripal v4, documentation will be added once this feature is available

For our gene pages and mRNA pages we want to link back to JGI where we obtained the genes. Therefore, we want to add a database reference for JGI. To add a new external databases, navigate to **Tripal** → **Data Loaders** → **Chado Databases** and click the link titled **Add a Database**. The resulting page provides fields for adding a new database:

Enter the following values for the fields:

| Field Name | Value |
|---|---|
| Database Name | Phytozome |
| Description | Phytozome is a joint project of the Department of Energy's Joint Genome Institute and the Center for Integrative Genomics to facilitate comparative genomic studies amongst green plants |
| URL | http://www.phytozome.net/ |
| URL prefix | https://phytozome.jgi.doe.gov/phytomine/portal.do?externalid=PAC:{accession} |

The URL prefix is important as it will be used to create the links on our gene pages. When an object (e.g. gene) is present in another database, typically those database have a unique identifier (or accession) for the resource. If we want to link records in our database to records in the remote database we need to provide a URL prefix that Tripal will use to create the URL. Typically a remote database has a standard URL schema by which someone can specify a unique resource. Often the resource accession is the last word in the URL to allow others to easily build the URL for any resource. Tripal can take advantage of these type URL schemas via the URL Prefix field.

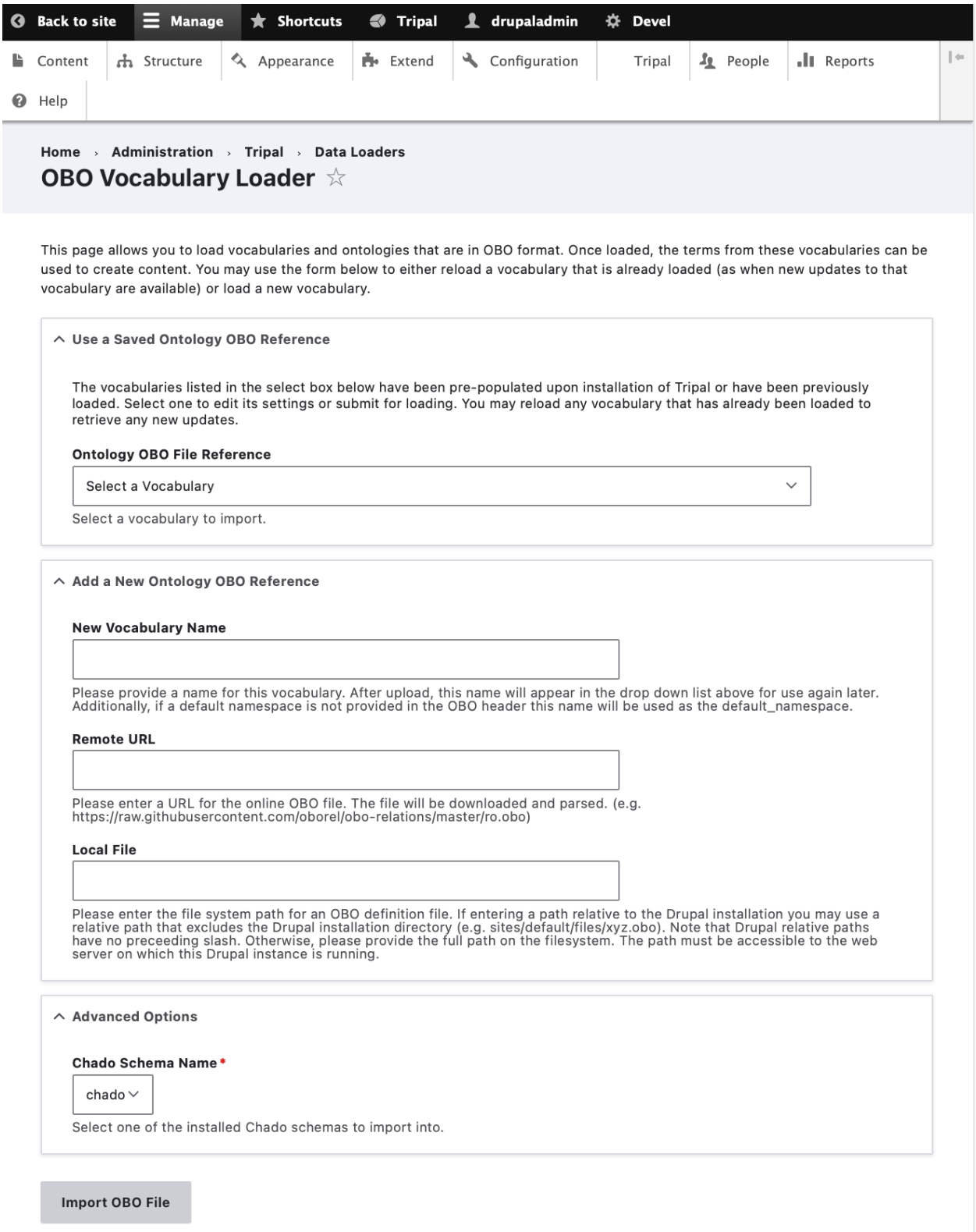
The URL prefix should be the URL used to identify a resource. Two tokens, {db} and {accession}, can be used in place of where the database name and accession might be needed to create the URL. If no {db} or {accession} are provided in the URL prefix then Tripal will append the database name and the accession to the URL prefix to form the final URL. In this example, the Phytozome URL only requires the accession. The position where that accession will be placed is indicated with the {accession} token. The {db} token is not needed.

Click **Add**.

We now have added a new database!

### 2.3.5 Import the Gene Ontology

Before we proceed with setup of our example genomics site we will want to load the Gene Ontology. This is because we will be loading a whole genome, genes and transcripts with annotations. These annotations include Gene Ontology terms. To load the Gene Ontolgoy, navigate to **Tripal** → **Data Loaders** → **OBO Vocabulary Loader**. You will see the following page:




The Ontology loader allows you to select a pre-defined vocabulary for loading or allow you to provide your own. If you provide your own, you give the remote URL of the OBO file or provide the full path on the local web server where the OBO file is located. In the case of a remote URL, Tripal first downloads and then parses the OBO file for loading. If you do provide your own OBO file it will appear in the saved drop down list for loading of future updates to the

ontology.

To import for example, the Sequence Ontology, select it from the drop-down and click the Import Vocabulary button. You will notice a job is added to the jobs system. Now manually launch the jobs

```
drush trp-run-jobs --username=administrator --root=/var/www/html
```

If Tripal is running from a docker container named $cntr_name,

```
docker exec -it $cntr_name drush trp-run-jobs --username=drupaladmin --root=/var/www/
↪drupal/web
```

**Note:** Loading an Ontology may take several hours.

### 2.3.6 Import a Genome Assembly + Annotation

Now that we have our organism and whole genome analysis ready, we can begin loading genomic data. For this tutorial only a single gene from sweet orange will be loaded into the databsae. This is to ensure we can move through the tutorial rather quickly. The following datasets will be used for this tutorial:

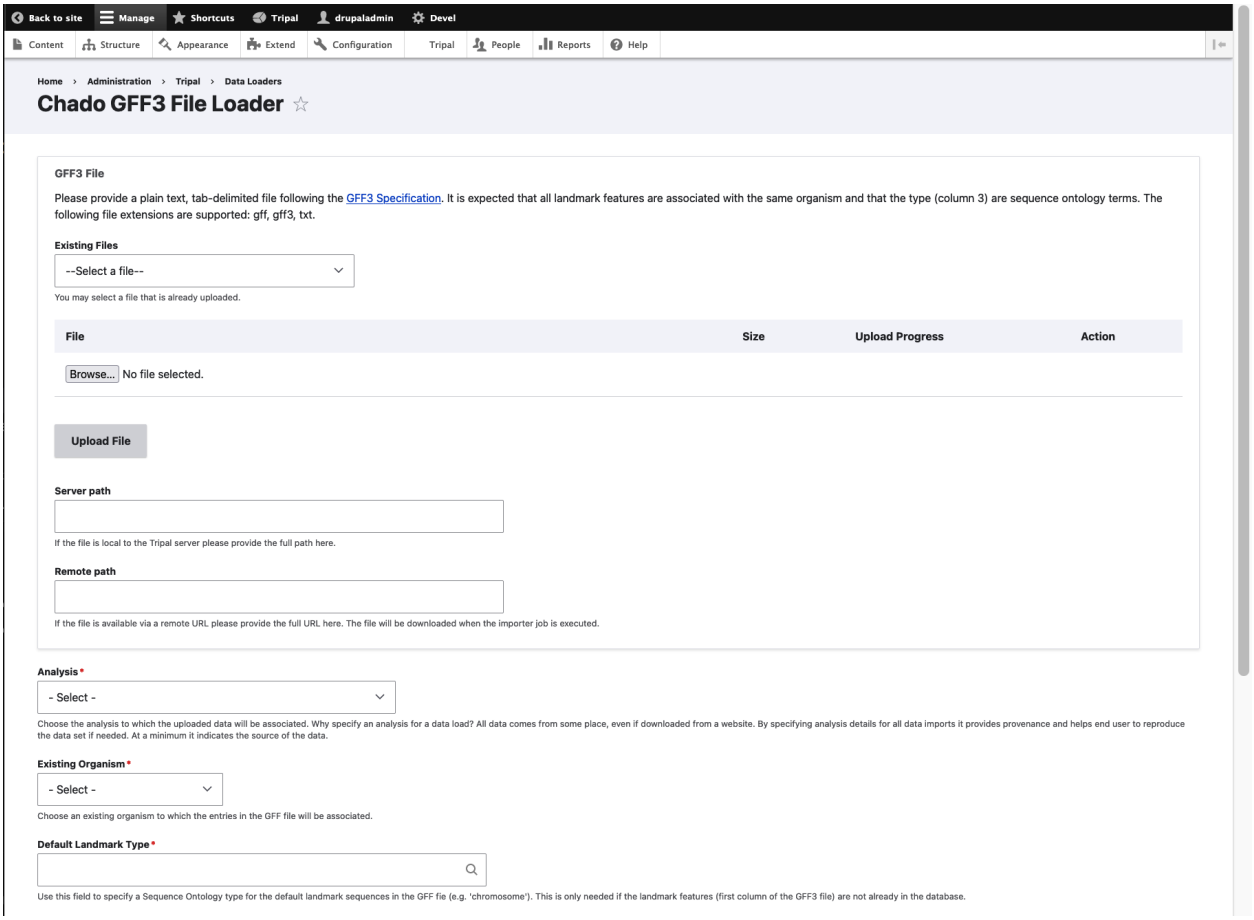
- Citrus sinensis-orange1.1g015632m.g.gff3
- Citrus sinensis-scaffold00001.fasta
- Citrus sinensis-orange1.1g015632m.g.fasta

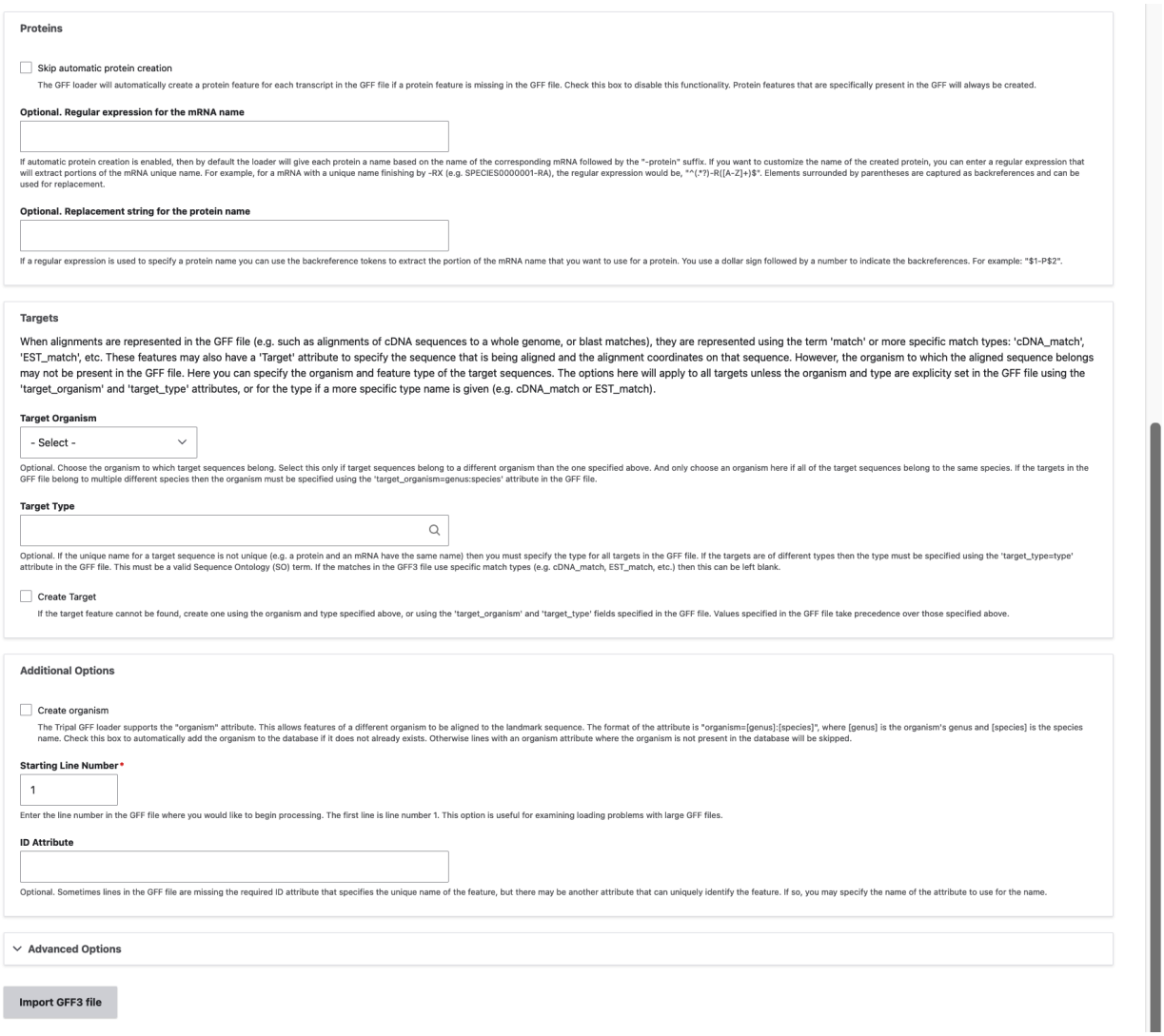
One of the new features available in many of the Tripal v4 data loaders is an HTML5 file upload element which allows administrators and users to upload large files reliably. This removes the requirement in previous versions of this tutorial to download these files directly on the server and provide a path to the file. Instead, if you have the file on your current local machine you can now simply upload it for loading.

Tripal v4 Data Loaders has the ability to provide a remote path of a file to be loaded alleviating the need to transfer large files multiple times and eases the loading process.

### Loading a GFF3 File

The gene features (e.g. gene, mRNA, 5_prime_UTRs, CDS 3_prime_UTRS) are stored in the GFF3 file downloaded in the previous step. We will load this GFF3 file and consequently load our gene features into the database. Navigate to **Tripal** → **Data Loaders** → **Chado GFF3 File Loader**.






Enter the following:

| Field Name | Value |
|---|---|
| File | Upload the file name *Citrus_sinensis-orange1.1g015632m.g.gff3* |
| Analysis | Whole Genome Assembly and Annotation of Citrus sinensis |
| Existing Organism | Citrus sinensis |
| Landmark Type | supercontig |
| All other options | leave as default |

**Note:** The Landmark Type is provided for this demo GFF3 file because the chromosome is not defined in the file, only the genomic features on the chromosomes. The landmark type is not needed if the GFF3 file has the chromosomes (scaffolds or contigs) defined in the GFF3 file.

Finally, click the Import GFF3 file button. You'll notice a job was submitted to the jobs subsystem. Now, to complete the process we need the job to run. We'll do this manually:

```
drush trp-run-jobs --username=administrator --root=/var/www/drupal/web
```

If Tripal is running from a docker container named $cntr_name,

```
docker exec -it $cntr_name drush trp-run-jobs --username=drupaladmin --root=/var/www/
↪drupal/web
```

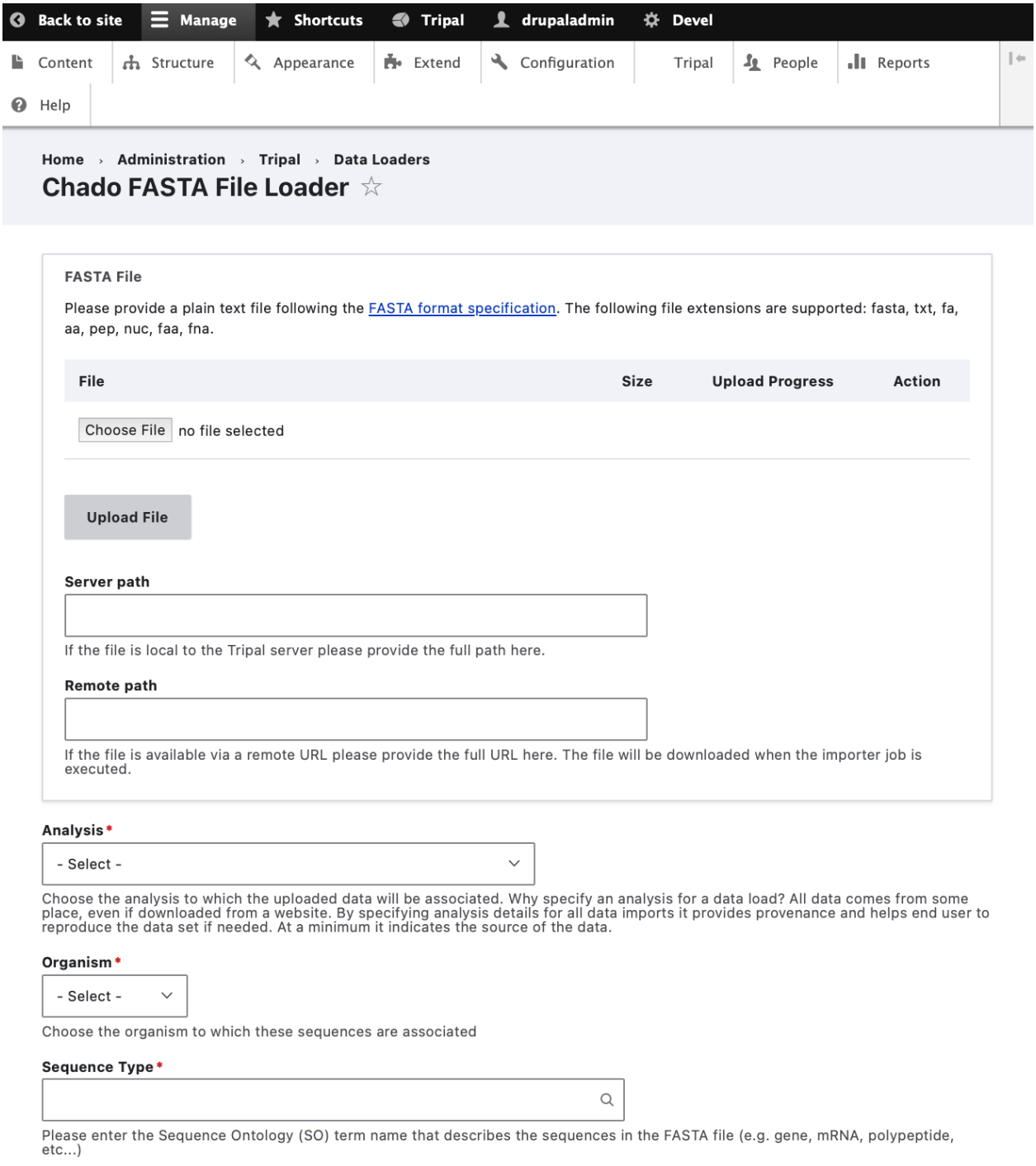
You should see output similar to the following:

```
2024-02-14 20:47:23
Tripal Job Launcher
Running as user 'drupaladmin'
-------------------
2024-02-14 20:47:23: Job ID 3.
2024-02-14 20:47:23: Calling: tripal_run_importer(6)
Running 'Chado GFF3 File Loader' importer
NOTE: Loading of this file is performed using a database transaction. If it fails or is␣
↪terminated prematurely then all insertions and updates are rolled back and will not be␣
↪found in the database
Opening //var/www/drupal/web/sites/default/files/tripal/users/1/Citrus_sinensis-orange1.
↪1g015632m.g.gff3
Opening temporary cache file: /tmp/TripalGFF3Import_1Zt2hI
Step  1 of 27: Caching GFF3 file...
Percent complete: 1.20 %. Memory: 33,996,128 bytes.
Percent complete: 3.06 %. Memory: 34,003,168 bytes.
:::
:::
Step 27 of 27: Adding sequences data (Skipped: none available)...
Done.
```

**Note:** For very large GFF3 files the loader can take quite a while to complete.

### Loading FASTA files

Using the Tripal GFF3 loader we were able to populate the database with the genomic features for our organism. However, those features now need nucleotide sequence data. To do this, we will load the nucleotide sequences for the mRNA features and the scaffold sequence. Navigate to the **Tripal** → **Data Loaders** → **Chado FASTA File Loader**.


Back to site
Manage
Shortcuts
Tripal
drupaladmin
Devel
Content
Structure
Appearance
Extend
Configuration
Tripal
People
Reports
Help
Home › Administration › Tripal › Data Loaders
Chado FASTA File Loader
FASTA File
Please provide a plain text file following the FASTA format specification. The following file extensions are supported: fasta, txt, fa, aa, pep, nuc, faa, fna.
File
Size
Upload Progress
Action
Choose File
no file selected
Upload File
Server path
If the file is local to the Tripal server please provide the full path here.
Remote path
If the file is available via a remote URL please provide the full URL here. The file will be downloaded when the importer job is executed.
Analysis *
- Select -
Choose the analysis to which the uploaded data will be associated. Why specify an analysis for a data load? All data comes from some place, even if downloaded from a website. By specifying analysis details for all data imports it provides provenance and helps end user to reproduce the data set if needed. At a minimum it indicates the source of the data.
Organism *
- Select -
Choose the organism to which these sequences are associated
Sequence Type *
Please enter the Sequence Ontology (SO) term name that describes the sequences in the FASTA file (e.g. gene, mRNA, polypeptide, etc...)

**Method** *

- ○ Insert only
- ○ Update only
- ◉ Insert and update

Select how features in the FASTA file are handled. Select "Insert only" to insert the new features. If a feature already exists with the same name or unique name and type then it is skipped. Select "Update only" to only update featues that already exist in the database. Select "Insert and Update" to insert features that do not exist and update those that do.

**Name Match Type** *

- ○ Name
- ◉ Unique name

Used for "updates only" or "insert and update" methods. Not required if method type is "insert". Feature data is stored in Chado with both a human-readable name and a unique name. If the features in your FASTA file are uniquely identified using a human-readable name then select the "Name" button. If your features are uniquely identified using the unique name then select the "Unique name" button. If you loaded your features first using the GFF loader then the unique name of each feature was indicated by the "ID=" attribute and the name by the "Name=" attribute. By default, the FASTA loader will use the first word (character string before the first space) as the name for your feature. If this does not uniquely identify your feature consider specifying a regular expression in the advanced section below. Additionally, you may import both a name and a unique name for each sequence using the advanced options.

**Additional Options**

**Regular expression for the name**

Enter the regular expression that will extract the feature name from the FASTA definition line. For example, for a defintion line with a name and unique name separated by a bar '|' (>seqname|uniquename), the regular expression for the name would be, "^(.*?)\|.*$". All FASTA definition lines begin with the ">" symbol. You do not need to incldue this symbol in your regular expression.

**Regular expression for the unique name**

Enter the regular expression that will extract the feature name from the FASTA definition line. For example, for a defintion line with a name and unique name separated by a bar '|' (>seqname|uniquename), the regular expression for the unique name would be "^.*?\|(.*)$". All FASTA definition lines begin with the ">" symbol. You do not need to incldue this symbol in your regular expression.

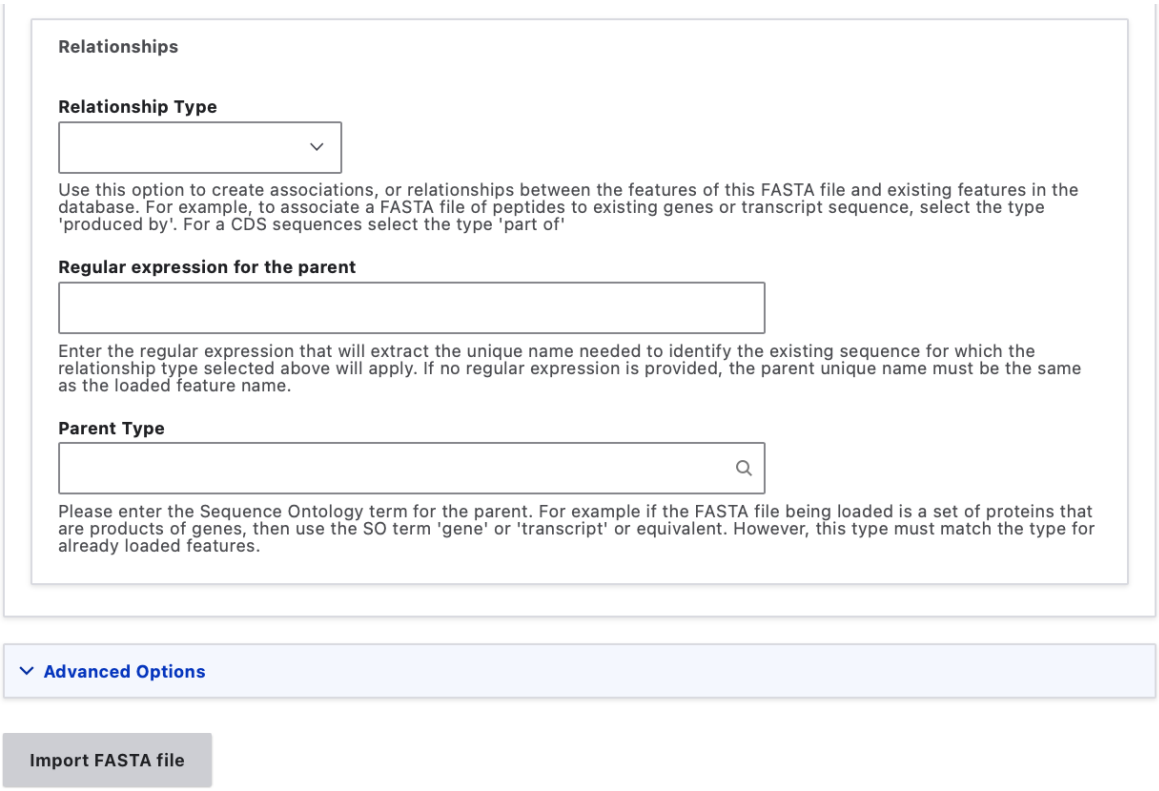
**External Database Reference**

**External Database**

Plese choose an external database for which these sequences have a cross reference.

**Regular expression for the accession**

Enter the regular expression that will extract the accession for the external database for each feature from the FASTA definition line.




Before loading the FASTA file we must first know the Sequence Ontology (SO) term that describes the sequences we are about to upload. We can find the appropriate SO terms from our GFF file. In the GFF file we see the SO terms that correspond to our FASTA files are 'scaffold' and 'mRNA'.

---

**Note:** It is important to ensure prior to importing, that the FASTA loader will be able to appropriately match the sequence in the FASTA file with existing sequences in the database. Before loading FASTA files, take special care to ensure the definition line of your FASTA file can uniquely identify the feature for the specific organism and sequence type.

---

For example, in our GFF file an mRNA feature appears as follows:

```
scaffold00001   phytozome6      mRNA    4058460 4062210 .       +       .       ␣
↪ID=PAC:18136217;Name=orange1.1g015632m;PACid=18136217;Parent=orange1.1g015632m.g
```

Note that for this mRNA feature the ID is **PAC:18136217** and the name is **orange1.1g015632m**. In Chado, features always have a human readable name which does not need to be unique, and also a unique name which must be unique for the organism and SO type. In the GFF file, the ID becomes the unique name and the Name becomes the human readable name.

In our FASTA file the definition line for this mRNA is:

```
>orange1.1g015632m PAC:18136217 (mRNA) Citrus sinensis
```

By default Tripal will match the sequence in a FASTA file with the feature that matches the first word in the definition line. In this case the first word is **orange1.1g015632m**. As defined in the GFF file, the name and unique name are different for this mRNA. However, we can see that the first word in the definition line of the FASTA file is the name and the second is the unique name. Therefore, when we load the FASTA file we should specify that we are matching

by the name because it appears first in the definition line.

If however, we cannot guarantee the that feature name is unique then we can use a regular expressions in the **Advanced Options** to tell Tripal where to find the name or unique name in the definition line of your FASTA file.

---

**Note:** When loading FASTA files for features that have already been loaded via a GFF file, always choose "Update only" as the import method. Otherwise, Tripal may add the features in the FASTA file as new features if it cannot properly match them to existing features.

---

Now, enter the following values in the fields on the web form:

| Field Name | Value |
|---|---|
| FASTA file | Upload the file named *Citrus_sinensis-scaffold00001.fasta* |
| Analysis | Whole Genome Assembly and Annotation of Citrus sinensis |
| Organism | Citrus sinensis (Sweet orange) |
| Sequence type | supercontig (scaffold is an alias for supercontig in the sequence ontology) |
| Method | Update only (we do not want to insert these are they should already be there) |
| Name Match Type | Name |

Click the Import Fasta File, and a job will be added to the jobs system. Run the job:

```
drush trp-run-jobs --username=administrator --root=/var/www/html
```

Notice that the loader reports the it "Found 1 sequences(s).". Next fill out the same form for the mRNA (transcripts) FASTA file:

| Field Name | Value |
|---|---|
| FASTA file | Upload the file named *Citrus_sinensis-orange1.1g015632m.g.fasta* |
| Analysis | Whole Genome Assembly and Annotation of Citrus sinensis |
| Organism | Citrus sinensis (Sweet orange) |
| Sequence type | mRNA |
| Method | Update only |
| Name Match | Name |

The FASTA loader has some advanced options. The advanced options allow you to create relationships between features and associate them with external databases. For example, the definition line for the mRNA in our FASTA file is:

```
>orange1.1g015632m PAC:18136217 (mRNA) Citrus sinensis
```

Here we have more information than just the feature name. We have a unique Phytozome accession number (e.g. PAC:18136217) for the mRNA. Using the **External Database Reference** section under **Additional Options** we can import this information to associate the Phytozome accession with the features. A regular expression is required to uniquely capture that ID. In the example above the unique accession is 18136217. Because Tripal is a PHP application, the syntax for regular expressions follows the PHP method. Documentation for regular expressions used in PHP can be found here. Enter the following value to make the associate between the mRNA and it's corresponding accession at Phytozome:

| Field Name | Value |
|---|---|
| External Database | Phytozome |
| Regular expression for the accession | ^.*PAC:(d+).*$ |

Remember, we have the name **Phytozome** in our **External Database** drop down because we manually added it as a database cross reference earlier in the turorial. After adding the values above, click the **Import FASTA file** button, and manually run the submitted job:

```
drush trp-run-jobs --username=administrator --root=/var/www/html
```

Now the scaffold sequence and mRNA sequences are loaded!

---

**Note:** It is not required to load the mRNA sequences as those can be derived from their alignments with the scaffold sequence. However, in Chado the **feature** table has a **residues** column. Therefore, it is best practice to load the sequence when possible.

---

---

**Note:** Features written below have not yet been implemented for Tripal v4, documentation will be updated once they are available

---

### Creating Gene Pages

Now that we've loaded our feature data, we must publish them. This is different than when we manually created our Organism and Analysis pages. Using the GFF and FASTA loaders we imported our data into Chado, but currently there are no published pages for this data that we loaded. To publish these genomic features, navigating to **Structure** → **Tripal Content Type** and click the link titled **Publish Chado Content**.

Here we can specify the types of content to publish. For our site we want to offer both gene and mRNA pages (these types were present in our GFF file). First, to create pages for genes select 'Gene' from the dropdown. A new Filter section is present and when opened appears as follows.

The **Filters** section allows you to provide filters to limit what you want to publish. For example, if you only want to publish genes for a single organism you can select that organism in the Organism drop down list. We only have one organism in our site, but for the sake of experience, add a filter to publish only genes for Citrus sinesis by selecting it from the Organism drop down. Scroll to the bottom a click the Publish button. A new job is added to the job queue. Manually run the job:

```
drush trp-run-jobs --username=administrator --root=/var/www/html
```

You should see output similar to the following:

```
Tripal Job Launcher
Running as user 'administrator'
-------------------
Calling: tripal_chado_publish_records(Array, 12)

NOTE: publishing records is performed using a database transaction.
If the load fails or is terminated prematurely then the entire set of
is rolled back with no changes to the database
```

(continues on next page)

(continued from previous page)

```
Succesfully published 1 Gene record(s).
```

Here we see that 1 gene was successfully published. This is because the GFF file we used previously to import the genes only had one gene present.

Now, repeat the steps above to publish the mRNA content type. You should see that 9 mRNA records were published:

```
Tripal Job Launcher
Running as user 'administrator'
-------------------
Calling: tripal_chado_publish_records(Array, 13)

NOTE: publishing records is performed using a database transaction.
If the load fails or is terminated prematurely then the entire set of
is rolled back with no changes to the database

Succesfully published 9 mRNA record(s).
```

---

**Note:** It is not necessary to publish all types of features in the GFF file. For example, we do not want to publish features of type **scaffold**. The feature is large and would have many relationships to other features, as well as a very long nucleotide sequence. These can greatly slow down page loading, and in general would be overwhelming to the user to view on one page. As another example, each **mRNA** is composed of several **CDS** features. These **CDS** features do not need their own page and therefore do not need to be published.

---

Now, we can view our gene and mRNA pages. Click the Find Tripal Content link. Find and click the new page titled **orange1.1g015632m.g**. Here we can see the gene feature we added and its corresponding mRNA's.

Next find an mRNA page to view. Remember when we loaded our FASTA file for mRNA that we associated the record with Phytozome. On these mRNA pages you will see a link in the left side bar titled **Database Cross Reference**. Clicking that will open a panel with a link to Phytozome. This link appears because:

- We added a Database Cross Reference for Phytozome in a previous step
- We associated the Phytozome accession with the features using a regular expression when importing the FASTA file.

All data that appears on the page is derived from the GFF file and the FASTA files we loaded.

### Customizing Transcripts on Gene Pages

By default the gene pages provided by Tripal will have a link in the sidebar table of contents named **Transcripts** and when clicked a table appears that lists all of the transcripts (or mRNA) that belong to the gene. The user can click to view more information about each published transcript.

Sometimes however, more than just a listing of transcripts is desired on a gene page. You can customize the information that is presented about each transcript by navigating to the gene content type at **Structure** → **Tripal Content Types** and clicking **mange fields** in the **Gene** row. This page allows you to customize the way fields are displayed on the gene page. Scroll down the page to the **Transcript** row and click the **edit** button.

Open the field set titled **Transcript (mRNA) Field Selection** to view a table that lists all of the available fields for a transcript.

On this page you can check the boxes next to the field that you want to show for a transcript on the gene page. For this example, we will select the fields **Name**, **Identifier**, **Resource Type**, **Anotations**, and **Sequences** (they may not

be in this order on your own site). You can control the order in which fields will be shown by dragging them using the crosshairs icon next to each one. Scroll to the bottom of the page and click the **Save Settings** button.

Next return to the gene page, reload it, and click on the **Transcripts** link. Now you are provided a select box with the transcript names. When a transcript is selected, the pane below will populate with the fields that you selected when editing in the Transcript field.

You can return to the Transcript field edit page under the Gene content type at any time to add, remove or change the order of fields that appear for the transcript.

---

**Note:** Transcripts on a gene page can only be customized if all of them are published. If not, the default table listing is shown.

---

CHAPTER

THREE

# GUIDING YOUR USERS

CHAPTER

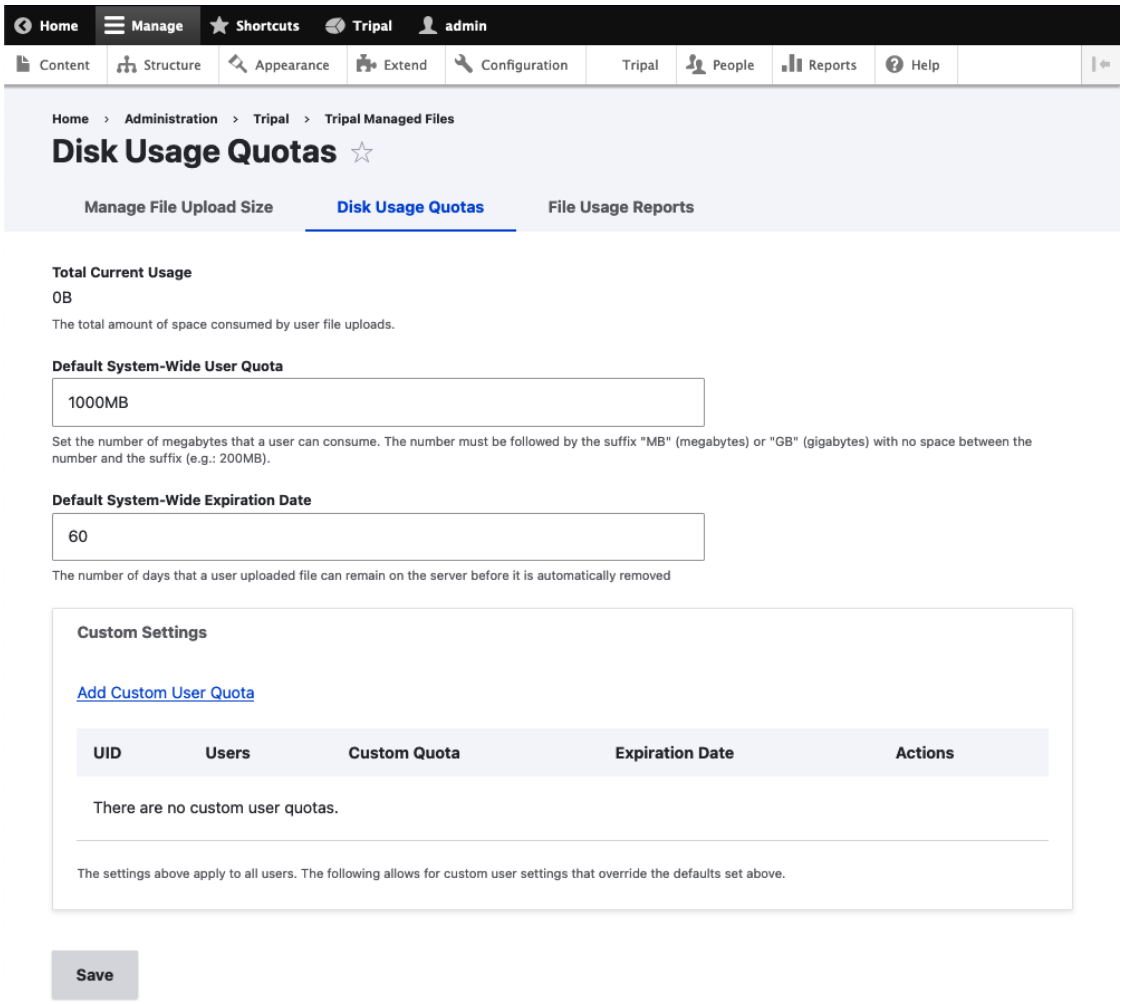
FOUR

# SITE ADMINISTRATION

## 4.1 File Management

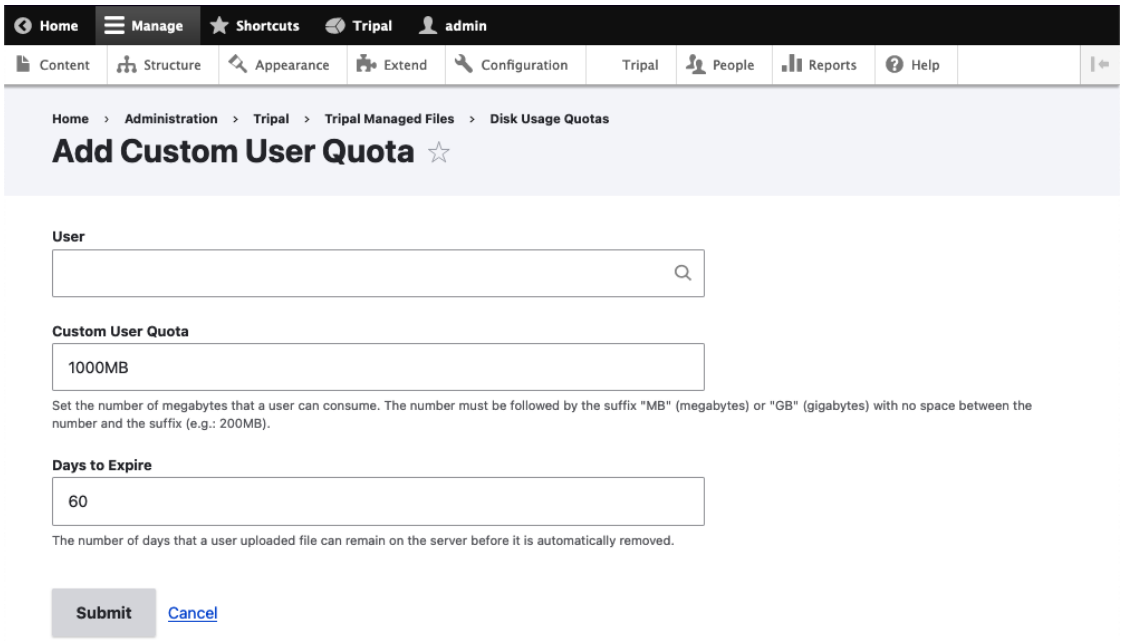
### 4.1.1 User Quotas

Data importers that use the Tripal API and Tripal supported widgets automatically associate uploaded files with users. If you are allowing end-users to upload files you may want to consider adding quotas to prevent the server storage from filling. To ensure that users do not exceed the limits of the server a quota system is available. Navigate to **Administer > Tripal > Tripal Managed Files** and click the **Disk User Quotas** tab to reveal the following page:

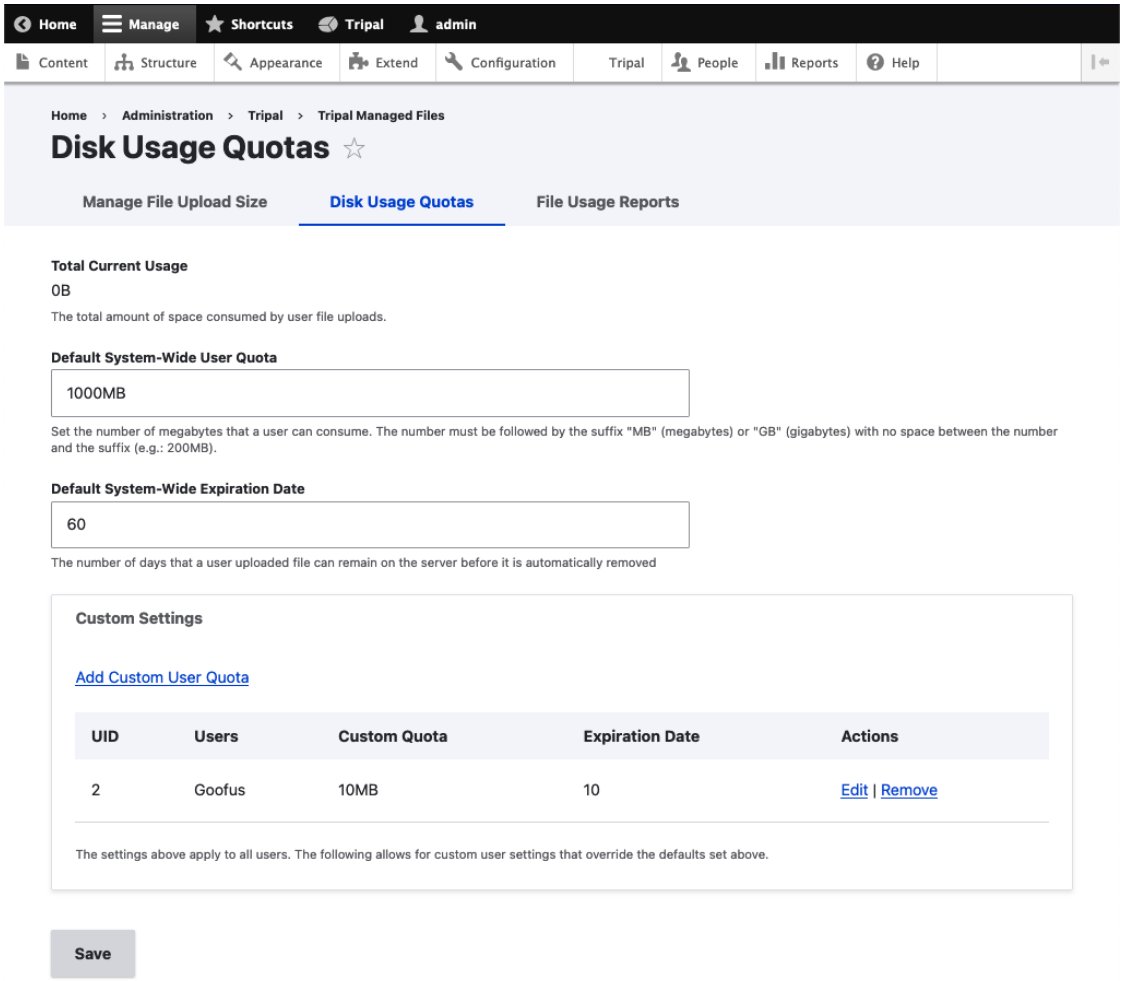



First, the total amount of space consumed by all uploaded files is shown at the top of the page. Initially this will indicate 0 B (for zero bytes); as users upload files this value will change. You may return to this page in the future to check how much space is currently used by user uploads. Here you can also specify the default system-wide quota that all users receive. By default this is set to 64 Megabytes and an expiration of 60 days. Once a file has existed on the site for 60 days the file is marked for deletion and will be removed when the Drupal cron is executed. The default of 64MB per user is most likely too small for your site. Adjust this setting and the days to expire as appropriate for your site's expected number of users and storage limitations and click the **Save** button to preserve any changes you have made.

In addition to the default settings for all users, you may want to allow specific users to have a larger (or perhaps smaller) quota. You can set user-specific quotas by clicking the **Add Custom User Quota** link near the bottom of the page. The following page appears:

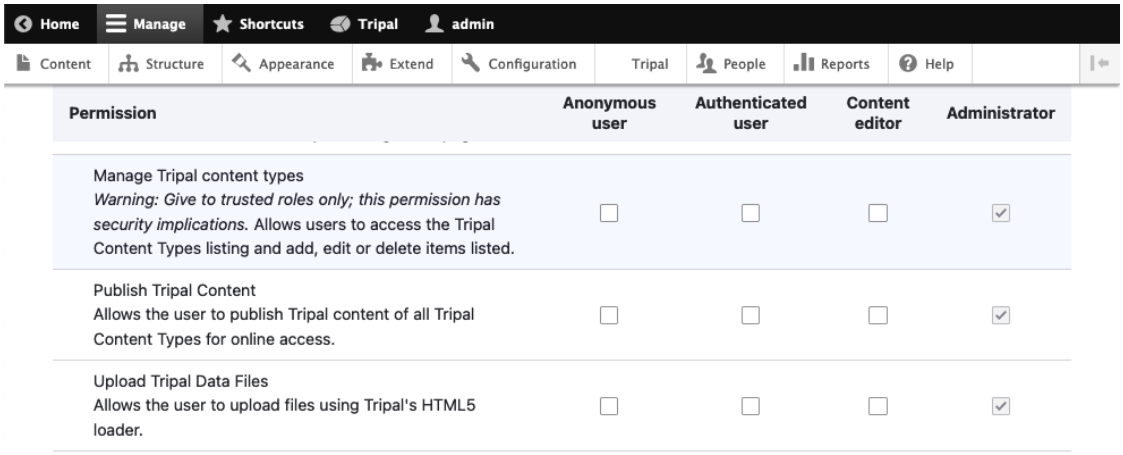



Here you must specify the Drupal user name of the user who should be granted a custom quota. This field will auto populate suggestions as you type to help you find the correct username. Enter the desired quota size and expiration days and click the **Submit** button. You will then see the user-specific quota listed in the table at the bottom of the page:




### 4.1.2 Users' Files

Users with permission to upload files are able to use the Tripal file uploader to add files to the server. The core Tripal Data Importers use the Tripal file uploader and extension modules may use it as well. You can enable this functionality for users by Navigating to **Admin > People** and click the **Permissions** Tab. next scroll to the **Tripal** section and set the **Upload Files** permissions as desired for your site. The following screenshot shows the permission on a default Drupal site.




Users who have the ability to upload files can manage files on their own Account pages.

As described in the previous section, the site administrator can set a system-wide or user-specific default expiration number of days for a file. This means files will be removed automatically from the server once their expiration data is set.

---

**Note:** Automatic removal of files can only occur if the Drupal cron is setup to run automatically.

---

## 4.2 Publishing

**Warning:** This functionality is still being actively developed. Currently you cannot create pages for records added to chado through any other method than the Create Content forms.

This work will be completed and documented before the next release of Tripal 4.

## 4.3 Tripal Jobs

Tripal comes with a robust job system for running on-demand or scheduled jobs. These jobs can be run manually or automatically using the Tripal Job Daemon.

### 4.3.1 Manual Job Execution

#### Overview

At various times, you may be given a command to run in order to launch a job. For example, when initially setting up the site to install and prepare Chado, or when using of the importers. The command will typically take this form:

```
drush trp-run-jobs --job_id=2 --username=admin --root=/var/www/tripal4/web
```

Let's break that down:

- `trp-run-jobs`

  This is the command. There are other commands available, see the section below.

- `--job_id=2`

  This specifies which job in the queue that will run.

- `--username=admin`

  This is the username of the owner of the job. This is important as it can help track down changes made to the site.

- `--root=/var/www/tripal4/web/`

  This is the root directory of the current Drupal site. This is especially important to specify when multiple sites are installed.

#### Commands and Arguments

There are two main Drush commands related to the Tripal Job system:

- `trp-run-jobs`

  This command will run jobs that are on the queue. If the *job_id* flag is provided, it will run that specific job, otherwise it will run all jobs that are on the queue in chronological order based on when they were submitted. The following arguments are available:

  - `job_id` [optional] - Specify the number (id) of the job that is to be run.
  - `username` [required] - Specify the username of the person who is running the job.

- `trp-rerun-job` — This command will rerun a specified job. The *job_id* flag is required.

The following is a list of arguments

### 4.3.2 Automating Job Execution

Currently, running jobs automatically is not supported. This functionality will come with the upgrade of the Tripal Daemon module.

**Warning:** These docs are still being developed. In the future this page will contain a very short introduction to user management in Drupal and practical examples for site administration of a Tripal site.

## 4.4 User Permissions

*Users* : Anyone who visits your website including you. There are three groups of users. *anonymous users* (who are not logged in), *authenticated users* (who are logged in) and *administrative users* (created when a site was installed, or User 1).

*Permissions* : A group of actions (example - import a GFF3 file, view/edit content and change configuration). Permissions are defined by the modules that provide the actions.

*Roles* : Permissions are grouped into roles, each of which can be defined and then permissions are granted. Exaample roles are Curators or Data Submittors.

Users and permissions allow you to give certain groups for example, researchers access to private data. Roles can help you setup groups of collaborators so you can assign the permission to the group as a whole which makes it easier if any one member leaves or joins the group.

Refer to https://www.drupal.org/docs/user_guide/en/user-concept.html for more details.

It is a good practice to make several roles on your Tripal site. For example, for managing biological data and knowledgebases like model organism database, you might want a Curator role that allows data curators to curate information on a specific organism using appropriate unique traceable identifiers, and providing necessary metadata including source and provenance. a set of genes, and their associated mRNA, CDS, UTRs, etc. For more information, see https://en.wikipedia.org/wiki/Biocuration.

### 4.4.1 Creating Roles to enable Curation

Biocuration involves the collection, curation, annotation, validation, writing related grants, and publications and integration of information related to the biological sciences into databases or resources.

Here is a walk through creating a "Curator" role in Tripal based on a need and then assign these roles permissions by the Administrator. For example, a curator of genomic data would need access to specific importers and content types associated with the genome of the organism.

### 4.4.2 Steps

From the top menu :

**Home** -> **Administration** -> **People** -> **Roles** ->

You will find default roles Anonymous user, Authenticated user, and Administrator already present.

#### Create User

From the top menu :

**Home -> Administration -> People -> +Add User ->**

- Username : curator_user
- Password : *abcd_123_!@#*
- Roles : Content editor

Click on **Create new account** leaving other items as default

From the top menu -> **People** -> *curator_user* now appears in list of Usernames.

#### Create Roles and Assign them to Users

**Home** -> **Administration** -> **People** -> **+Add Role** ->

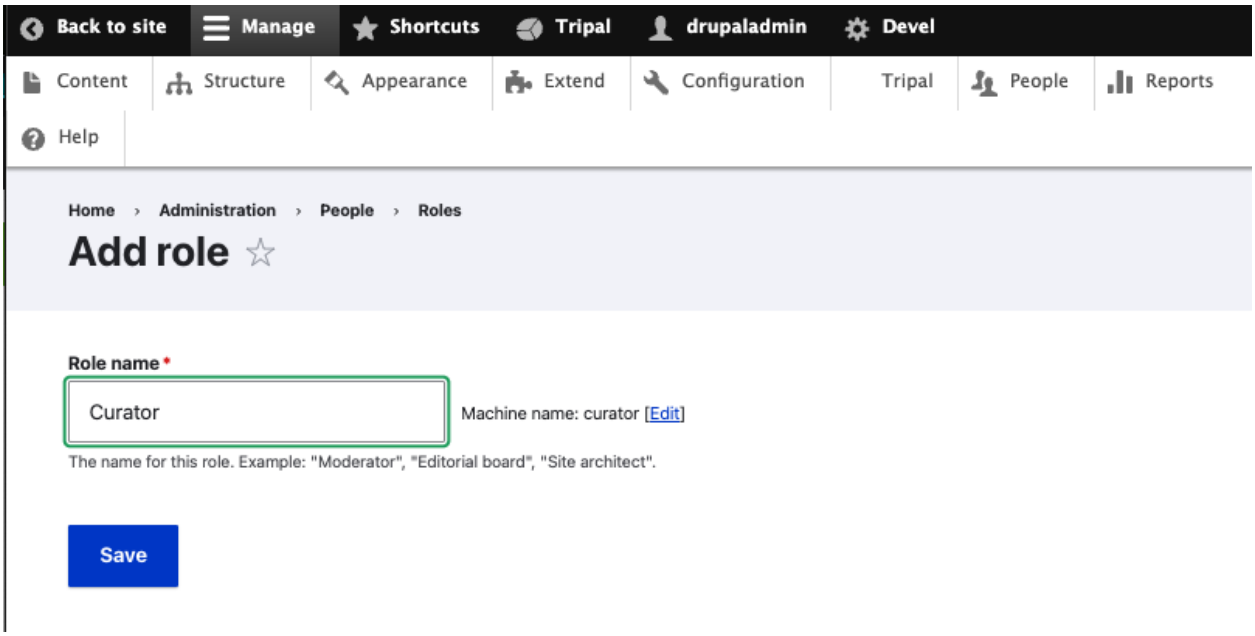
Type *Curator* for Role name in text box and click Save. A status message is displayed.

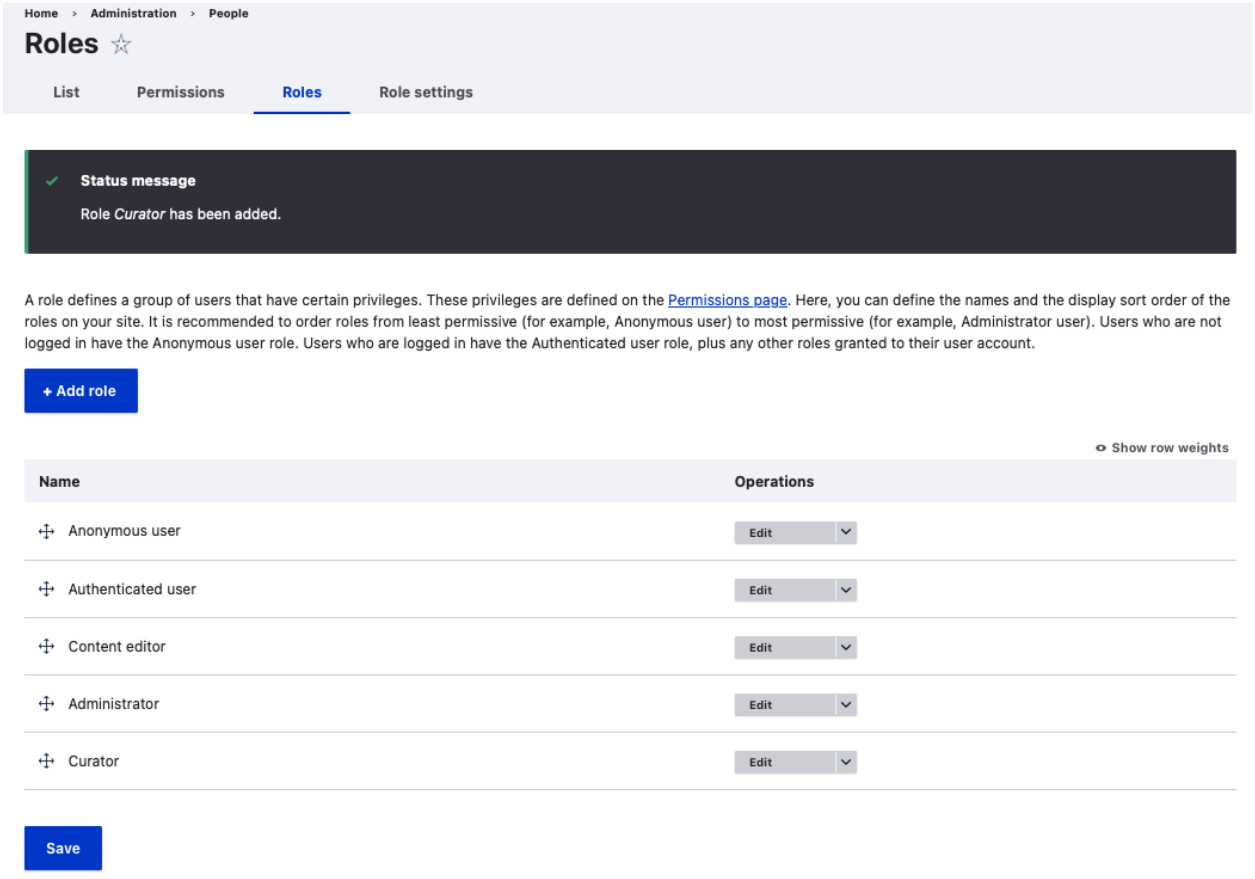
And you can find the Curator Role added to the list of Roles under Name.

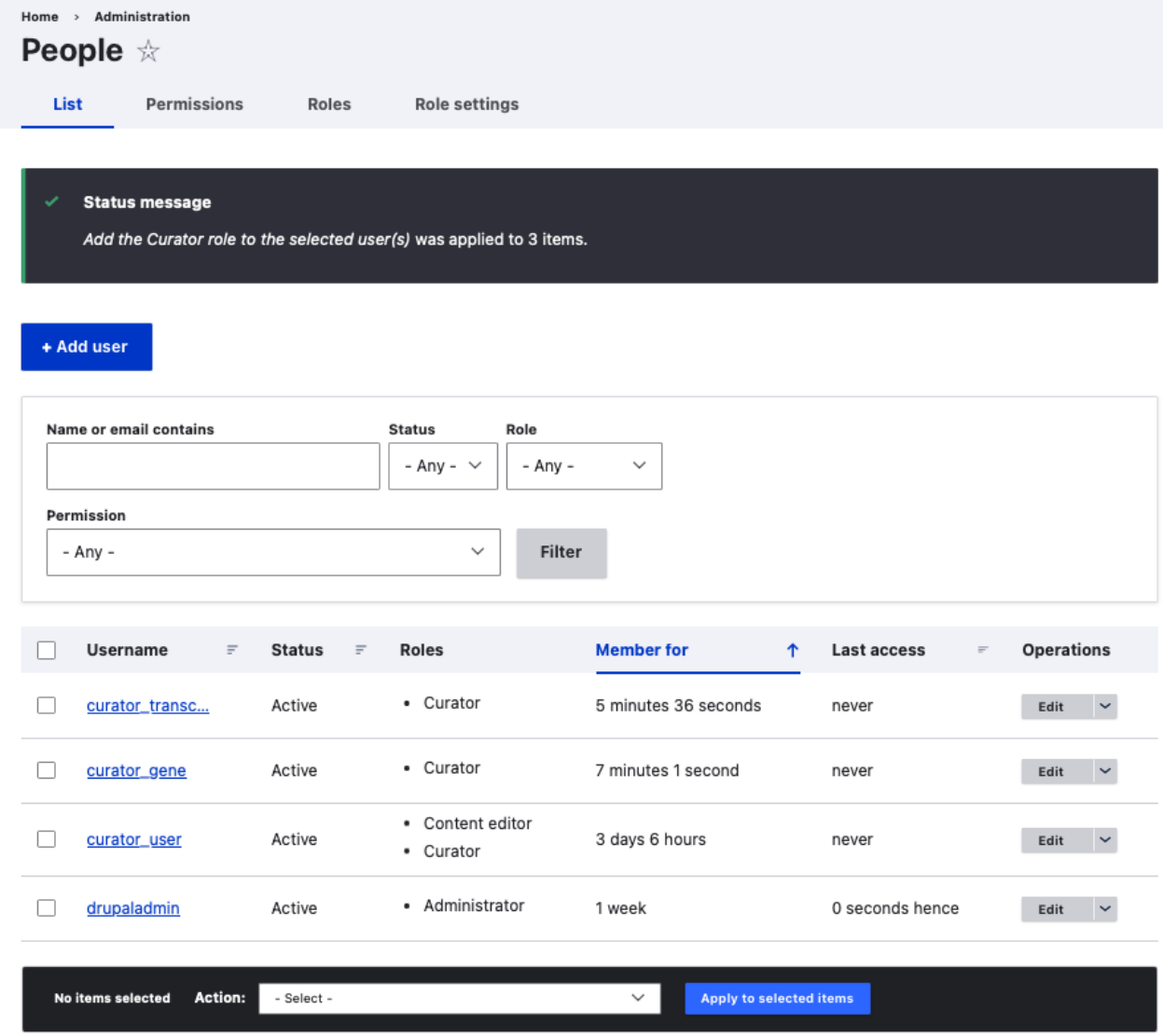
To perform same action on multiple users, for example, to add the Curator role to more than 1 user, from the top menu,

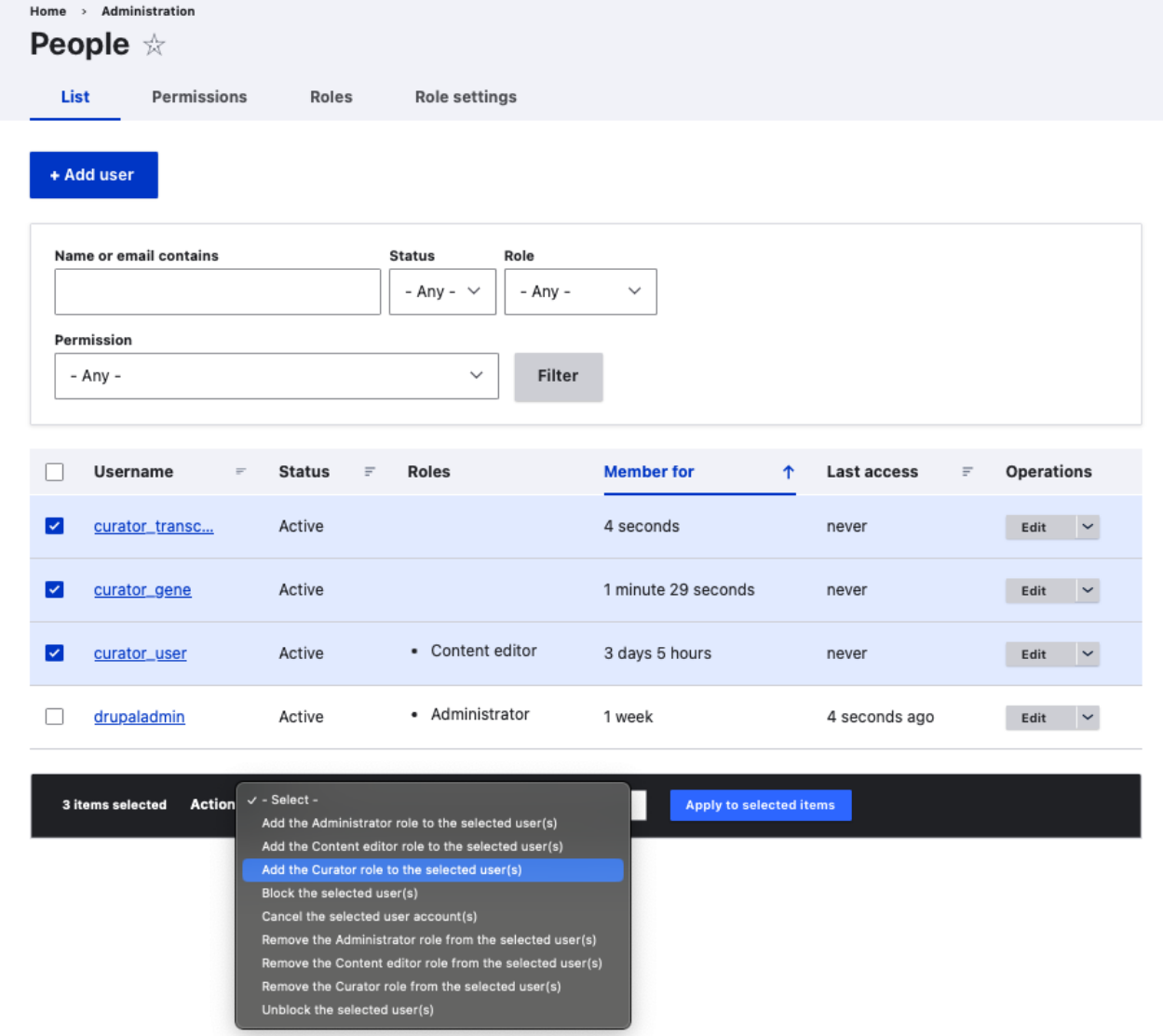
**Home** -> **Administration** -> **People** -> click inside checkbox before all usernames having prefix *curator_*.

Click on dropdown next to Action -> select **Add the Curator role to the selected users** -> **Apply to selected items**.


Back to site
Manage
Shortcuts
Tripal
drupaladmin
Devel
Content
Structure
Appearance
Extend
Configuration
Tripal
People
Reports
Help
Home › Administration › People › Roles
Add role
Role name *
Curator
Machine name: curator [Edit]
The name for this role. Example: "Moderator", "Editorial board", "Site architect".
Save

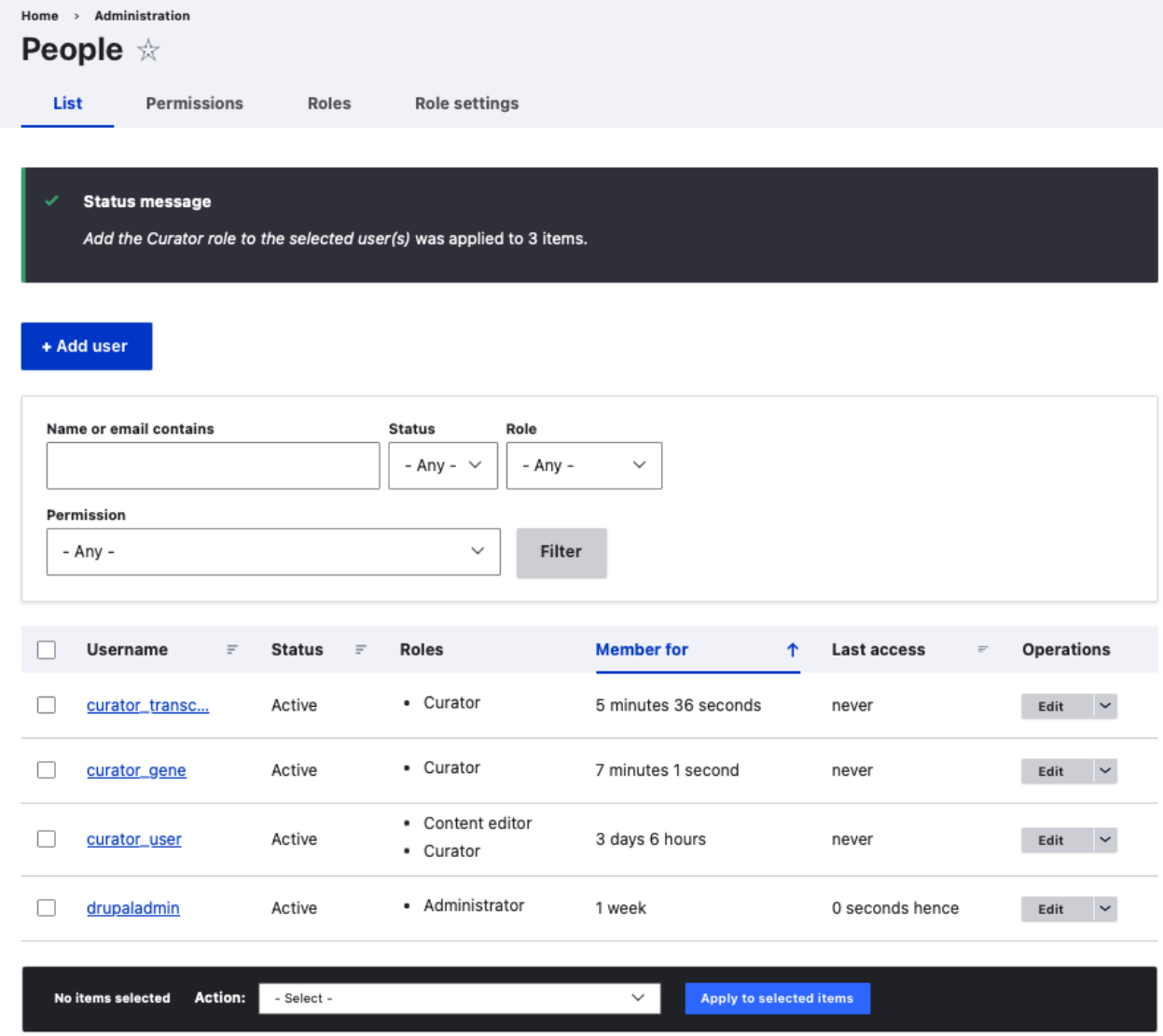

Home › Administration › People
Roles
List
Permissions
Roles
Role settings
Status message
Role Curator has been added.
A role defines a group of users that have certain privileges. These privileges are defined on the Permissions page. Here, you can define the names and the display sort order of the roles on your site. It is recommended to order roles from least permissive (for example, Anonymous user) to most permissive (for example, Administrator user). Users who are not logged in have the Anonymous user role. Users who are logged in have the Authenticated user role, plus any other roles granted to their user account.
+ Add role
Show row weights
Name
Operations
Anonymous user
Edit
Authenticated user
Edit
Content editor
Edit
Administrator
Edit
Curator
Edit
Save


Home › Administration
People
List
Permissions
Roles
Role settings
Status message
Add the Curator role to the selected user(s) was applied to 3 items.
+ Add user
Name or email contains
Status
- Any -
Role
- Any -
Permission
- Any -
Filter
Username
Status
Roles
Member for
Last access
Operations
curator_transc...
Active
Curator
5 minutes 36 seconds
never
Edit
curator_gene
Active
Curator
7 minutes 1 second
never
Edit
curator_user
Active
Content editor
Curator
3 days 6 hours
never
Edit
drupaladmin
Active
Administrator
1 week
0 seconds hence
Edit
No items selected
Action:
- Select -
Apply to selected items


Home › Administration
People
List
Permissions
Roles
Role settings
+ Add user
Name or email contains
Status
- Any -
Role
- Any -
Permission
- Any -
Filter
Username
Status
Roles
Member for
Last access
Operations
curator_transc...
Active
4 seconds
never
Edit
curator_gene
Active
1 minute 29 seconds
never
Edit
curator_user
Active
Content editor
3 days 5 hours
never
Edit
drupaladmin
Active
Administrator
1 week
4 seconds ago
Edit
3 items selected
Action
- Select -
Add the Administrator role to the selected user(s)
Add the Content editor role to the selected user(s)
Add the Curator role to the selected user(s)
Block the selected user(s)
Cancel the selected user account(s)
Remove the Administrator role from the selected user(s)
Remove the Content editor role from the selected user(s)
Remove the Curator role from the selected user(s)
Unblock the selected user(s)
Apply to selected items

All user names having prefix *curator_* now have the role of Curator.

**Home -> Administration -> People ->**



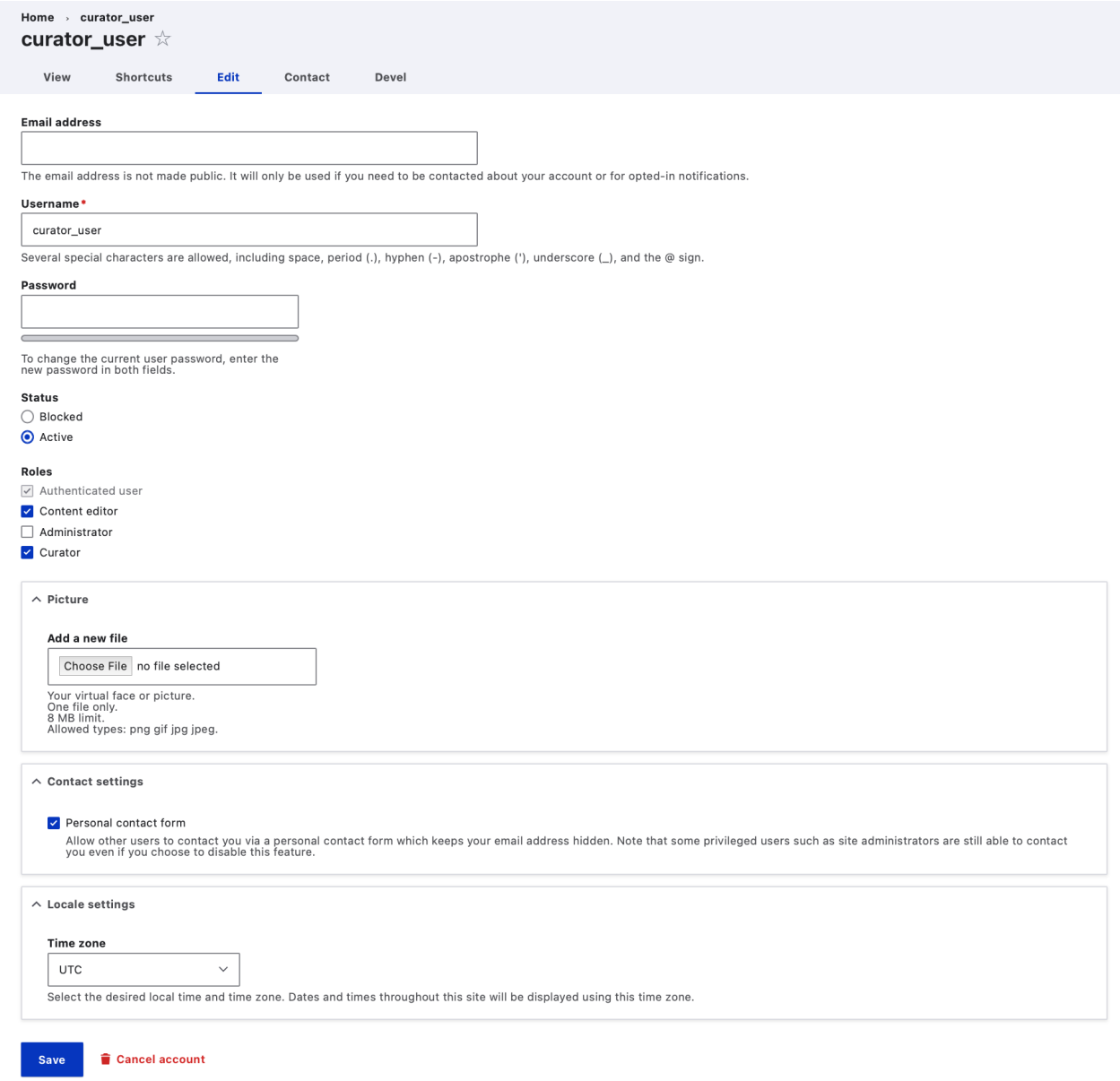

Curator Roles are now assigned to the users under Roles.

## Edit User's Role

To Edit options for a user

**Home** -> **Administration** -> **People** -> **curator_user** -> **Edit** (under **Operations** column )

To remove the Content editor role for this user,




Uncheck Content Editor Role for example, make any other changes in this screen as required and Click Save.

### Permissions for Role to define collaborative groups

From the top menu -> **People** -> **Permissions**

Click in applicable checkboxes for **Content editor**.

**The following Tripal content sections are available to assign permission options for each Role :**

- Block
- Block Content
- Comment
- Configuration Manager
- Contact
- Contextual Links
- Devel
- Devel PHP
- Field UI
- File
- Filter
- Image
- Node
- Path
- Search
- Shortcut
- System
- Taxonomy
- Toolbar
- Tour
- Tripal
- Tripal Chado
- Update Manager
- User
- Views UI

Some of the checkboxes are already checked are some are not changeable.

An administrator can change the default permissions for roles. For example, to change the recently created role of *Curator*,

From the top menu click on -> **People** -> **Permissions**.

In this screen individual permissions can be set for a Role by the administrator viewing the permissions checked for other roles.

Here are some recommended permissions checked for the Role of the Curator in the File, Node and Tripal categories:

## File permissions

| Permission | Anonymous user | Authenticated user | Content editor | Administrator | Curator |
|---|---|---|---|---|---|
| **File** | | | | | |
| Access the Files overview page | ☐ | ☐ | ☑ | ☑ | ☐ |
| Delete any file<br>*Warning: Give to trusted roles only; this permission has security implications.* | ☐ | ☐ | ☐ | ☑ | ☐ |
| Delete own files | ☐ | ☑ | ☑ | ☑ | ☑ |

## Node permissions

| Permission | Anonymous user | Authenticated user | Content editor | Administrator | Curator |
|---|---|---|---|---|---|
| **Node** | | | | | |
| *Article*: Create new content | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Basic page*: Create new content | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Article*: Delete any content | ☐ | ☐ | ☐ | ☑ | ☐ |
| *Basic page*: Delete any content | ☐ | ☐ | ☐ | ☑ | ☐ |
| *Article*: Delete own content<br>Note that anonymous users with this permission are able to delete any content created by any anonymous user. | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Basic page*: Delete own content<br>Note that anonymous users with this permission are able to delete any content created by any anonymous user. | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Article*: Delete revisions<br>To delete a revision, you also need permission to delete the content item. | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Basic page*: Delete revisions<br>To delete a revision, you also need permission to delete the content item. | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Article*: Edit any content | ☐ | ☐ | ☐ | ☑ | ☐ |
| *Basic page*: Edit any content | ☐ | ☐ | ☐ | ☑ | ☐ |
| *Article*: Edit own content<br>Note that anonymous users with this permission are able to edit any content created by any anonymous user. | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Basic page*: Edit own content<br>Note that anonymous users with this permission are able to edit any content created by any anonymous user. | ☐ | ☐ | ☑ | ☑ | ☑ |
| *Article*: Revert revisions<br>To revert a revision, you also need permission to edit the content item. | ☐ | ☐ | ☐ | ☑ | ☑ |
| *Basic page*: Revert revisions<br>To revert a revision, you also need permission to edit the content item. | ☐ | ☐ | ☐ | ☑ | ☑ |
| *Article*: View revisions<br>To view a revision, you also need permission to view the content item. | ☐ | ☐ | ☐ | ☑ | ☑ |
| *Basic page*: View revisions<br>To view a revision, you also need permission to view the content item. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Access the Content overview page | ☐ | ☐ | ☑ | ☑ | ☑ |

### Tripal permissions

| Permission | Anonymous user | Authenticated user | Content editor | Administrator | Curator |
|---|---|---|---|---|---|
| **Tripal** | | | | | |
| Access Tripal Content listing<br>Allows the user to access the Tripal content listing. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Access Tripal Importer Admin Listing<br>Allow access to the listing of importers available through Tripal. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Administer Tripal Content<br>*Warning: Give to trusted roles only; this permission has security implications.* Allows users to access the Tripal Content listing and add, edit, delete Tripal content of any type. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Administer Tripal Managed Files<br>Allows the user to administer file upload maximum size, user quotas and file expiration dates. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Create new Tripal Content | ☐ | ☐ | ☐ | ☑ | ☑ |
| Delete Tripal Content | ☐ | ☐ | ☐ | ☑ | ☑ |
| Edit Tripal Content | ☐ | ☐ | ☐ | ☑ | ☑ |
| Make Tripal Data Files Permanent<br>Allows the user to make files they have access to permanent. Files that are permanent will not expire and will not be cleaned up. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Manage Own Tripal Data Collections<br>Allows users to manage their own Tripal data collections. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Manage Own Tripal Data Files<br>Allows users to manage their own data files uploaded via Tripal. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Manage Tripal Jobs<br>Allows the user to access the job management pages. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Manage Tripal content types<br>*Warning: Give to trusted roles only; this permission has security implications.* Allows users to access the Tripal Content Types listing and add, edit or delete items listed. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Publish Tripal Content<br>Allows the user to publish Tripal content of all Tripal Content Types for online access. | ☐ | ☐ | ☐ | ☑ | ☐ |
| Tripal Importer: Use the *Chado FASTA File Loader*<br>*Warning: Give to trusted roles only; this permission has security implications.* Allow the user to import data using the *Chado FASTA File Loader*. Note: you may also need to give the "Upload Files" permission for importers to work. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Tripal Importer: Use the *Chado GFF3 File Loader*<br>*Warning: Give to trusted roles only; this permission has security implications.* Allow the user to import data using the *Chado GFF3 File Loader*. Note: you may also need to give the "Upload Files" permission for importers to work. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Tripal Importer: Use the *Newick Tree Loader*<br>*Warning: Give to trusted roles only; this permission has security implications.* Allow the user to import data using the *Newick Tree Loader*. Note: you may also need to give the "Upload Files" permission for importers to work. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Tripal Importer: Use the *OBO Vocabulary Loader*<br>*Warning: Give to trusted roles only; this permission has security implications.* Allow the user to import data using the *OBO Vocabulary Loader*. Note: you may also need to give the "Upload Files" permission for importers to work. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Tripal Importer: Use the *Taxonomy Loader*<br>*Warning: Give to trusted roles only; this permission has security implications.* Allow the user to import data using the *Taxonomy Loader*. Note: you may also need to give the "Upload Files" permission for importers to work. | ☐ | ☐ | ☐ | ☑ | ☑ |
| Upload Tripal Data Files<br>Allows the user to upload files using Tripal's HTML5 loader. | ☐ | ☐ | ☐ | ☑ | ☑ |
| View Tripal Administration Pages<br>*Warning: Give to trusted roles only; this permission has security implications.* Allow the user to access administrative pages of Tripal. This includes the menus and listings (e.g. jobs, storage systems, extensions) but not management of the various subsystems as they have their own permissions. | ☐ | ☐ | ☐ | ☑ | ☐ |
| View Tripal Content | ☐ | ☐ | ☐ | ☑ | ☑ |

Permissions checked for the Curator role shown in screenshots above help in editing, revising and reverting content in addition to several others not available to other Roles for importing content into Tripal, edit and maintain them.

Site administrators wanting to allow their curators to delete Tripal content can do so by applying the “Delete Tripal Content” permission. If their curator also imports data via available custom data importers like GFF3 importer they may want to assign the Tripal Importer permissions, publish and “Upload Tripal Data files”.

#### Permissions by Term

The Permissions by Term is a module that extends Drupal by providing functionality for restricting view access to single nodes via taxonomy terms. This module can be useful for Tripal users interested in creating, documenting and maintaining Ontologies, for example.

Taxonomy term permissions can be coupled to specific user accounts and/or user roles. It relies on the entities, which are shipped traditionally with Drupal core: taxonomy terms and nodes.

More information is available at https://www.drupal.org/docs/contributed-modules/permissions-by-term and https://www.drupal.org/project/permissions_by_term.

An example use-case in Tripal is Sub-editors working on a research publication. Collecting content together in a taxonomy term allows you to manage that content as a sub site and assign its own administrator. This is useful where you might need someone to produce lots of different types of content but only want them to be able to add it to a specific area of the website that is working on the publication.

Sub-communities within a membership organisation. The topics a membership organisation may cover can be very broad and individual members may only be interested in seeing content from a sub-selection of the areas it covers. The sub-community may have their own executive members who can contribute to the research topic or approve new members to their sub-community.

**Additional Resources:**

- Official Drupal Docs: What are Users, Roles, and Permissions?
- Official Drupal Docs: Creating a Role
- Official Drupal Docs: Assigning Permissions to a Role
- Official Drupal Docs: Changing a User's Roles
- Official Drupal Docs: Creating a User Account

## 4.5 Database Backups

### 4.5.1 The Importance of Backups

Drupal, on which Tripal is based, is not like a word processor or spreadsheet, there is no "Undo" function. For this reason, and to guard against hard drive failures, etc. it is essential to maintain current backups of your PostgreSQL database.

### 4.5.2 How to make a backup

You can use the PostgreSQL command `pg_dump` to make a backup of your database. Here is a simple script that makes a database backup, and saves it in a file which is named using today's date. You would of course substitute your site's hostname, database name, file path, username, and password.

```
#!/bin/bash

file="/var/www/drupal9/web/$(date +%Y%m%d).sql"
echo "Backup Tripal database to \"$file\""

PGPASSWORD=drupaldevelopmentonlylocal \
pg_dump \
```

(continues on next page)

(continued from previous page)

```
  --verbose \
  --format=plain \
  --username="drupaladmin" \
  --host=localhost \
  "sitedb" \
  > "$file"
```

You could then run this script daily to make sure you always have a current database backup. One way to do that is to use cron on your system. Although setting that up is beyond the scope of this documentation, you may find this tutorial by GeeksForGeeks helpful.

### 4.5.3 How to restore from a backup

For convenience we can first specify the PostgreSQL password once so we don't need to enter it for each step. You may prefer a more secure authentication method, but for learning Tripal on a local system this is the simplest method. Substitute your password here.

```
export PGPASSWORD=drupaldevelopmentonlylocal
```

In order to restore from a backup, we first need to remove the current database, and create a new empty database. **This will erase all data**, so make sure you successfully made a backup first!

```
dropdb \
  --username="drupaladmin" \
  --host=localhost \
  --if-exists sitedb

createdb \
  --username="drupaladmin" \
  --host=localhost \
  sitedb -O drupaladmin
```

Restoring data into the database is then performed as follows, use the file name generated by the previous backup step, for example it might be `20240506.sql`.

```
psql \
  --set ON_ERROR_STOP=on \
  --username="drupaladmin" \
  --host=localhost \
  --dbname=sitedb \
  < substitutefilenamehere.sql
```

While not essential, for your convenience you may want to place chado in the default search path. To do that, execute this command:

```
psql -h localhost -U drupaladmin -d sitedb \
  -c "ALTER DATABASE sitedb SET search_path = '$user', public, chado"
```

Finally, it is recommended to rebuild the Drupal caches

```
drush cache:rebuild
```

### 4.5.4 Best Practices

If you are just learning Tripal, we recommend you start out with a *Tripal Docker* container. This makes initial installation as easy as possible, and if you make mistakes with your site, it is easy to start over with a new clean starting point. You can also backup and restore the database inside your docker container as described earlier. Another approach when using docker is to use docker commit to create an image from a running container at the point you want to save. Then you can use the docker run command with this committed image in order to start a fresh site at the exact same point.

If you have a publicly facing web site, which we usually call a "Production" site, it is highly recommended to also have a "Staging" or "Testing" site. Here you can load a database backup from your production site, and then test new loaders or procedures on the staging site without danger of harming your production site. Once your procedures are verified as working correctly, only then do you make changes to your production site.

CHAPTER

FIVE

# EXTENDING TRIPAL

## 5.1 Object-Oriented Development

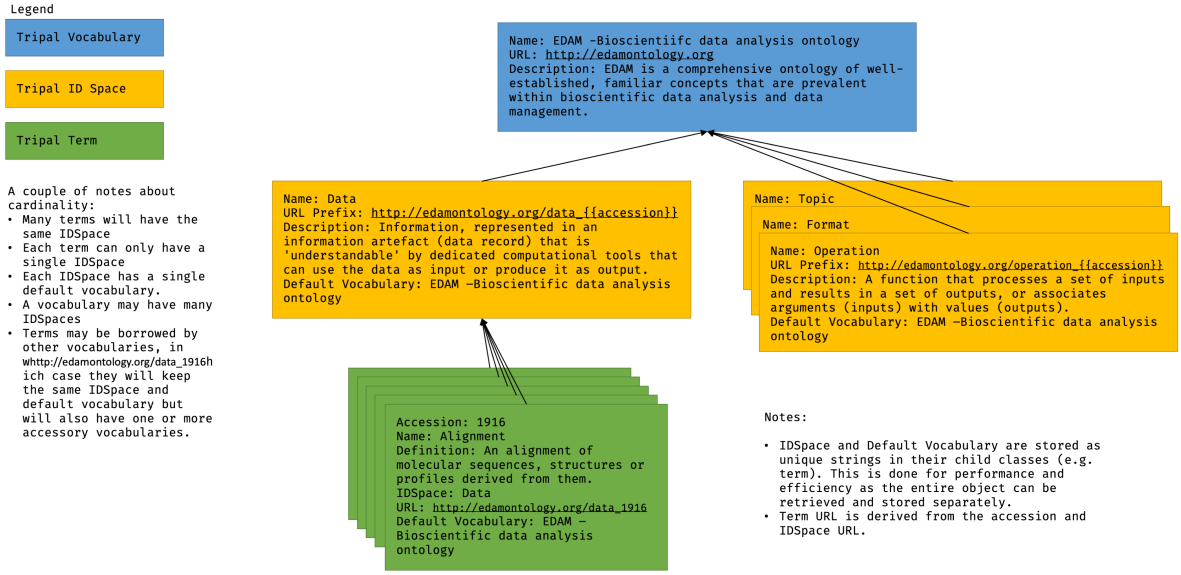
While object-oriented development may be initially daunting, it is now well established as a best practice. For a general overview of PHP best practices, read through phptherightway.com. Additionally, Drupal lists a multitude of resources on their Background to Custom Module Development documentation. From the Tripal perspective, it is important to note that Tripal 4 follows the object-oriented design patterns laid out by Drupal.

## 5.2 Controlled Vocabularies (CVs)

Controlled vocabularies are simply a collection of agreed upon names (knowns as terms) for items of interest. Within biology this may mean we have a controlled vocabulary describing the parts of a gene or the types of germplasm. And ontology is a specialized type of controlled vocabulary with additional structure including relationships between terms (i.e. the sequence ontology).

Tripal and Chado both use controlled vocabulary terms extensively to categorize data and metadata. The use of controlled vocabulary terms also allows both Tripal and Chado to be extremely flexible while also remaining very descriptive with rich metadata.

As you can see in the figure below, controlled vocabularies are collections of 1+ ID spaces and each ID Space is a collection of terms. The ID space provides the unique namespace for the term accession and a controlled vocabulary groups a bunch of similar terms.




In Tripal 4, the Controlled Vocabulary API is designed to enable flexible backend storage and thus can be completely independent of Chado. That said, there is a default implementation of this API provided by the core Tripal Chado module that tightly integrates Tripal and Chado.

### 5.2.1 How are CVs used in Tripal?

Controlled vocabulary terms (CVTerms) are used to define Tripal Content Types. Additionally, all Tripal Fields are defined using a controlled vocabulary term. As such, all biological content managed by Tripal is associated with a categorizing controlled vocabulary term and each piece of metadata defining a single piece of content is also defined using a controlled vocabulary. This ensures that Tripal content is semantic web ready, as well as, ensuring it is well organized for both researchers and computer software.

### 5.2.2 Identifying a CVTerm

Before creating a new content type for your site you must identify a CVTerm that best matches the content type you would like to create. CVs are plentiful and at times selection of the correct term from the right vocabulary can be challenging. If there is any doubt about what term to use, then it is best practice to reach out to others to confirm your selection. The Tripal User community is a great place to do this by posting a description of your content type and your proposed term on the Tripal Issue Queue. Confirming your term with others will also encourage re-use across Tripal sites and improve data exchange capabilities.

The EBI's Ontology Lookup Service is a great place to locate terms from public vocabularies. At this site you can search for terms for your content type. If you can not find an appropriate term in a public vocabulary or via discussion with others then you create a new **local** term within the **local** vocabulary that comes with Tripal.

> **Warning:** Creation of **local** terms is discouraged but sometimes necessary. When creating local terms, be both precise and verbose in your description.

### 5.2.3 Retrieving Tripal Terms

Terms can be retrieved by

1. Using the Tripal Vocabulary Manager to search terms by name within a vocabulary.
2. Using the Tripal ID Space Manager to search terms by name or accession within an ID Space.
3. Using the Tripal ID Space Manager to get parent or child terms of an existing term.
4. Using the static TripalTerm::suggestTerms method to search by name across vocabularies and ID spaces.

Table 1: Ways to search for TripalTerms

| Class | Method | Return | Search Property | IDSpace* | Vocabulary* |
|---|---|---|---|---|---|
| TripalTerm | suggestTerms+ | array | name | No | No |
| TripalVocabulary | getTerms | array | name | No | Yes |
| TripalIdSpace | getTerms | array | name | Yes | Yes |
| TripalIdSpace | getTerm | TripalTerm | accession | Yes | Yes |
| TripalIdSpace | getParent | TripalTerm | TripalTerm | Yes | Yes |
| TripalIdSpace | getChildren | array | TripalTerm | Yes | Yes |

** These two columns indicate information which is required for the search.*
*+ This is a static method of TripalTerm.*

### 5.2.4 Chado CV module

In Chado, controlled vocabularies and ontologies are stored in the CV Module. This module provides flexible storage of the individual terms, as well as, the relationships between them.

## 5.3 Biological Data Storage

A critical component of Tripal is interfacing between biological data stores and Drupal.

The Tripal Chado core module integrates Drupal with the GMOD Chado schema to provide a great foundation of support for biological data with the ability to store sequence, sequence comparisons, germplasm, phenotypes, genotypes, ontologies, publications, and phylogeny, as well as associated metadata and relations for each of these data types. Integration with Chado allows Tripal to support most biological data types out of the box and use of a common schema allows Tripal sites to share data and Tripal extension modules to be built to enforce best practices.

That said, there are still situations in which you may want to support additional data storage backends such as graph databases, NoSQL databases, or flat files (e.g. variant call format: VCF). Tripal seamlessly supports additional data storage backends through a variety of APIs which will be described here.

### 5.3.1 Tripal DBX: Generic cross database support for Drupal

Tripal has always focused on providing cross database schema support. From the very first version, this support focused on providing integration between Drupal and the GMOD Chado Schema. With Tripal 4, we have made a generic base for our Chado integration which is known as Tripal DBX. This ensures that we can still provide high integration between Drupal and Chado, while also proving a really solid, well-documented API for additional biological data storage options.

Tripal DBX extends the Drupal Database API. Specifically, it extends two core Drupal abstract classes:

- DrupalCoreDatabaseConnection: a Drupal-specific extension of the PDO database abstraction class in PHP.
- DrupalCoreDatabaseSchema: provides a means to interact with a schema including management tasks such as adding tables.

Currently Tripal DBX relies on the Drupal PostgreSQL implementations of these classes (PostgreSQL Connection and PostgreSQL Schema), although there is structure in place to expand it to other Drupal database drivers.

#### Tripal DBX Connection

There are two main parts of the Drupal Connection class that the Tripal DBX Api overrides:

**A) One Database Connection per Instance:**

Drupal attempts to minimize the number of connections to the database by only creating a new connection if one does not already exist. This makes sense in the context of a single database schema; however, when dealing with multiple schema you will often need to make changes to the search path in order to access the tables of each schema. This API creates an independent connection to the database for each TripalDbxConnection instance which ensures the primary Drupal connection remains unaffected by search path changes.

**B) Table Prefixing in Queries:**

Drupal has always had support for prefixing table names; however, this has been a simple string prepended to the beginning of the table name. This is likely due to many database systems not supporting multiple schema within a single database. In version 2 and 3, Tripal used the native Drupal prefixing to access the Chado database in a separate schema. It did this by using a prefix of "chado." which takes advantage of the PostgreSQL syntax for accessing a specific schema (named "chado" in the previous example). In Tripal version 4, we've created the Tripal DBX API which takes this one step further by extending the native PHP/Drupal PDO database layer to use cross schema focused table prefixing. This allows module developers to access multiple chado and additional Tripal DBX managed schema both through direct queries and using the object-oriented query builder.

The PHP/Drupal PDO query is very useful for building dynamic queries as it handles user provided parameters in a very secure manner and can be very accessible for those who are less familar with PostgreSQL syntax. The following example shows how the native Drupal query builder can be used with Chado through the Tripal DBX API:

```
// Open a Connection to the default Tripal DBX managed Chado schema.
$connection = \Drupal::service('tripal_chado.database');

// Start a select query on the Chado feature table and assign an alias of x.
$query = $connection->select('feature', 'x');

// Add a where condition that feature.is_obsolete is FALSE.
$query->condition('x.is_obsolete', 'f', '=');

// Select the name and residues columns/fields from the table.
$query->fields('x', ['name', 'residues']);

// And only show the first 10 records/entries.
```

(continues on next page)

(continued from previous page)

```
$query->range(0, 10);

// Finally execute the query we generated against the database.
$result = $query->execute();

// And iterate through the returned results.
foreach ($result as $record) {
  // Do something with the $record object here.
  // e.g. echo $record->name;
}
```

The above example used a Chado implementation of the Tripal DBX API provided by the `tripal_chado.database` service to generate a select query, execute it agains the database focusing on a specific non-Drupal schema and then iterates through the results. It is the equivalent of the following SQL statement: `SELECT x.name, x.residues FROM chado.feature x WHERE x.is_obsolete = f LIMIT 10` if the default Chado schema is named `chado`.

---

**Note:** For more information on the Drupal query builder, See the Drupal.org documentation. There is full support for all the documented Drupal functionality with Tripal DBX managed schema.

---

#### — Multiple Schema Support

Tripal DBX provides multiple database schema support through table prefixing. The first step is to set the schema you are working on in your specific connection. For example, if you were working with two Chado schema (named "chado1" and "chado2" in this example) in addition to the Drupal schema then you would use `setSchemaName()` to specify your main schema and then `addExtraSchema()` to specify any additional ones.

```
$connection = \Drupal::service('tripal_chado.database');
$connection->setSchemaName('chado1');
$connection->addExtraSchema('chado2');
```

---

**Note:** The primary schema indicated using `setSchemaName()` can be decided in a number of ways depending on your use case for multiple schema and the specific query you are executing. The rule of thumb is to make the primary schema match the one "prepared" to work with Tripal (i.e. the schema used as a base for Tripal Entities).

---

Now that you have your connection set up indicating the schema you are interested in, you can use the query builder to generate as many queries as you need within the current scope. For example, the following code will generate a query returning chromosome features stored in a separate chado schema (i.e. `chado2`) and using the primary chado schema (i.e. `chado1`) for organism + ontology information:

```
// Start a select query on the feature table in the chado2 schema.
// Note the schema is indicated by prefixing a "2:" on the table name.
$query = $connection->select('2:feature', 'f');

// Add a join to the organism + cvterm table in the chado1 schema.
// Note that no prefix is needed for the primary Tripal DBX managed schema.
$query->join('organism', 'o', 'o.organism_id = f.organism_id');
$query->join('cvterm', 'cvt', 'cvt.cvterm_id = f.type_id');

// Add a where clause ensuring only records associated with the Tripalus genus are␣
↪returned.
```

(continues on next page)

(continued from previous page)

```
$query->condition('o.genus', 'Tripalus', '=');

// Add a where clause ensuring only "chromosome" feature types are returned.
$query->condition('cvt.name', 'chromosome', '=');

// Select the feature feature_id, name + uniquename and the organism genus, species +␣
↪common name.
$query->fields('f', ['feature_id', 'name', 'uniquename']);
$query->fields('o', ['genus', 'species', 'common_name']);

// Finally execute the query we generated against the database.
$result = $query->execute();

// And iterate through the returned results.
foreach ($result as $record) {
  // Do something with the $record object here.
  // e.g. echo $record->name;
}
```

**Note:** This API expects all table names to be wrapped in curly brackets with an integer indicating the schema the table is in. For example, `{1: feature}` would indicate the feature table in the current Tripal DBX managed schema, `{0: system}` would indicate the Drupal system table and additional numeric indices would be used for extra Tripal DBX managed schema (i.e. `{2: feature}`).

Alternatively, if you have a specific query in mind and do not need the security or overhead of the query builder, then you can use the Drupal `query()` method to execute it directly. The following example shows how you would execute the equivalent query built by the query builder above:

```
// Set some variables or retrieve them from your users.
$type = 'chromosome';
$genus = 'Tripalus';

// The SQL statement to be executed.
// Note that we've used the {1:organism} and {2:feature} for the primary and extra␣
↪schemas respectively.
// Also note that placeholders (i.e. :type) are used for user input.
$sql = 'SELECT f.feature_id, f.name, f.uniquename, o.genus, o.species, o.common_name
        FROM {2:feature} f
        LEFT JOIN {1:organism} o ON o.organism_id=f.organism_id
        LEFT JOIN {1:cvterm} cvt ON cvt.cvterm_id=f.type_id
        WHERE o.genus = :genus AND cvt.name = :type';

// Finally execute the query we generated against the database
// by providing the values for any placeholders.
$results = $connection->query($sql, [':genus' => $genus, ':type' => $type]);

// And iterate through the returned results.
foreach ($results as $record) {
  // Do something with the $record object here.
  // e.g. echo $record->name;
}
```

**Warning:** When using the `query` method to submit SQL statements directly, it is very important to be aware of security and the source of any information. Variables should NEVER be embedded directly in the SQL and all dynamic and/or user input should be handled using placeholders in the SQL statement and then provided when the query is executed.

---

**Note:** The `query` method shown for multiple schema can also be used for single schema queries as an alternative to the query builder. As indicated in the query builder, for a single schema the `{tablename}` can be used and the `1:` prefix omitted.

---

#### Tripal DBX Schema

---

**Note:** This class should not be instantiated directly but rather it should be accessed through a TripalDbxConnection object using the schema() method. This is to avoid issues when the default Tripal DBX managed schema name is changed in the TripalDbxConnection object which could lead to issues.

**Warning:** If you choose to instantiate a TripalDbxSchema object yourself, you are responsible to not change the Tripal DBX managed schema name of the connection object used to instantiate this TripalDbxSchema.

---

This class provides a Tripal-specific implementation of the Drupal Schema abstract class. The Drupal PostgreSQL (and other database driver) implementations of the base Drupal Schema class follow the assumption that there is a single schema. As such the core Drupal implementations focus on managing tables within a single schema.

The TripalDBXSchema class extends that table-management functionality to also include schema-focused management including creation, cloning, renaming, dropping and definition export. Additionally, it removes the assumption of a single schema by allowing the default schema to be set based on a Tripal DBX connection.

### 5.3.2 GMOD Chado Schema Integration

The Tripal Chado module provides integration between Tripal and the GMOD Chado schema. This provides flexible, default storage for many biological data types including genes, genetic markers, germplasm, as well as associated meta data such as project and analysis details. More specifically:

> Chado is a relational database schema that underlies many GMOD installations. It is capable of representing many of the general classes of data frequently encountered in modern biology such as sequence, sequence comparisons, phenotypes, genotypes, ontologies, publications, and phylogeny. It has been designed to handle complex representations of biological knowledge and should be considered one of the most sophisticated relational schemas currently available in molecular biology. The price of this capability is that the new user must spend some time becoming familiar with its fundamentals.
>
> —GMOD Chado Documentation

Chado was selected for Tripal because it is open-source, it is maintained by the community in which anyone can provide input, and use of Chado encourages common data storage between online biological sites which decreases duplication of effort.

Chado is meant to be installed into a PostgreSQL database and is designed to house a variety of biological data. For example, Tripal comes with a variety of content types. However, if you want to create new content types you must know how that data will be stored in Chado. Additionally, use of the Bulk Loader (a tab-delimited data loader for custom

data formats) requires a good understanding of Chado. Finally, creating extensions to Tripal requires an understanding of Chado to write SQL and or new Tripal fields. The following links provide training for Chado.

| Resource | Link |
|---|---|
| Chado Home Page | http://gmod.org/wiki/Chado |
| Chado Tutorial | http://gmod.org/wiki/GMOD_Online_Training_2014/Chado_Tutorial |
| Chado ReadtheDocs | https://chado.readthedocs.io/en/rtd/ |
| Chado Table List | https://chado.readthedocs.io/en/rtd/_static/schemaspy_integration/index.html |
| Chado Best Practices | http://gmod.org/wiki/Chado_Best_Practices |
| Chado GitHub | https://github.com/GMOD/Chado |

### Chado Installation

When you install the Tripal Chado module you will be automatically prompted to install Chado. This creates a schema within your Drupal database to house the Chado tables listed in the resources above. To install Chado manually navigate to Structure > Tripal > Data Storage > Chado > Install Chado. Then just choose your version and run the associated Tripal job.

If you need to install Chado programmatically, use the following service from within a fully bootstrapped Tripal site.

Listing 1: Installs Chado version 1.3 in a schema named 'chado'.

```
$installer = \Drupal::service('tripal_chado.chadoInstaller');
$installer->setSchema('chado');
$success = $installer->install(1.3);
```

Alternatively, you can install chado via the command line using the following Drush command.

Listing 2: Installs Chado version 1.3 in a schema named 'chado' using Drush.

```
drush trp-install-chado --schema-name='chado' --version=1.3
```

### Tripal Vocabularies & Terms

Tripal Vocabularies and Terms are database agnostic and store their details in the Drupal database as controlled by the Drupal Entity API. Tripal has implemented a TripalTermStorage Plugin to allow Tripal extension modules to provide additional storage for Tripal Vocabularies, IDSpaces and Terms. The core Tripal Chado module has implemented this plugin to ensure these Tripal entities are linked to their Chado equivalents.

The following describes the mapping between Tripal Entities and their Chado counterparts:

- Tripal Vocabularies (see TripalVocab class) = cv
  - TripalVocab::namespace = cv.name
  - TripalVocab::name = cv.definition
  - TripalVocab::url = db.url
- Tripal Vocabulary IDSpaces (see TripalVocabSpace class) = db
  - TripalVocabSpace::name = db.name
  - TripalVocabSpace::description = db.description

  - TripalVocabSpace::urlprefix = db.urlprefix

- Tripal Terms (see TripalTerm class) = cvterm and dbxref
  - TripalTerm::name = cvterm.name
  - TripalTerm::definition = cvterm.definition
  - TripalTerm::accession = dbxref.accession

For information on this mapping on a per entity basis, the chado details have been added to the Tripal entities. The following examples show how to access them.

```
// First, retrieve the Tripal Vocabulary object.
$tripalvocab = \Drupal::service('tripal.tripalVocab.manager')->getVocabularies([
        'namespace' => 'sequence',
]);

// Now access the cv.cv_id from that object.
$cv_id = $tripalvocab->chado_record_id;

// You can also access the chado record directly...
$cv = $tripalvocab->chado_record;

// The same pattern holds true for IDspaces and Terms.
$db_id = $tripalIDSpace->chado_record_id;
$db = $tripalIDSpace->chado_record;
$cvterm_id = $tripalTerm->chado_record_id;
$cvterm = $tripalTerm->chado_record;
```

### 5.3.3 Bulk Schema Install for PostgreSQL

Tripal provides a service to bulk import SQL files into a PostgreSQL database. The following code block shows how this service can be used to create a fictional database schema and populate it within your Drupal/Tripal database. It's recommended to use a schema within your Drupal/Tripal database to ensure you can easily join between your biological and application data. That said, this is not a requirement as Tripal can support data from outside sources.

```
$schema_name = 'biodb'; // This is a fictional example schema name.
// First initialize the Tripal Services.
$service = \Drupal::service('tripal.bulkPgSchemaInstaller');
// Next Create the schema.
$success = $service->createSchema($schema_name);
// And if the schema exists...
if ($service->checkExists($schema_name)) {
        // Then populate the schema in bulk using an SQL file.
        $sql_file = \Drupal::service('extension.list.module')->getPath('tripal') . '/
↪tests/fixtures/smallTestSchema.sql';
        $success = $service->applySQL($sql_file, $schema_name, TRUE);
}
```

### 5.3.4 Custom Tables in Chado

The Chado database provides a variety of tables for storing biological and ancillary data, however there may be situations where you need to make customizations to Chado to hold new types of data or to link existing data. Rather than make Changes to existing Chado tables, it is recommended to develop custom tables. Modules can provide such tables during installation and provide the necessary tools to interact with the data in the custom table (e.g., new data loaders, content types, materialized views, and/or fields). This section describes how to programmatically create and manage Custom tables in Chado.

**Warning:** You should avoid making any changes to existing Chado tables as it could make upgrades to future releases of Chado more difficult and could break functionality in Tripal that expects Chado tabes to be a certain way. Instead, use custom tables!

To support custom tables in Chado, Tripal provides a service that has management functions for working with custom tables in general, and provides the *ChadoCustomTable* object which has support for working with individual tables. Each table can be identified using its name and a unique ID that is automatically assigned to the Table.

**Note:** For a hands-on example to programmatically create and manage custom tables see *How to use Custom Tables in Chado*.

### 5.3.5 Creating a Chado Data Importer

Often we want to provide a user interface by which a site developer can easily import data into Chado. Examples of existing loaders compatible with Tripal include the FASTA and GFF3 loaders that come with the Tripal Genome module. These loaders allow users to import data into Chado that are in FASTA or GFF3 format.

If you would like to create a new data importer for Chado you will need to create your own **TripalImporter** plugin. The Tripal importers use the Drupal Plugin API. The plugin infrastructure of Drupal allows a module to provide new functionality that builds off of a common interface. For importing data this interface is provided by the `TripalImporterInterface` class. The **TripalImporter** plugin provides many conveniences. For example, it provides an input form that automatically handles files uploads, it performs job submission, logging, and provides progress updates during execution. Adding a **TripalImporter** plugin to your module will allow anyone who installs your module to also use your new loader!

Here, we will show how to create a **TripalImporter** plugin by building a simple importer called the **ExampleImporter**. This importer reads a comma-separated file containing genomic features and their properties (a fictional "Test Format" file). The loader will split each line into feature and property values, and then insert each property into the `featureprop` table of Chado using a controlled vocabulary term (supplied by the user) as the `type_id` for the property.

**Note:** Prior to starting your data loader you should plan how the data will be imported into Chado. Chado is a flexible database schema and it may be challenging at times to decide in to which tables data should be placed. It is recommended to reach out to the Chado community to solicit advice. Doing so will allow you to share your loader will other Tripal users more easily!

### Step 1: Create Your Module

To create your an importer, you first need to have a Drupal module in which the loader will be provided. If you do not know how to create a module, see the section titled *Custom Module Development* for further direction. For this document we will describe creation of an importer in a fake module named `tripal_example_importer`.

### Step 2: Create the Importer Class File

To define a new `TripalImporter` plugin, you should first create the directory `src/Plugin/TripalImporter/` in your module. For the example here, we will create a new plugin named `ExampleImporter`. We must name the file the same as the class (with a `.php` extension) and place the file in the `src/Plugin/TripalImporter/` directory we just created. For example: `tripal_example_importer\src\Plugin\TripalImporter\ExampleImporter.inc`. Placing the importer class file in the `src/Plugin/TripalImporter` directory is all you need for Tripal to find it. Tripal will automatically place a link for your importer on the Drupal site at **admin > Tripal > Data Loaders**.

### Step 3: Stub the Class File

In the Class file created in the previous step we will write our **TripalImporter** plugin. First, we must set the namespace for this class. It should look similar to the path where the file is stored (but without the `src` directory):

```
<?php
namespace Drupal\tripal_example_importer\Plugin\TripalImporter;
```

Next, an importer that is meant to load data into Chado should extend the `ChadoImporterBase` class.

```
<?php
namespace Drupal\tripal_example_importer\Plugin\TripalImporter;

use Drupal\tripal_chado\TripalImporter\ChadoImporterBase;

class ExampleImporter extends ChadoImporterBase {
```

All **TripalImporter** plugins uss the `TripalImporterInterface` which requires that several unctions are included in your plugin. These functions are:

- `form()`
- `formSubmit()`
- `formValidate()`
- `run()`
- `postRun()`

We will discuss each fucntion later but for now, we will create "stubs" for each of these functions in our class. For our example empty class it should look like the following:

```
<?php
namespace Drupal\tripal_example_importer\Plugin\TripalImporter;

use Drupal\tripal_chado\TripalImporter\ChadoImporterBase;

class ExampleImporter extends ChadoImporterBase {

```

(continues on next page)

(continued from previous page)

```
  /**
   * @see ChadoImporterBase::form()
   */
  public function form($form, &$form_state) {

    // Always call the parent form.
    $form = parent::form($form, $form_state);

    return $form;
  }

  /**
   * @see ChadoImporterBase::formSubmit()
   */
  public function formSubmit($form, &$form_state) {

  }

  /**
   * @see ChadoImporterBase::formValidate()
   */

  public function formValidate($form, &$form_state) {

  }

  /**
   * @see ChadoImporterBase::run()
   */
  public function run() {

  }

  /**
   * @see ChadoImporterBase::postRun()
   */
  public function postRun() {

  }
}
```

Notice in the `form()` function there is a call to the `parent::form()`:

```
/**
 * @see ChadoImporterBase::form()
 */
public function form($form, &$form_state) {

  // Always call the parent form.
  $form = parent::form($form, $form_state);
```

(continues on next page)

(continued from previous page)

```
  return $form;
}
```

Without calling the `parent::form()` function your importer's form may not properly work. This is required.

### Step 4: Add Class Annotations

All Drupal plugins require an Annotation section that appears as a PHP comment just above the Class definition. The annotation section provides settings that the **TripalImporter** plugin requires. As a quick example here is the Annotation section for the GFF3 importer. The GFF3 importer is provided by the Tripal Genome module and imports features defined in a GFF3 file into Chado.

```
/**
*  GFF3 Importer implementation of the ChadoImporterBase.
*
*  @TripalImporter(
*    id = "chado_fasta_loader",
*    label = @Translation("Chado FASTA File Loader"),
*    description = @Translation("Import a FASTA file into Chado"),
*    file_types = {"fasta","txt","fa","aa","pep","nuc","faa","fna"},
*    upload_description = @Translation("Please provide a plain text file following the
↪<a target='_blank' href='https://en.wikipedia.org/wiki/FASTA_format'>FASTA format␣
↪specification</a>."),
*    upload_title = @Translation("FASTA File"),
*    use_analysis = True,
*    require_analysis = True,
*    use_button = True,
*    button_text = @Translation("Import FASTA file"),
*    file_upload = True,
*    file_remote = True,
*    file_local = True,
*    file_required = True,
*    submit_disabled = False
*  )
*/
class GFF3Importer extends ChadoImporterBase {
```

In the code above, the annotation section consists of multiple settings in key/value pairs. The meaning of each settings is as follows:

- `id`: A unique machine readable plugin ID for the loader. It must only contain alphanumeric characters and the underscore. It should be lowercase.
- `label`: the human readable name (or label) for this importer. It is wrapped in a `@Translation()` function which will allow Drupal to provide translations for it. This label is shown to the user in the list of available data importers.
- `description`: A short description for the site user that briefly indicates what this loader is for. It too is wrapped in a `@Translation()` function. This description is shown to the user for the loader.
- `file_types`: A list of file extensions that the importer will allow to be uploaded. If a file does not have an extension in the list then it cannot be uploaded by the importer.
- `upload_title`: Provides the title that should appear above the upload button. This helps the user understand what type of file is expected.

- `upload_description`: Provides the information for the user related to the file upload. You can provide additional instructions or help text.
- `use_analysis`: To support FAIR data principles, we should ensure that provenance of data is available. Chado provides the `analysis` table to link data to an analysis. The analysis record provides the details for how the data in the file was created or obtained. Set this to `False` if the loader should not require an analysis when loading. if `use_analysis` is set to `True` then the user will be presented with a form element to select an analysis and this analysis will be available to you for your importer.
- `require_analysis`: If the `use_analysis` value is set then this value indicates if the analysis should be required. If `True` it will be required, otherwise it will be optional.
- `button_text`: The text that should appear on the button at the bottom of the importer form.
- `use_button`: Indicates whether a submit button should be present. This should only be `False` in situations were you need multiple buttons or greater control over the submit process (e.g., multi-page forms).
- `submit_disabled`: Indicates whether the submit button should be disabled when the form appears. The form can then be programmatically enabled via AJAX once certain criteria is set.
- `file_upload`: Indicates if the loader should provide a form element for uploading a file.
- `file_remote`: Indicates if the loader should provide a form element for specifying the URL of a remote file.
- `file_local`: Indicates if the loader should provide a form element for specifying the path available to the web server where the file is located.
- `file_required`: Indicates if the file must be provided.

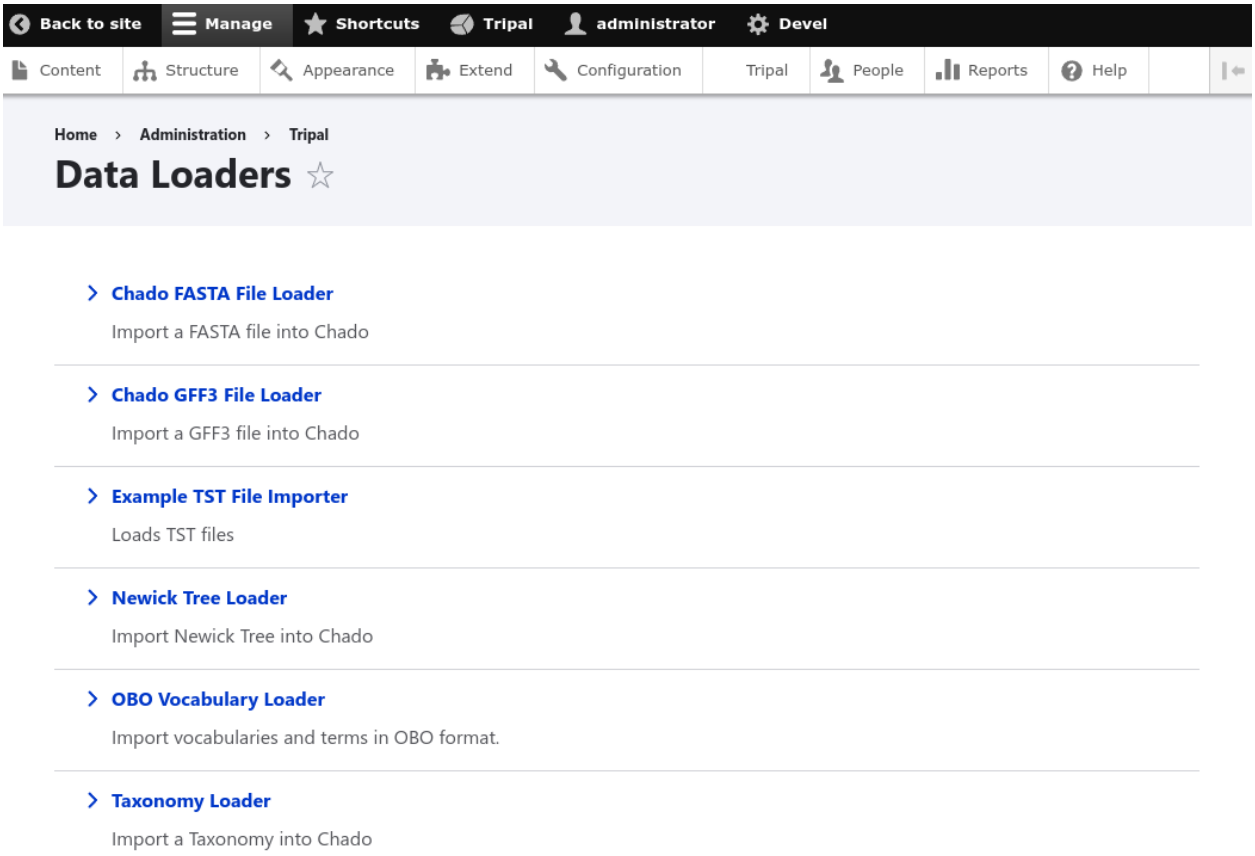
For our `ExampleImporter` class we will set the annotations accordingly:

```
/**
 *  TST Importer implementation of the ChadoImporterBase.
 *
 *  @TripalImporter(
 *    id = "tripal_tst_loader",
 *    label = @Translation("Example TST File Importer"),
 *    description = @Translation("Loads TST files"),
 *    file_types = {"txt", "tst", "csv"},
 *    upload_description = @Translation("TST is a fictional format.  Its a 2-column,␣
↪CSV file.  The columns should be of the form featurename, and text"),
 *    upload_title = @Translation("TST File"),
 *    use_analysis = True,
 *    require_analysis = True,
 *    use_button = True,
 *    button_text = @Translation("Import TST file"),
 *    file_upload = True,
 *    file_remote = True,
 *    file_local = True,
 *    file_required = True,
 *    submit_disabled = False
 *  )
 */
class ExampleImporter extends ChadoImporterBase {
```

**Warning:** You must use double quotes when specifying strings in the Annotations.

### Step 5: Check Availability

Now that we have created the plugin and set it's annotations it should appear in the list of Tripal Importers at **admin > Tripal > Data Loaders** after we clear the Drupal cache (`drush cr`).

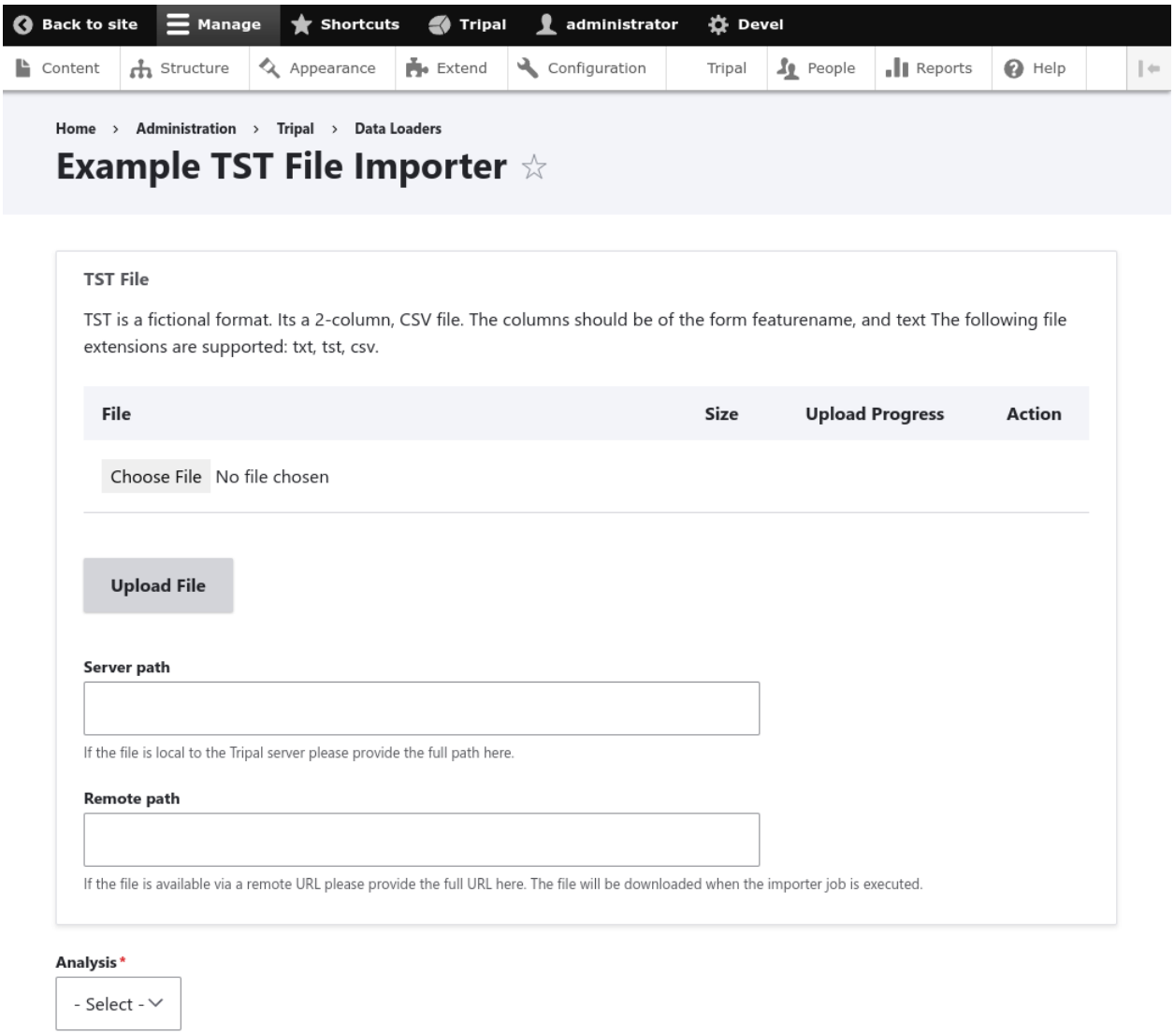



---

**Note:** If your importer does not show in the list of data loaders, check the Drupal recent logs at **admin > Manage > Reports > Recent log messages** .

---

Using the annotation settings we provided, the importer form will automatically provide a **File Upload** field set, and an **Analysis** selector. The **File Upload** section lets users choose to upload a file, provide a server path to a file already on the web server or a specify a remote path for files located via a downloadable link on the web. The **Analysis** selector is important because it allows the user to specify an analysis that describes how the data file was created. It will look like the following screenshot:




### Step 6: Customize the Form

Most likely, you will want to add elements the importer form. For our example TST file importer we want to split the file to retrieve feature and their properties, and then insert properties into the `featureprop` table of Chado. That table requires a controlled vocabulary term ID for the `type_id` column of the table. Therefore, we want to customize the importer form to request a controlled vocabulary term. To customize the form we can use three functions:

- `form()`: Allows you to add additional form elements to the form.
- `formValidate()`: Provides a mechanism by which you can validate the form elements you added.
- `formSubmit()`: Allows you to perform some preprocessing prior to submission of the form. Typically this function does not need to be implemented–only if you want to do preprocessing before submission.

---

**Note:** If you are not familiar with form creation in Drupal you may want to find a Drupal reference book that provides step-by-step instructions. Additionally, you can explore the API documentation for form construction for Drupal 10.

---

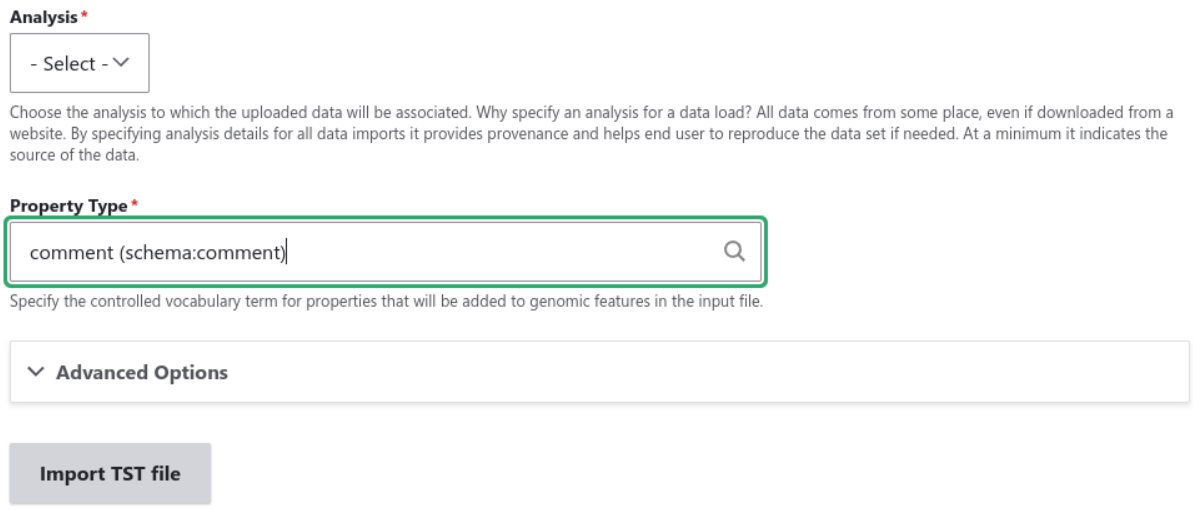
### The form() function

We can use the `form()` function to add the element to request the property CV term. To help, Tripal provides a handy service for searching for a controlled vocabulary term, we can use this as part of a text box with an autocomplete. The following code shows the addition of a new `textfield` form element with a `#autocomplete_route_name` setting that tells the form to use Tripal's CV search service to support autocompletion as the user types.

```
public function form($form, &$form_state) {

  // Always call the parent form.
  $form = parent::form($form, $form_state);

  // Add an element to the form to allow a user to pick
  // a controlled vocabulary term.
  $form['pick_cvterm'] = [
    '#title' => t('Property Type'),
    '#type' => 'textfield',
    '#required' => TRUE,
    '#description' => t("Specify the controlled vocabulary term for "
      . "properties that will be added to genomic features in the input file."),
    '#autocomplete_route_name' => 'tripal_chado.cvterm_autocomplete',
    '#autocomplete_route_parameters' => ['count' => 5, 'cv_id' => 0],
  ];

  return $form;
}
```

The `#autocomplete_route_parameters` setting takes an array of two arguments: `count` and `cv_id`. The `count` argument specifies the maximum number of matching CV terms that will be shown as the user types. The `cv_id` in the example is set to zero, indicating that there are no restrictions on which vocabulary the terms can come from. If you wanted to restrict the user to only selecting terms from a specific vocabulary then you would set the `cv_id` to the vocabulary ID from Chado.

Reloading the importer, the form now has an autocomplete text box for selecting a CV term.



### The formValidate() function

The `formValidate()` function is responsible for verifying that the user supplied values are valid. This function receives two arguments: `$form` and `$form_validate`. The `$form` object contains the fully built form. The `$form_validate` argument contains the object that represents the user submitted state of the form. To warn the user of inappropriate values, the `$form_state->setErrorByName()` function is used. It provides an error message, highlights in red the widget containing the bad value, and prevents the form from being submitted–allowing the user to make corrections. In our example code, we will check that the user selected a CV term from the `pick_cvterm` widget.

```
public function formValidate($form, &$form_state) {

  // Get the values submitted by the user.
  $form_state_values = $form_state->getValues();

  // The pick_cvterm element is required and Drupal will handle that
  // check for us, so we only need to make sure the user let the user selected
  // a term from the autocomplete with the accession in parentheses.
  $term = $form_state_values['pick_cvterm'];

  if (!preg_match('/\(.+?:.+?\)/', $term)) {
    $form_state->setErrorByName('pick_cvterm',
      t('Please choose a property type from the list that appears while typing. '
        . 'It must include the controlled vocabulary term accession'));
  }
}
```

The implementation above gets the `pick_cvterm` element from the `$form_state` object. The PHP `preg_match` function uses a regular expression to make sure the term selected by the user has the format provided by the autocomplete (e.g., *comment (rdfs:comment)*). It checks to make sure the term accession is present in parentheses. For your importer, use this function to check as many form elements as you add to the importer.

### The formSubmit() function

If you need to perform any steps prior to running the importer you can use the `formSubmit()` function. Suppose we wanted to add to our form the ability for a user to add new terms that do not already exist in the database. We would create the form elements in the `form()` function, make sure we have validation checks in the `formValidate()` and then we could insert the new term into the database prior to job submission in the `formSubmit()` function. Most likely you will not need to use this function. For most existing importers provided by Tripal this function is not used.

## Step 7: Write Importing Code

When an importer form is submitted and passes all validation checks, a job is automatically added to the *Tripal Jobs* system. The `TripalImporter` parent class does this for us! The *Tripal Jobs* system is meant to allow long-running jobs to execute behind-the-scenes on a regular time schedule. As jobs are added they are executed in order. Therefore, if a user submits a job using the importer's form then the *Tripal Jobs* system will automatically run the job the next time it is scheduled to run or it can be launched manually by the site administrator.

When the **Tripal Job** system executes an importer job it will call three different functions:

- `preRun()`: contains code to be executed prior to the the `run()` function.
- `run()`: contains the code that performs the import of the file.
- `postRun()`: contains code to be executed after executiong of the ``run()` function.

These functions were added to our class as "stubs" in Step 3 above and now we discuss each of these.

### The preRun() function

The `preRun()` function is called automatically by Tripal and should contain code that must be executed prior to running hte importer. This function provide any setup that is needed prior to importing the file. In the case of our example importer, we will not need to use the `preRun()` function so it will remain empty.

### The run() function

The `run()` function is called automatically when Tripal runs the importer. For our `ExampleImporter`, the run function should read and parse the input file and load the data into Chado. The first step, is to retrieve the user provided values from the form and the file details. The inline comments in the code below provide instructions for retrieving these details.

```
public function run() {

  // All values provided by the user in the Importer's form widgets are
  // made available to us here by the Class' arguments member variable.
  $arguments = $this->arguments['run_args'];

  // The path to the uploaded file is always made available using the
  // 'files' argument. The importer can support multiple files, therefore
  // this is an array of files, where each has a 'file_path' key specifying
  // where the file is located on the server.
  $file_path = $this->arguments['files'][0]['file_path'];

  // The analysis that the data being imported is associated with is always
  // provided as an argument.
  $analysis_id = $arguments['analysis_id'];

  // Convert the cvterm text provided by the user submitted form
  // to the actual cvterm ID from chado.
  $cvterm = $arguments['pick_cvterm'];
  $cv_autocomplete = new ChadoCVTermAutocompleteController();
  $cvterm_id = $cv_autocomplete->getCVtermId($cvterm);

  // Now that we have our file path, analysis_id and CV term we can load
  // the file.  We'll do so by creating a new function in our class
  // called "loadMyFile" and pass these arguments to it.
  $this->loadMyFile($analysis_id, $file_path, $cvterm_id);
}
```

In the example code above the `loadMyFile()` function is a function we add to our importer class that completes the loading of the file. We do not show the code of that function here, but it will be responsible for reading in the file provided by the `$file_path` variable and import the feature properties into Chado.

### Logging

During execution of our importer it is often useful to inform the user of progress, status and issues encountered. All **TripalImporter** plugins have several built-in objects and functions that support logging and reporting of progress. For logging, each importer has access to a **TripalLogger** accessible as `$this->logger` which uses the Drupal Logging API. There are several functions that you can use with the logger than can report errors, warnings, notices or debugging information. A quick list of these are:

- `$this->logger->emergency($message)`
- `$this->logger->alert($message)`
- `$this->logger->critical($message)`
- `$this->logger->error($message)`
- `$this->logger->warning($message)`
- `$this->logger->notice($message)`
- `$this->logger->info($message)`
- `$this->logger->debug($message)`

For each of the functions above, the `$message` argument should contain the text that is reported. The following is an example code from the GFF3 loader where logging is used to report progress of each step:

```
$this->logger->notice("Step  1 of 27: Caching GFF3 file...");
```

> **Warning:** Do not use `print` or `print_r` statements as a way to inform the user of warnings, errors or progress. These will not be logged and may interfere with functional testing.

### Throwing errors

The **TripalImporter** plugins can throw errors if needed. Tripal will catch the error, perform appropriate logging and recover gracefully. An example of throwing an error from the GFF3 loader:

```
throw new \Exception(t('Cannot find landmark feature type \'%landmark_type\'.',
    ['%landmark_type' => $this->default_landmark_type]));
```

After an error is caught by Tripal, all database changes will be rolled back and any changes made to the database during the process of running the importer will no longer exist.

### Reporting Progress

For progress reporting, **TripalImporter** plugins can utilize two different functions:

- `$this->setTotalItems()`: Indicates the total number of items (or steps) that must be processed for the importer to complete.
- `$this->setItemsHandled()`: Reports the total number of items that have been handled.

An **item** is an arbitrary term indicating some countable "units" that will be processed by our importer. This can be lines in a file, bytes in a file, or steps that an importer performs. To initialize the progress, first set the number of items handled to zero:

```
$this->setItemsHandled(0);
```

Next indicate how many items you plan to process:

```
$this->setTotalItems($num_items);
```

Then, as you process each item in the loader you can rerun the `setItemsHandled()` function and update the number of items that have been handled.

### Step 8: Write Functional Tests

Functional testing is a critically important component of any software project. You should always strive to write tests for your software to ensure that the software performs as expected and bugs are less likely to enter the code. Drupal provides automated testing by integrated the `phpunit` testing framework. We can create functional tests by utilizing this infrastructure provided by Drupal.

### Create the Testing Class

Tests for **TripalImporters** are Class files that should be placed in the `tests/src/Kernel/Plugin/TripalImporter` folder of your module and the file should have the same name as the importer class with a "Test" suffix. For our example importer it would be found in this location `tripal_example_importer\tests\src\Kernel\Plugin\TripalImporter\ExampleImporterTest.inc`.

The testing class should have the same name as the file and should extend the `ChadoTestKernelBase`. The following example code shows the empty starter class for our `ExampleImporterTest` class:

```
namespace Drupal\Tests\tripal_chado\Kernel\Plugin\TripalImporter;

use Drupal\Core\Url;
use Drupal\Tests\tripal_chado\Kernel\ChadoTestKernelBase;

/**
* Tests the functionality of the ExampleImporter Importer.
*
*/
class ExampleImporterTest extends ChadoTestKernelBase {


  /**
   * {@inheritdoc}
   */
  protected function setUp(): void {
    parent::setUp();
  }

  /**
   * Tests the importer form.
   */
  public function testImporterForm() {

  }
```

(continues on next page)

(continued from previous page)

```
  /**
   * Tests the run function.
   */
  public function testRun() {
  }
}
```

During automated testing, a new temporary instance of Drupal is created. Drupal will run each test class and for each test class, the `setUp()` function will be run first. Afterwards, all of the functions that have the prefix `test` will be executed.

### The setUp() function

The `setup()` function allows you to prepare the test Drupal site so that it is ready for the tests that follow. The following code provides a common set of steps that all **TripalImporters** should use to setup the class for testing:

```
/**
 * {@inheritdoc}
 */
protected function setUp(): void {

  // Always call the parent::setUp function.
  parent::setUp();

  // Ensure we see all logging in tests.
  \Drupal::state()->set('is_a_test_environment', TRUE);

              // Open a connection to Chado. This will ensure we have a properly
  // prepared Chado database.
              $this->connection = $this->getTestSchema(ChadoTestKernelBase::PREPARE_TEST_
↪CHADO);

  // Ensure we can access file_managed related functionality from Drupal.
  // ... users need access to system.action config?
  $this->installConfig('system');
  // ... managed files are associated with a user.
  $this->installEntitySchema('user');
  // ... Finally the file module + tables itself.
  $this->installEntitySchema('file');
  $this->installSchema('file', ['file_usage']);
}
```

In the `setUp()` function above, we first allow the parent class to perform its setup functions. Then we ensure the logger messages are set to be captured for testing, Chado is property initialized and the test supports managed files.

### The testImporterForm() function

For our first, test, we will write code to ensure that the user interface form is working as intended. Here we will test that all of the form elements that out importer added to the form are working as expected. You do not need to write tests for the file uploader nor for the analysis selection element. The testing of these elements will happen by the parent classes. Here we only need to test our additions to the form.

To perform tests, the `ChadoTestKernelBase` class automatically provides a set of PHPUnit assertion functions. For example, to test a boolean variable is `True` you would use the assertTrue() function in the following way:

```
$this->assertTrue($my_boolean_var, "The variable is not a boolean.");
```

Note that the assertion functions are available in your test class a member function (via `$this->assertTrue()`). All assertion functions are available in this way. For our example importer we only need to test the property cvterm element. T

For our example importer, the following code shows how we can test the form:

```
/**
 * Tests the importer form.
 */
public function testImporterForm() {

  // Store the plugin ID and label for easy access later.
              $plugin_id = 'tripal_tst_loader';
  $importer_label = 'Loads TST files';

  // Build the form using the Drupal form builder.
  $form = \Drupal::formBuilder()->getForm('Drupal\tripal\Form\TripalImporterForm',
↪$plugin_id);

  // Ensure we are able to build the form.
  $this->assertIsArray($form,
    "We expect the form builder to return a form but it did not.");
  $this->assertEquals('tripal_admin_form_tripalimporter', $form['#form_id'],
    "We did not get the form id we expected.");

  // Now that we have provided a plugin_id, we expect it to have a title matching our␣
↪importer label.
  $this->assertArrayHasKey('#title', $form,
    "The form should have a title set.");
  $this->assertEquals($importer_label, $form['#title'],
    "The title should match the label annotated for our plugin.");

  // The plugin_id stored in a value form element.
  $this->assertArrayHasKey('importer_plugin_id', $form,
    "The form should have an element to save the plugin_id.");
  $this->assertEquals($plugin_id, $form['importer_plugin_id']['#value'],
    "The importer_plugin_id[#value] should be set to our plugin_id.");

  // The form has a submit button.
  $this->assertArrayHasKey('button', $form,
    "The form should have a submit button since we indicated a specific importer.");
```

(continues on next page)

(continued from previous page)

```
  // Make sure the importer requires an analysis.
  $this->assertArrayHasKey('analysis_id', $form,
    "The from should not include analysis element, yet one exists.");

  // We should also have our importer-specific form elements added to the form!
  $this->assertArrayHasKey('pick_cvterm', $form,
    "The form should include an the 'pick_cvterm' form element.");

     }
```

**Error:** The documentation about testing is not yet complete. More information is needed to describe how to test form submission, the `run()`, `preRun()` and `postRun()`

## 5.4 Custom Module Development

Tripal is an extension of the Drupal Content Management System. Drupal 10 is written in PHP, is very object oriented and uses technology standards such as Composer, Symphony, YAML and Twig. This documentation is meant to act as an orientation to Tripal development in Drupal 10 and will link to additional Drupal-specific tutorials for specific topics.

### 5.4.1 Module File Structure

All Tripal extension modules require a module folder and a `.info.yml` file. With just these two items the module will display in Drupal 10's Extend administration page or can be activated directly with drush.

#### Choosing a module name

Choose a module machine name that is descriptive, short and unique. It is always a good idea to check out the Tripal Extensions module list to ensure you module name has not already been used. You module machine name must also meet the following rules:

- It must start with a letter.
- It must contain only lower-case letters and underscores.
- It must not contain any spaces.
- It must be unique. Your module should not have the same short name as any other module, theme, or installation profile you will be using on the site.
- It should not be any of the reserved terms : src, lib, vendor, assets, css, files, images, js, misc, templates, includes, fixtures, drupal.

It is also a good idea to ensure your module name encompasses the full functionality you would like to develop. For example, while your current goal may be importing a specific file format, you are likely to want to develop customized display through Tripal fields in the future. As such, you would want to stay away from `my_file_format_importer` and go with something more general like `my_data_type`. We also recommend you prefix your module name with a short identifier for you lab. This will ensure your module name is unique.

### Prepare a module skeleton

Start by creating a folder for your module in the modules directory of your Tripal site. This folder should use the machine name you choose above and includes all the files describing the functionality of your module. Read more regarding this containing folder

Next we let Drupal know about our module by describing it in an `.info.yml` file. The structure of this file is quite simple but descriptive:

```
name: Hello World Module
description: Creates a page showing "Hello World".
package: Custom

type: module
core: 10.x

dependencies:
 - tripal
```

Not only does this file let Drupal know about your module so you can enable it in your Tripal site but it also provides information to the site administrator. There are additional keys for this file; the ones above are the most common. Read more on the Drupal info.yml file

Once you have added an `.info.yml` file, you can navigate to your Drupal site in the browser, go to "Extend" in the administration toolbar at the top. Your module will now appear in this list and checking the checkbox enables it! If this doesn't happen, read some great debugging tips here.

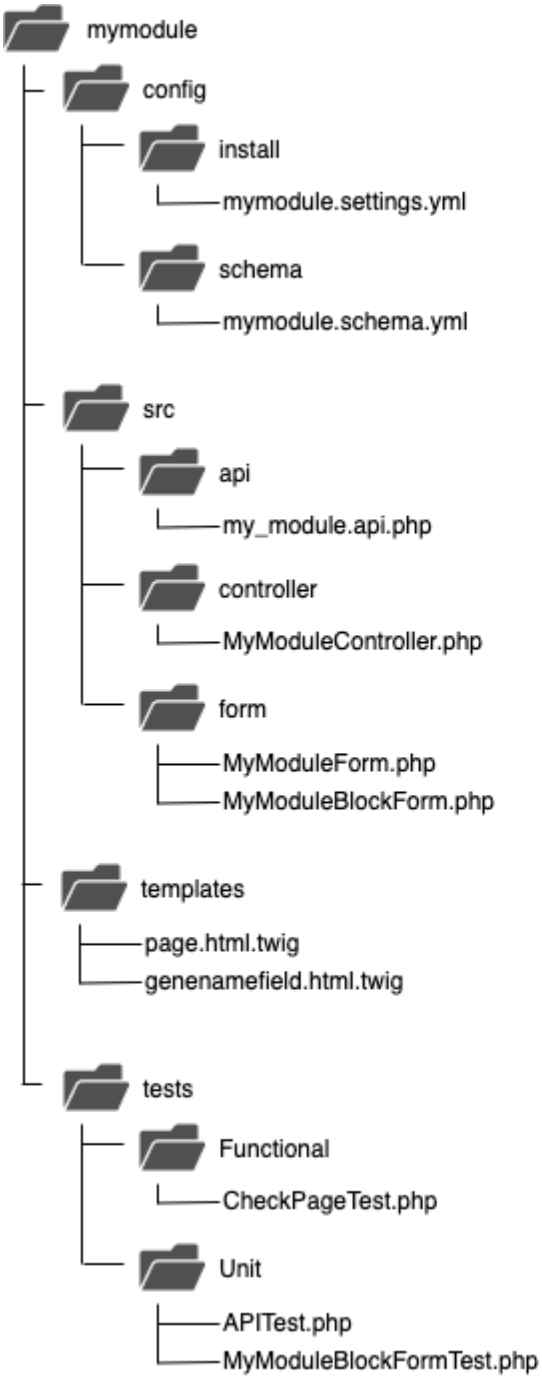
### Directory Structure

This section will explain the typical directory structure of a Tripal 4 extension module. These directories follow Drupal standards and the structure is often necessary for your classes to be automatically discovered.

As mentioned when preparing a module skeleton above, your entire module will be contained within a directory named using the machine name of your module. Within that base directory are the following:

- `config`: contains files defining default configuration including variables and schema.
- `src`: contains the bulk of your module including controllers, forms, fields and blocks.
- `templates`: contains your Twig template files for modifying display of fields and pages.
- `tests`: contains your automated phpunit tests.

The sub-directories and files within are described in the following image:




**Purpose:**

*Define default values for your configuration values.*

*Define schema for configuration types.*

*Contains re-usable functions or classes for use by developers extending your module.*

*Contains methods which define the action to take when a given route is invoked.*

*Each file defines a form for user input.*

*Style a given page or field.*

*Functional tests can be used to confirm text appears on a specific page or that buttons redirect as expected.*

*Unit tests are used to confirm a single function or method works as expected.*

### 5.4.2 Menus, Links and URLs

Defining URL paths and the programmatic flow to a rendered webpage is known as **Routing**. In Drupal 8, routing is handled by the Symfony Routing component which replaces `hook_menu` in Drupal 7. In your custom module, you will define all static routes in the `.routing.yml` file using the YAML format. The following is an example of a route:

```
hello_world.hello:
   path: '/hello/{{name}}'
   defaults:
     _controller:
 '\Drupal\hello_world\Controller\HelloWorldController::helloWorld'
     _title: 'Our first route'
   requirements:
     _permission: 'access content'
```

This route defines that a user navigating to `https://yourdrupalsite/hello` will render the content defined by the `helloWorld` method in the `HelloWorldController` class. The `defaults` key provides parameters to the handlers responsible for returning your content to the user. In this case, that includes the title of the page and where to get the content from. Finally, the `requirements` key defines conditions which must be met for the content to display; for example, permissions that the current user must have.

The other key thing to note about the above route is the `{{name}}` placeholder used in the path. By surrounding a variable name in curly brackets you can pass parameters to your controller. The important thing to note is that the name of your path variable must match exactly and exist in your specified controller method (i.e. `HelloWorldController::helloWorld($name)`).

**Additional Resources:**

- Official Drupal Routing documentation. Tripal uses the default Drupal routing system with no modifications.
- Official Drupal Converting hook_menu to Drupal 8 Routing
- BeFused Tutorial: "Introduction to creating menu links in a custom Drupal 8 module"
- Appnovation Tutorial: "Drupal 8 Routing: Decoupling hook_menu"

#### Menu Items

The menu system has an extensive user interface (UI) for defining menus and the links within them. This is great for management for your Tripal site as it allows you to dynamically add menu items requested by your community. However, when developing custom modules, you will also want to define these menu items programmatically to save time and provide navigation to other sites using your module. To do this you will want to use the `.links.menu.yml` file which lives in the base directory of your module. It looks like this:

```
hello_world.hello:
  title: 'Hello'
  description: 'Get your dynamic salutation.'
  route_name: hello_world.hello
  menu_name: main
  weight: 0
```

This defines a typically menu link; in this case, a link labelled "Hello" will appear at the top level of the main navigational menu. The machine name of the menu can be found in the path when adding links through the UI. In addition to the basic menu link demonstrated above there are also:

- local tasks: tabs at the top of the page linking to different sub-pages.

- local action: link at the top of a page (i.e. "+ Add Content" on `admin/content`) which allows the admin to complete an action.
- contextual links: similar to tabs but appear near the title (e.g. view, edit on a content page). These are different from tabs because they are dynamic and often require a parameters (e.g. entity id).

For Tripal Extension modules, it is good practice to have the main configuration page for your module available on Tripal's own Extensions menu page. To accomplish this, the main configuration page for your module should have an entry with tripal.extension as the parent. For example:

```
hello_world.config:
  route_name: hello_world.config
  title: 'Hello World Config'
  description: 'Configure how to greet the world'
  parent: tripal.extension
  weight: 0
```

Check out the additional resources for how to define these other types of menu items and for more information in general.

**Additional Resources**

- Official Drupal 8 Menu API docs
- What is a menu?
- Official Drupal docs: Add a menu link
- BeFused Tutorial: Introduction to creating menu links in a custom Drupal 8 module

### Links

When programmatically creating page content, you will often want to add links. To add internal links, use the route name as shown below. This ensures that your link doesn't break if the route is changed.

```
use Drupal\Core\Link;
$link = Link::createFromRoute('This is a link', 'entity.node.canonical', ['node' => 1]);
```

To generate a link to an external resource, you can use the following:

```
$link = Link::fromTextAndUrl('This is a link',
  Url::fromUri('http://www.google.com'));
```

For all the different ways to generate URLs see the following resources -the tutorial is particularly complete.

**Additional Resources**

- Agaric Tutorial: Creating Links in Code for Drupal 8
- Official Drupal docs: How to upgrade links from Drupal 7

### 5.4.3 Pages and Page Types

In Drupal terminology, each Tripal content page is an content entity (e.g. MyFavGene-1) and each Tripal Content Type (e.g. Genes) is a configuration entity. Tripal core has already done the work of defining these to Drupal and creates a number of common content types on install.

> A content entity (or more commonly, entity) is an item of content data, which can consist of text, HTML markup, images, attached files, and other data that is intended to be displayed to site visitors. Content entities can be defined by the core software or by modules.
>
> Content entities are grouped into entity types, which have different purposes and are displayed in very different ways on the site. Most entity types are also divided into entity sub-types, which are divisions within an entity type to allow for smaller variations in how the entities are used and displayed.
>
> *Excerpt from* Official Drupal Docs: What are Content Entities and Fields

This architecture allows you to categorize your data by type (e.g. gene versus germplasm variety) and provide specialized displays specific to each type.

Both Tripal content and Tripal content types can be created through the administrative user interface or programmatically. Tripal content entities and entity types have extended Drupal's default content entities to provide functionality specific to biological data. As such we recommend you create custom Tripal Content types rather then using the Drupal API directly.

**Additional Resources:**

- Official Drupal Docs: What are Content Entities and Fields
- Official Drupal Docs: Introduction to Entity API in Drupal 8
- Official Drupal Docs: Entity Types
- Unleashed Technologies: Drupal Entities - Part 1: What are they?
- Drupalize Me: Entity API Overview

### 5.4.4 Fields (content building blocks)

Fields are the building blocks of content in Drupal. For example, all content types (e.g. "Article", or "Basic Page") provide content to the end-user via fields that are bundled with them. For example, when adding a basic page (a default Drupal content type), the end-user is provided with form elements (or widgets) that allow the user to set the title and the body text for the page. The "Body" is a field. When a basic page is viewed, the body is rendered on the page using formatters and Drupal stores the values for the body in the database. Every field, therefore, provides three types of functionality: instructions for storage, widgets for allowing input, and formatters for rendering.

Drupal provides a variety of built-in fields, and extension module developers have created a multitude of new fields that can be added by the site admin to add new functionality and support new types of data. Tripal follows this model, but adds a variety of new object oriented classes to support storage of data in biological databases such as Chado.

---

**Note:** Not every custom module will require fields. But if you need a new way to store and retrieve data, or if you need data to appear on an existing Tripal content type then you will want to create a new field for your custom module.

---

### Field Classes

Anyone who wants to implement a new field in Drupal must implement three different classes:

- FieldItemBase: the class that defines a new field. This class interacts directly with the data storage plugin to load and save the data managed by this field.
- WidgetBase: the class that defines the form elements (widgets) provided to the end-user to supply or change the data managed by this field.
- FormatterBase: the class that defines how the field is rendered on the page.

These classes were extended by Tripal to provide additional functionality that allows Tripal-based fields to communicate with additional data stores housing biological data. There is support for a number of types of datastores (e.g. MySQL, PostgreSQL, SQLite) in core Drupal but you are required to choose a single data store for your site. The extension to support multiple data stores provided by Tripal allows you to keep your biological data separate from the website and still available to scientific analysis and visualization tools.

Chado is the default data store implemented within Tripal as it offers flexible support for a wide breadth of biological data types, ontology-focused metadata, and robust data integrity. The following documentation will demonstrate how to develop custom fields with data stored in Chado. However, the Tripal data storage plugin and Tripal Fields are designed to work with additional data stores and documentation showing how to take advantage of this will be written in the future.

Custom module developers who wish to add new fields to Tripal whose data are stored in Chado should implement the following three classes for every new field:

- **ChadoFieldItemBase**: extends the Tripal class *TripalFieldItemBase* which extends the Drupal class *FiedlItemBase*. The *TripalFieldItemBase* must be used for all fields attached to Tripal content types and the *ChadoFieldItemBase* adds Chado-specific support.
- **TripalWidgetBase**: a class that extends the Drupal class *WidgetBase*.
- **TripalFormatterBase**: a class that extends the Drupal class *FormatterBase*.

### How to Write a New Field for Chado

### Directory Setup

Drupal manages fields using its Plugin API. this means that as long as new field classes are placed in the correct directory and have the correct "annotations" in the class comments then Drupal will find them and make the field available. All new fields must be placed in the custom extension module inside of the *src/Plugin/Field* directory. There are three subdirectories, one each for the three elements of a field: *FieldType*, *FieldWidget*, *FieldFormatter*. For a new field named *MyField* the directory structure would look like the following:

```
mymodule
├── config
├── src
│   └── Plugin
│       └── Field
│           ├── FieldFormatter
│           │   └── MyFieldFormatter.php
│           ├── FieldType
│           │   └── MyFieldType.php
│           └── FieldWidget
│               └── MyFieldWidget.php
│
```

(continues on next page)

(continued from previous page)

```
├── tests
└── templates
```

Note that the file name must match the class name.

### Naming convention

The filename for your new field should adhere to the following schema. Please note the casing used. In addition, for fields that will be included in Tripal Core, note the 'Default' designation, any fields added by extension modules should **not** use 'Default':

Table 2: Tripal Core modules:

| File | Filename |
|---|---|
| Type | MyFieldTypeDefault.php |
| Formatter | MyFieldFormatterDefault.php |
| Widget | MyFieldWidgetDefault.php |

Table 3: Extension modules:

| File | Filename |
|---|---|
| Type | MyFieldType.php |
| Formatter | MyFieldFormatter.php |
| Widget | MyFieldWidget.php |

Within the individual files, in the annotation section, the ID also has to follow a specific format, and would look like the following:

Table 4: Tripal Core modules:

| File | ID within annotation |
|---|---|
| Type | my_field_type_default |
| Formatter | my_field_formatter_default |
| Widget | my_field_widget_default |

Table 5: Extension modules:

| File | ID within annotation |
|---|---|
| Type | my_field_type |
| Formatter | my_field_formatter |
| Widget | my_field_widget |

### About the Storage Backend

### Default Drupal Behavior

By default, all built-in fields provided by Drupal store their data in the Drupal database. This is provided by Drupal's SqlContentEntityStorage storage plugin. This storage plugin will create a database table for every field. For example, if you explore the Drupal database tables you will see the following for the body field attached to the node content type:

```
                              Table "public.node__body"
    Column    |          Type          | Collation | Nullable |        Default
--------------+------------------------+-----------+----------+-----------------------
 bundle       | character varying(128) |           | not null | ''::character varying
 deleted      | smallint               |           | not null | 0
 entity_id    | bigint                 |           | not null |
 revision_id  | bigint                 |           | not null |
 langcode     | character varying(32)  |           | not null | ''::character varying
 delta        | bigint                 |           | not null |
 body_value   | text                   |           | not null |
 body_summary | text                   |           |          |
 body_format  | character varying(255) |           |          |
Indexes:
    "node__body____pkey" PRIMARY KEY, btree (entity_id, deleted, delta, langcode)
    "node__body__body_format__idx" btree (body_format)
    "node__body__bundle__idx" btree (bundle)
    "node__body__revision_id__idx" btree (revision_id)
Check constraints:
    "node__body_delta_check" CHECK (delta >= 0)
    "node__body_entity_id_check" CHECK (entity_id >= 0)
    "node__body_revision_id_check" CHECK (revision_id >= 0)
```

The values provided by the user for the body of a node type are housed in this table. The following describes the columns of the table.

These columns are present for all fields

- *bundle*: the machine name of the content type (e.g. node)
- *deleted*: a value of 1 indicates the field is marked for deletion
- *entity_id*: the unique ID of the node that this field belongs to.
- *revision_id*: the node revision ID.
- *langcode*: for fields that are translatable, this indicates the language of the saved value.
- *delta*: for fields that support multiple values, this is the index (starting at zero) for the order of the values.

These columns are specific to the field:

- *body_value*: stores the value for the body
- *body_summary*: stores the body summary
- *body_format*: instructions for how the body should be rendered (e.g. plain text, HTML, etc.)

### Support for Chado

For fields storing biological data in something other than Drupal tables, Tripal provides its own plugin named *TripalStorage*. If a custom module wants to store data in a data backend other than in Drupal tables, it must create an implementation of this plugin. By default, Tripal provides the *ChadoStorage* implementation that allows a field to interact with a Chado database.

The *ChadoStorage* backend extends the *SqlContentEntityStorage* and will create a table in the Drupal schema for every Tripal field that is added to a content type. The table columns will have the same default columns. It will also have a set of additional columns for every property the field wants to manage.

The *ChadoStorage* backend is different from the *SqlContentEntityStorage* in that it will not store the values of the properties in the table. This is because those values need to be stored in Chado–we do not want to duplicate the data in the Drupal schema and the Chado schema. The *ChadoStorage* backend is also different in that it requires a set of property settings that help it control how properties of a field are stored, edited and loaded from Chado. Instructions for working with properties and storing data in Chado are described in the following sections.

---

**Note:** The *ChadoStorage* backend will not store biological data in the Drupal tables–only in the Chado tables. The only exceptions are record IDs that associate the field with data in Chado.

---

### Implementing a ChadoFieldItemBase Class

When creating a new Tripal field, the first class that must be created is the "type" class. This must extend the *ChadoFieldItemBase* class.

### Single-Value Fields

A single-value field is the simplest Chado field. This is a field that manages a data value from a single column in a single Chado table. For example, the *genus* column of the *organism* table of Chado stores the genus of an organism. For the organism pages provided by Tripal, a single-value field is used to provide the genus.

Tripal provides some ready-to-use field classes for single-values. These are:

- **ChadoIntegerTypeItem**: for integer data.
- **ChadoStringTypeItem**: for string data with a max length.
- **ChadoTextTypeItem**: for string data with unlimited length.
- **ChadoRealTypeItem**: for real (floating point) numberic data.
- **ChadoBoolTypeItem**: for boolean data.
- **ChadoDateTimeTypeItem**: for data/time data.

**Warning:** The alpha v1 version of Tripal v4 does not yet implement these fields: *ChadoRealTypeItem*, *ChadoBoolTypeItem*, *ChadoDateTimeTypeItem*

If you need to add a single-value field for your custom module then you do not need to write your own field! You can use one of these existing field types. See the section *Automate Adding a Field to a Content Type* for instructions to add the field during installation of your module.

### Complex Fields

A complex field is one that manages multiple properties (or multiple values) within a single field. An example of a complex field is one that stores/loads the organism of a germplasm content type. Within Chado, a record in the *stock* table is used to store germplasm data. The *stock* table has a foreign key constraint with the *organism* table. Therefore, a germplasm page must provide a field that allows the user to specify an organism for saving. It should also format the organism name for display.

In practice, the *stock* table stores the numeric *organism_id* when saving a germplasm. We could use a single-value *ChadoIntegerTypeItem* to allow the user to provide the numeric ID for the organism. But, this is not practical. Users should not be required to use a look-up table of numeric organism IDs.

Instead what we need is:

- A field that will store and load a numeric organism ID value that the user will never see.
- A field that has access to the genus, species, infraspecific type, infraspecific name, etc., of the organism.
- A widget (form element) that allows the user to select an existing organism.
- A formatter that prints the full scientific name of the organism.

### Class Setup

To create a new field, we will extend the *ChadoFieldItemBase*. For a new field named *MyField* we would create a new file in our module here: *src/Plugin/Field/FieldType/MyfieldType.php*. The following is an empty class example:

```
<?php

namespace Drupal\mymodule\Plugin\Field\FieldType;

use Drupal\tripal_chado\TripalField\ChadoFieldItemBase;
use Drupal\tripal_chado\TripalStorage\ChadoVarCharStoragePropertyType;
use Drupal\tripal_chado\TripalStorage\ChadoIntStoragePropertyType;
use Drupal\tripal_chado\TripalStorage\ChadoTextStoragePropertyType;
use Drupal\tripal\TripalStorage\StoragePropertyValue;

/**
 * Plugin implementation of Tripal string field type.
 *
 * @FieldType(
 *   id = "MyField",
 *   label = @Translation("MyField Field"),
 *   description = @Translation("An example field"),
 *   default_widget = "MyFieldWidget",
 *   default_formatter = "MyFieldFormatter"
 * )
 */
class MyField extends ChadoFieldItemBase {

  public static $id = "MyField";

  /**
   * {@inheritdoc}
   */
```

(continues on next page)

(continued from previous page)

```
  public static function defaultFieldSettings() {
    $settings = [];
    return $settings + parent::defaultFieldSettings();
  }

  /**
   * {@inheritdoc}
   */
  public function fieldSettingsForm(array $form, FormStateInterface $form_state) {
    $elements = [];
    return $elements + parent::fieldSettingsForm($form, $form_state);
  }

  /**
   * {@inheritdoc}
   */
  public static function defaultStorageSettings() {
    $settings = parent::defaultStorageSettings();
    return $settings;
  }

  /**
   * {@inheritdoc}
   */
  public function storageSettingsForm(array &$form, FormStateInterface $form_state, $has_
↪data) {
    $elements = [];
    return $elements + parent::storageSettingsForm($form,$form_state,$has_data);
  }

  /**
   * {@inheritdoc}
   */
  public function getConstraints() {
    $constraints = parent::getConstraints();
    return $constraints;
  }

  /**
   * {@inheritdoc}
   */
  public static function tripalTypes($field_definition) {
    $entity_type_id = $field_definition->getTargetEntityTypeId();

    // Get the settings for this field.
    $settings = $field_definition->getSetting('storage_plugin_settings');
    $base_table = $settings['base_table'];

    // Determine the primary key of the base table.
    $chado = \Drupal::service('tripal_chado.database');
    $schema = $chado->schema();
    $base_schema_def = $schema->getTableDef($base_table, ['format' => 'Drupal']);
```

(continues on next page)

(continued from previous page)

```
    $base_pkey_col = $base_schema_def['primary key'];

    // Return the array of property types.
    return [
      new ChadoIntStoragePropertyType($entity_type_id, self::$id,'record_id', [
        'action' => 'store_id',
        'drupal_store' => TRUE,
        'chado_table' => $base_table,
        'chado_column' => $base_pkey_col
      ]),
    ];
  }
}
```

Below is a line-by-line explanation of each section of the code snippet above.

### Namespace and Use Statements

The following should always be present and specifies the namespace for this field.

```
namespace Drupal\mymodule\Plugin\Field\FieldType;
```

**Note:** Be sure to change *mymodule* in the *namespace* to the name of your module.

**Warning:** If you misspell the *namespace* your field will not work properly.

The following "use" statements are required for all Chado fields.

```
use Drupal\tripal_chado\TripalField\ChadoFieldItemBase;
use Drupal\tripal\TripalStorage\StoragePropertyValue;
```

The following "use" statements are for each type of property your class will support. See the *Property Types* section for a listing of property classes you could import if needed.

```
use Drupal\tripal_chado\TripalStorage\ChadoVarCharStoragePropertyType;
use Drupal\tripal_chado\TripalStorage\ChadoIntStoragePropertyType;
use Drupal\tripal_chado\TripalStorage\ChadoTextStoragePropertyType;
```

### Annotation Section

The annotation section in the class file is the in-line comments for the class. Note the @FieldType stanza in the comments. Drupal uses these annotations to recognize the new field. It provides information such as the field ID, label and description. It also indicates the default widget and formatter class. This annotation is required.

```
/**
 * Plugin implementation of Tripal string field type.
 *
```

(continues on next page)

(continued from previous page)

```
 * @FieldType(
 *   id = "MyField",
 *   label = @Translation("MyField Field"),
 *   description = @Translation("An example field"),
 *   default_widget = "MyFieldWidget",
 *   default_formatter = "MyFieldFormatter"
 * )
 */
```

**Warning:** If the annotation section is not present, has misspellings or is not complete, the field will not be recognized by Drupal.

### Class Definition

Next, the class definition line must extend the *ChadoFieldItemBase* class. You must name your class the same as the filename in which it is contained (minus the *.php* extension).

```
class MyField extends ChadoFieldItemBase {
```

**Warning:** If you misspell the class name such that it is not the same as the filename of the file in which it is contained, then the field will not be recognized by Drupal.

### The defaultFieldSettings() Function

This is an optional function. If your field requires some additional settings that must be set when the field is added to a content type you can set those here.

```
public static function defaultFieldSettings() {
  $settings = [];
  return $settings + parent::defaultFieldSettings();
}
```

This function will return an associative array of all settings your field supports. You are free to use whatever settings you want. However, all fields in Tripal must be mapped to a controlled vocabulary term. Therefore, Tripal will automatically add the following settings to every field:

- **termIdSpace**: the namespace of the controlled vocabulary of the term assigned to this field (e.g. GO for the Gene Ontology; SO for the Sequence Ontology).
- **termAccession**: the accession of the term assigned to this field.

These settings are automatically attached to the field when the *parent::defaultFieldSettings()* function is called.

As an example, the Tripal organism field sets the term ID space and accession:

```
public static function defaultFieldSettings() {
  $settings = parent::defaultFieldSettings();
  $settings['termIdSpace'] = 'OBI';
  $settings['termAccession'] = '0100026';
```

(continues on next page)

(continued from previous page)

```
  return $settings;
}
```

Not all fields will need the *termIdSpace* and *termAccession* hardcoded like in the example above. A field can be re-used for different terms and those can be set with the field is added automatically. See the *Automate Adding a Field to a Content Type* section.

### The defaultStorageSettings() Function

The field settings described in the previous function apply to the field. But some settings may be needed for the storage backend. Drupal distinguishes between field settings and field storage settings.

```
/**
 * {@inheritdoc}
 */
public static function defaultStorageSettings() {
  $settings = parent::defaultStorageSettings();
  $settings['storage_plugin_settings']['base_column'] = '';
  return $settings;
}
```

In the example above the first line calls `parent::defaultStorageSettings()`. this will retrieve the default settings for all Chado fields. This includes a setting named `base_table` in the `storage_plugin_settings` array. The `ChadoStorage` backend requires a `base_table` setting to tell it what table of Chado this field works with. Tripal will pass to the storage backend any settings in the `storage_plugin_settings` array. But you are free to add any additional settings you would like to help manage your field, especially if those settings help the field define how it will interact with Chado.

An example where a storage settings is needed is in the `ChadoStringTypeItem` field that gets used for any single-value string mapped to a Chado table column. Here we must set the maximum length of the string. Here is the corresonding `defaultStorageSettings` function from this field:

```
public static function defaultStorageSettings() {
  $settings = parent::defaultStorageSettings();
  $settings['max_length'] = 255;
  $settings['storage_plugin_settings']['base_table'] = '';
  $settings['storage_plugin_settings']['base_column'] = '';
  return $settings;
}
```

### The storageSettingsForm() Function

If a field needs input from the user to provide values for settings, then the *storageSettingsForm()* function can be implemented. Add the form elements needed for the user to provide values.

For example, the *ChadoStringTypeItem* field wants to allow the site admin to set the maximum string length.

```
public function storageSettingsForm(array &$form, FormStateInterface $form_state, $has_
↪data) {
  $elements = [];
  $elements['max_length'] = [
```

(continues on next page)

(continued from previous page)

```
    '#type' => 'number',
    '#title' => t('Maximum length'),
    '#default_value' => $this->getSetting('max_length'),
    '#required' => TRUE,
    '#description' => t('The maximum length of the field in characters.'),
    '#min' => 1,
    '#disabled' => $has_data,
  ];
  return $elements + parent::storageSettingsForm($form,$form_state,$has_data);
}
```

The site admin will be able to change the storage settings if they:

- Navigate to *Structure* > *Tripal Content Types*
- Choose the *Manage fields* option in the dropdown next to the Tripal content type.
- Choose the *Edit* option in the dropdown next to a field of type "Chado String Field Type"
- Clicking on the *Settings* tab.

**Warning:** The key of the *$elements* array must match the name of the setting. In the example code above, notice that "max_length" is used in the elements array and is the name of the setting.

**Note:** Site admins can change storage settings for a field only before it is used. Once the field is used to store data on a live entity, storage settings are fixed.

### The fieldSettingsForm() Function

The *fieldSettingsForm()* functions in the same was as the *storageSettingsForm()* function but for the field settings.

### The getConstraints() Function

The *getConstraints()* function is used to provide a set of constraints to ensure that values provided to fields are appropriate. You can read more about defining validation contraints for fields here.

For following code example, is from the *ChadoStringTypeItem* field. It wants to ensure that that max length of the string is not exceeded.

```
public function getConstraints() {
  $constraints = parent::getConstraints();
  if ($max_length = $this->getSetting('max_length')) {
    $constraint_manager = \Drupal::typedDataManager()->getValidationConstraintManager();
    $constraints[] = $constraint_manager->create('ComplexData', [
      'value' => [
        'Length' => [
          'max' => $max_length,
          'maxMessage' => t('%name: may not be longer than @max characters.', [
            '%name' => $this
```

(continues on next page)

(continued from previous page)

```
            ->getFieldDefinition()
            ->getLabel(),
            '@max' => $max_length,
          ]),
        ],
      ],
    ]);
  }
  return $constraints;
}
```

### The tripalTypes() Function

The *tripalTypes()* function is used to specify the property types that this field will manage. A field may house as many properties as it needs. For example, the organism field that may appear on a stock page needs to track the genus, species, infraspecific type, and infraspecific name for an organism. These can be tracked using properties. Each property is of a specific type such as string, text, integer, etc. This function is used to define the property types. A property type is actually an object, thus, this function returns an array of property type objects. See the *Property Types* section below for more information about these object classes.

In the code block below you can see the steps where the field settings are retrieved, and then used to create an array containing a single property. More about properties is described in the next section.

```
public static function tripalTypes($field_definition) {
  $entity_type_id = $field_definition->getTargetEntityTypeId();

  // Get the settings for this field.
  $settings = $field_definition->getSetting('storage_plugin_settings');
  $base_table = $settings['base_table'];

  // Determine the primary key of the base table.
  $chado = \Drupal::service('tripal_chado.database');
  $schema = $chado->schema();
  $base_schema_def = $schema->getTableDef($base_table, ['format' => 'Drupal']);
  $base_pkey_col = $base_schema_def['primary key'];

  // Return the array of property types.
  return [
    new ChadoIntStoragePropertyType($entity_type_id, self::$id,'record_id', [
      'action' => 'store_id',
      'drupal_store' => TRUE,
      'chado_table' => $base_table,
      'chado_column' => $base_pkey_col
    ]),
  ];
}
```

### Property Types

As was introduced in the *The tripalTypes() Function* section above, each field must define the set of properties that it will manage. The set of property types is returned by the *tripalTypes()* function.

Tripal provides a variety of property type classes that you will use to define these properties. These are named after PostgreSQL column types:

- **ChadoBoolStoragePropertyType**: a boolean property.
- **ChadoDateTimeStoragePropertyType**: a date/time property.
- **ChadoIntStoragePropertyType**: an integer property.
- **ChadoRealStoragePropertyType**: a floating point property.
- **ChadoTextStoragePropertyType**: an unlimited string property.
- **ChadoVarCharStoragePropertyType**: a string property with a maximum length.

All of these classes can be instantiated with four arguments:

- The entity type ID: the unique ID for the entity type.
- The field ID: the unique ID of the field this property belongs to.
- The property "key": a unique key for this property.
- The property settings: an array of settings for this property. See the *Property Settings* section below for more information on how to specify the property settings array.

### Property Settings

The *Property Types* section above indicated that each property type class has a fourth argument that provides settings for the property. These settings are critical for describing how the property is managed by the `ChadoStorage` backend. The settings are an associative array of key-value pairs that specify an "action" to perform for each property and corresponding helper information. The following actions can be used:

- **store_id**: indicates that the value of this property will hold the record ID (or primary key ID) of the record in the base table of Chado. Common base tables include: analysis, feature, stock, pub, organism. This action uses the following key/value pairs:
  - **chado_table**: (required) the name of the table that this property will get stored in. This will always be the base table name (e.g. feature).
  - **chado_column**: (required) the name of the column in the table where This property value will get stored. This will always be the primary key of the base table (e.g., feature_id).
- **store_link**: indicates that the value of this property will hold the value of a foreign key ID to the base table. A property with this action is required for fields that provide ancillary information about a record but that information is not stored in a column of the base table, but instead in a linked table. Examples for such a situation would be values from property table: e.g., analysisprop, featureprop, stockprop, etc. This action uses the following key/value pairs:
  - **chado_table**: (required) the name of the linked table (e.g. analysisprop)
  - **chado_column**: (required) the name of the foreign key column that links to the base table (e.g. analysis_id)
  - **drupal_store**: (requited) this setting should always be TRUE for this action. This forces Tripal to store this value in the Drupal field tables. Without this, Tripal cannot link the fields in Drupal with a base record.

- **store_pkey**: indicates that the value of this property will hold the primary key ID of a linked table. As with the `store_link` action, a property with this action is required for fields that provide ancillary information about a record but that information is not stored in a column of the base table, but instead in a linked table. Examples for such a situation would be values from property table: e.g., analysisprop, featureprop, stockprop, etc. This action uses the following key/value pairs:
  - **chado_table**: (required) the name of the linked table (e.g. analysisprop)
  - **chado_column**: (required) the name of the primary key column that links to the base table (e.g. analysisprop_id)
  - **drupal_store**: (requited) this setting should always be TRUE for this action. This forces Tripal to store this value in the Drupal field tables. Without this, Tripal cannot link the fields in Drupal with a base record.
- **store**: indicates that the value of this property should be stored in the Chado table. This action uses the following key/value pairs:
  - **chado_table**: (required) the name of the table that this property will get stored in.
  - **chado_column**: (required) the name of the column in the table where this property value will get stored.
  - **delete_if_empty**: (optional) if TRUE and this field is for ancillary data then the ancillary record should be removed if this value is empty.
  - **empty_value**: (optional) the value that indicates an empty state. This could be `0`, an empty string or NULL. Whichever is appropriate for the property. This value is used in conjunction with the **delete_if_empty** setting.
- **join**: indicates that the value of this property is obtained by joining the record ID in the property with the **store_id** action with another table in Chado.
  - **path**: (required) the sequence of joins that should be performed.
    - For example if the base table for the record is *feature* and we want to join on the *organism_id* to get the spcies then the path would be: *feature.organism_id>organism.organism_id*.
    - Separate multiple joins with a semicolon. For example to get the infraspecific name of an organism: *feature.organism_id>organism.organism_id;organism.type_id>cvterm.cvterm_id*.
  - **chado_column**: (required) the name of the column from the last join that will contain the value for this field.
  - **as**: (optional) to prevent a naming conflict in the SQL that the *ChadoStorage* backend will generate, you can rename the *chado_column* with a different name.
- **replace**: indicates that the value of this property is a tokenized string and should be replaced with values from other properties.
  - **template**: (required) a string containing the value of the field. The string should contain tokens that will be replaced by values of other properties. Tokens are surrounded by square brackets and contain the keys of other properties. For example. if the keys for other properties are "genus", "species", "iftype", "ifname" you can create a property that builds the full scientific name of an organism with the following template string: "[genus] [species] [iftype] [ifname]".
- **function**: indicates that the value of this property will be set by a callback function.
  - *Currently not implemented in Alpha release v1*

As an example, let's look at the `tripalTypes()` function of the field that allows an end-user to add an organism to content. This code is found in the `tripal_chado\src\Plugin\Field\FieldType\obi__organism.php` file of Tripal:

```
public static function tripalTypes($field_definition) {
  $entity_type_id = $field_definition->getTargetEntityTypeId();

  // Get the length of the database fields so we don't go over the size limit.
  $chado = \Drupal::service('tripal_chado.database');
  $schema = $chado->schema();
  $organism_def = $schema->getTableDef('organism', ['format' => 'Drupal']);
  $cvterm_def = $schema->getTableDef('cvterm', ['format' => 'Drupal']);
  $genus_len = $organism_def['fields']['genus']['size'];
  $species_len = $organism_def['fields']['species']['size'];
  $iftype_len = $cvterm_def['fields']['name']['size'];
  $ifname_len = $organism_def['fields']['infraspecific_name']['size'];
  $label_len = $genus_len + $species_len + $iftype_len + $ifname_len;

  // Get the base table columns needed for this field.
  $settings = $field_definition->getSetting('storage_plugin_settings');
  $base_table = $settings['base_table'];
  $base_schema_def = $schema->getTableDef($base_table, ['format' => 'Drupal']);
  $base_pkey_col = $base_schema_def['primary key'];
  $base_fk_col = array_keys($base_schema_def['foreign keys']['organism']['columns'])[0];

  // Return the properties for this field.
  return [
    new ChadoIntStoragePropertyType($entity_type_id, self::$id, 'record_id', [
      'action' => 'store_id',
      'drupal_store' => TRUE,
      'chado_table' => $base_table,
      'chado_column' => $base_pkey_col
    ]),
    new ChadoIntStoragePropertyType($entity_type_id, self::$id, 'organism_id', [
      'action' => 'store',
      'chado_table' => $base_table,
      'chado_column' => $base_fk_col,
    ]),
    new ChadoVarCharStoragePropertyType($entity_type_id, self::$id, 'label', $label_len,␣
↪[
      'action' => 'replace',
      'template' => "<i>[genus] [species]</i> [infraspecific_type] [infraspecific_name]",
    ]),
    new ChadoVarCharStoragePropertyType($entity_type_id, self::$id, 'genus', $genus_len,␣
↪[
      'action' => 'join',
      'path' => $base_table . '.organism_id>organism.organism_id',
      'chado_column' => 'genus'
    ]),
    new ChadoVarCharStoragePropertyType($entity_type_id, self::$id, 'species', $species_
↪len, [
      'action' => 'join',
      'path' => $base_table . '.organism_id>organism.organism_id',
      'chado_column' => 'species'
    ]),
    new ChadoVarCharStoragePropertyType($entity_type_id, self::$id, 'infraspecific_name',
↪ $ifname_len, [
```

(continues on next page)

(continued from previous page)

```
      'action' => 'join',
      'path' => $base_table . '.organism_id>organism.organism_id',
      'chado_column' => 'infraspecific_name',
    ]),
    new ChadoIntStoragePropertyType($entity_type_id, self::$id, 'infraspecific_type', [
      'action' => 'join',
      'path' => $base_table . '.organism_id>organism.organism_id;organism.type_id>cvterm.
↪cvterm_id',
      'chado_column' => 'name',
      'as' => 'infraspecific_type_name'
    ])
  ];
}
```

The Tripal organism property is used to associate an organism to a base record that has an `organism_id` column in the Chado table. We only need to store the `organism_id` to make this work, but again, requiring an end-user to enter a numeric organism is not ideal. Also we want our formatter to print a nicely formatted scientific name for the organism. We need more properties.

In the code above, we create seven properties for this field. As required we must have a property that uses the action `store_id` that will house the record ID (e.g., feature.feature_id). Because this field is supposed to store the `organism_id` for the feature, stock, etc., we have a property that uses the action `store` and maps to the `organism_id` column of the table.

We also have a variety of properties with a join action. These are used to join on the base table to get information such as the genus, species, and infraspecific type. Lastly, we have a property with the action `replace` that uses a tokenized string to create a nicely formatted scientific name for the organism.

### Implementing a TripalWidgetBase Class

**Warning:** This documentation is still being developed. In the meantime there are examples in the Tripal core codebase. Specifically, look in the *tripal_chado/src/Plugin/Field/FieldWidget* directory.

### Implementing a TripalFormatterBase Class

**Warning:** This documentation is still being developed. In the meantime there are examples in the Tripal core codebase. Specifically, look in the *tripal_chado/src/Plugin/Field/FieldFormatter* directory.

### Automate Adding a Field to a Content Type

**Warning:** This documentation is still being developed. In the meantime there are examples for programmatically adding TripalFields in the Tripal core codebase. Specifically, look in the Chado Preparer class in *tripal_chado/src/Task/ChadoPreparer.php*.

#### What About Fields not for Chado?

**Warning:** This documentation is still being developed. Currently ChadoStorage provides an example for implementing the TripalStorage data store extension. It can be found in *tripal_chado/src/Plugin/TripalStorage/ChadoStorage.php*.

### 5.4.5 Forms (user input)

In Drupal 8, all the parts of the form are contained within a single class. The `buildForm` method is where you define what your form will look like. This is done using the Form API which is very similar to what was used in Drupal 7. The `validateForm` method allows you to validate the user submitted data and give feedback to the user and the `submitForm` method allows you to act on the data submitted.

> Defining forms as structured arrays, instead of as straight HTML, has many advantages including:
>
> - Consistent HTML output for all forms.
> - Forms provided by one module can be easily altered by another without complex search and replace logic.
> - Complex form elements like file uploads and voting widgets can be encapsulated in reusable bundles that include both display and processing logic.
>
> *Excerpt from* Official Drupal Docs: Introduction to Form API

Tripal uses the Drupal Form API without any modification and your form classes will be saved, one per file, in the `src/forms` directory of your extension module.

**Additional Resources:**

- Official Drupal Docs: Introduction to Form API
- Official Drupal Docs: Form API
- Karim Boudjema: Create a custom form with Form API in Drupal 8
- Jaywant Topno: Step by step method to create a custom form in Drupal 8
- Official Drupal Docs: Upgrading forms from Drupal 7

### 5.4.6 Error Reporting and Logging

#### Tripal Logger

The Tripal logger service can be used to report status and errors to both the site user and to site administrators through the server log. A basic example is

```
$logger = \Drupal::service('tripal.logger');
$logger->notice('Hello world');
$logger->warning('Hello world');
$logger->error('Hello world');
```

There are eight levels of logging available. In order of increasing severity they are: *debug*, *info*, *notice*, *warning*, *error*, *critical*, *alert*, *emergency*

The log message can be a simple quoted PHP string, or a string that utilizes placeholders. In the latter case, pass in an associative array of placeholder keys and values into the second parameter. For example:

```
$options = [];
$logger->error('Error, status code @errornumber, error message @message', [
  '@errornumber' => $resultcode,
  '@message' => $errormessage,
], $options);
```

There are a few settings that can be passed in using the third `$options` parameter to control where the message is sent.

> Use `$options['drupal_set_message'] = TRUE;` if you want the message to appear on the user's screen, default is `FALSE`.
>
> Use `$options['logger'] = FALSE;` if you do NOT want the message to go to the log at */admin/reports/dblog*, default is `TRUE`.

The logger checks the `TRIPAL_SUPPRESS_ERRORS` environment variable. If it is defined with the value `true` (case insensitive), then all logging is suppressed even if it is not an "error" message. This is generally only used for automated testing to prevent output from being printed.

An additional pair of options is available for implementing progress bars. For the first message, set `$options['is_progress_bar'] = TRUE;` and `$options['first_progress_bar'] = TRUE;` For subsequent updates to the progress bar, only set `$options['is_progress_bar'] = TRUE;` It is likely for a progress bar that you will also want to include `$options['logger'] = FALSE;` to avoid overpopulating your server log.

### 5.4.7 Views (content listings)

Drupal Views is a module which allows administrators to create and/or customize existing content listings. In Drupal 8, the views module is part of core so you simply need to enable it to get this functionality. Developers can provide default views through extension modules which allows your content listings and searches to be both easily customizable and under version control.

**Additional Resources:**

- Subhojit Paul: How to create views programmatically in Drupal 8
- ComputerMinds: Render a Drupal 8 View programmatically (Render arrays FTW!)
- hook_views_data: how to integrate custom tables

### 5.4.8 Configuration Variables

Configuration variables can be done two ways in Drupal 8 depending upon whether you want your variables sync'd between production and development environments. Specifically,

- Simple configuration (Config API) — Settings that require synchronization between different environments, e.g. site name, slogan, user account settings, etc.
- Local configuration (State API) — Settings that are more transient or subject to change and which should not be synchronized between environments. e.g. the last cron run, the timestamp for statistics, last update time, etc.

**Additional Resources:**

- Official Drupal Docs: Config API
- Official Drupal Docs: State API
- X-Team: Configuration vs State
- Antistatique: Drupal 8 - Differences between Configuration API & State API

### 5.4.9 Theme (display)

Theming is the process of customizing the display of pages or fields. Drupal 8 theming is a two part process, as described below. Tripal uses the Drupal theming system without alteration.

1. Use `hook_theme` to tell Drupal about your custom template. This hook implementation should go in your `my_module.module` file.
2. Use the preprocess hook to use prepare your variables and do any processing needed.
3. Finally, you use Twig templates to format the HTML and insert the variables prepared in the preprocess hook. No processing should be done in these templates.

**Additional Resources:**

- Official Drupal Docs: Theming in your custom module
- Official Drupal Docs: Create custom twig templates for custom module
- Official Drupal Docs: Working With Twig Templates
- Official Drupal Docs: Defining a custom theme

### 5.4.10 Caching (performance)

**Additional Resources:**

- Official Drupal Docs: Cache concept
- Official Drupal Docs: Cache API
- Valuebound: A beginners guide to caching in Drupal 8
- Drupal 8: Quick Handbook on Cache API
- Acquia: Coding with Cache Tags in Drupal 8

### 5.4.11 Alternate Database Backends

Drupal is database agnostic; however, Tripal is still PostgreSQL leaning. Tripal 4+ are completely Chado agnostic though with all core functionality including Biological Vocabularies and Content Types standing independent of Chado. This allows for the development of alternate data backends with Chado already being implemented by the Tripal Chado core module. **This guide will describe how to integrate additional storage backends using Chado as an example.**

#### Tripal Vocabularies, IDSpaces and Terms

Tripal provides a Drupal Plugin for hooking into TripalVocab, TripalVocabSpace and TripalTerm classes. The TripalTermStorage plugin provides a number of methods mapping to preSave, postSave, load and delete functions for each entity type. This allows developers to implement this plugin through creation of a single class which can then handle full integration of all three classes with an additional data backend.

To create your own data backend for Vocabularies and Terms, you can follow the standard Drupal procedure for implementing plugins which will be detailed below.

### Step 1: Create your plugin implementation class

Your entire data backend will exist in a single class. This class implements the TripalTermStorageInterface and extends the TripalTermStorageBase. It also uses a number of other classes in order to pull them into the current scope. The following shows the Chado Integration class as an example. To create your own change the namespace to match your modules, the annotation to describe your data backend and the class name. This file should be created in your `src/Plugin/TripalTermStorage` directory in order to be discovered by Drupal/Tripal.

```
namespace Drupal\tripal_chado\Plugin\TripalTermStorage;

use Drupal\tripal\Entity\TripalVocab;
use Drupal\tripal\Entity\TripalVocabSpace;
use Drupal\tripal\Entity\TripalTerm;

use Drupal\tripal\Plugin\TripalTermStorage\TripalTermStorageBase;
use Drupal\tripal\Plugin\TripalTermStorage\TripalTermStorageInterface;
use Drupal\Core\Entity\EntityStorageInterface;

/**
 * TripalTerm Storage plugin: Chado Integration.
 *
 * @ingroup tripal_chado
 *
 * @TripalTermStorage(
 *   id = "chado",
 *   label = @Translation("GMOD Chado Integration"),
 *   description = @Translation("Ensures Tripal Vocabularies are linked with chado␣
↪cvterms."),
 * )
 */
class TripalTermStorageChado extends TripalTermStorageBase implements␣
↪TripalTermStorageInterface {

}
```

This plugin will work without any methods implemented although it obviously will not connect your data backend just yet. To test that your new implementation is registered properly with Drupal/Tripal you can use Drupal Console. Specifically you would use the debug:plugin Command command as shown here:

```
drupal debug:plugin tripal.termStorage
```

This will output a list of implementations for the Tripal Term Storage plugin and should include both your plugin implementation, as well as, the chado one.

#### Step 2: Implement the methods you need for integration.

For a list of available methods including documentation, check out the `tripal/src/Plugin/TripalTermStorage/TripalTermStorageInterface.php` file. There is an example for chado available in the `tripal_chado/src/Plugin/TripalTermStorage/TripalTermStorageChado.php`

## 5.5 Automated Testing

Tripal 4 is being developed with automated testing as it is upgraded. This greatly improves the stability of our software and our ability to fix any bugs. We highly recommend developing automated testing alongside any extension modules you create! This guide is intended to explain how automated testing is working for Tripal 4 and help you develop similar tests for your extensions.

### 5.5.1 How run automated tests locally

See the Drupal "Running PHPUnit tests" guide for instructions on running tests on your local environment. In order to ensure our Tripal functional testing is fully bootstrapped, tests should be run from Drupal core.

If you are using the docker distributed with this module, then you can run tests using:

```
docker exec --workdir=/var/www/drupal/web/modules/contrib/tripal tripal phpunit
```

### 5.5.2 Tripal-focused Testing

The following automated testing documentation and tutorials are focused on testing Tripal-specific functionality within Tripal Core and Extension modules. If there is a topic you would like covered that is not yet documented, please add an issue on our github at https://github.com/tripal/tripal_doc/issues!

#### Tripal Testing Environment

Tripal uses the Drupal testing environment to automate our tests using PHPUnit. This means that when you run tests on any Tripal site, the following will happen **FOR EACH TEST**:

1. Drupal will setup a virtual Drupal site whose level of functionality depends on whether you are running a Kernel or Functional Test. This will actually setup a full drupal database schema within your current site temporarily. Note: This is not a new postgresql schema but rather creates all the tables used by Tripal in the same public schema but with prefixes in the table names.

   - Kernel tests will give you a fully functional site that cannot be interacted with through the browser and that only has the specific modules, tables, etc. that you specify in your test *setUp()*
   - Functional tests will have a fully functional site with methods to explore the fully rendered pages within the test. It will still only have the modules you specify enabled but it will run the entire install for those modules whereas kernel tests do not.

2. Tripal does not yet do any additional preparing of the Drupal test environment so as not to include any more than you need for your tests. This means there are no content types, no fields, no TripalTerms, etc. However, we do provide a number of methods to complement the Drupal methods in helping you setup the environment exactly as you want to. These will be described in the next section.

3. The code inside your tests *setUp()* is now run to setup the environment for this test. The same setup will be run for all tests in the same test class.

4. Finally your test method is called. Note: any services, plugins, etc you use here will only have the test environment available. You will not have access to any data in your main site, nor should this long term affect your main site. **That said, we do not recommend running tests on production sites!**
5. Once your test is complete, the *tearDown()* method is called to clean the entire development environment up. This includes dropping the development drupal tables including any changes made by your test.

### Setting up Content Types

As mentioned above, there are no content types in the testing environment. That means to do any testing related to content types or fields, you will first need to setup content types in your environment. **Tripal has provided helper methods to make this easier! These are available in any test that extends the `TripalTestBrowserBase`, `TripalTestKernelBase`, `ChadoTestKernelBase`, and `ChadoTestBrowserBase`.**

For example, the following example creates the organism Tripal Term and then the organism content type:

```
/**
 * {@inheritdoc}
 */
protected function setUp(): void {
  parent::setUp();

  // Create the TripalTerm used for the organism content type.
  $this->createTripalTerm([
    'vocab_name' => 'obi',
    'id_space_name' => 'OBI',
    'term' => [
      'name' => 'organism',
      'definition' => '',
      'accession' =>'0100026',
    ]],
    'chado_id_space', 'chado_vocabulary'
  );

  // Create the Organism Content Type
  $this->createTripalContentType([
    'label' => 'Organism',
    'termIdSpace' => 'OBI',
    'termAccession' => '0100026',
    'category' => 'General',
    'id' => 'organism',
    'help_text' => 'A material entity that is an individual living system, ' .
      'such as animal, plant, bacteria or virus, that is capable of replicating ' .
      'or reproducing, growth and maintenance in the right environment. An ' .
      'organism may be unicellular or made up, like humans, of many billions ' .
      'of cells divided into specialized tissues and organs.',
  ]);
}
```

The above will work just fine on its own in a functional test as long as you have tripal listed in your *$modules* array. However, you will need to make additional parts available in kernel tests. Specifically, add the following install calls to the top of your setUp method:

```
/**
 * {@inheritdoc}
 */
protected function setUp(): void {
  parent::setUp();

  // Make the TripalTerm database tables available.
  $this->installSchema('tripal_terms');
  $this->installSchema('tripal_terms_vocabs');
  $this->installSchema('tripal_terms_idspaces');

  // Make the User, Tripal Content and Tripal Content Type entities available.
  $this->installEntitySchema('user');
  $this->installEntitySchema('tripal_entity');
  $this->installEntitySchema('tripal_entity_type');
```

**Warning:** The above content type will **NOT have any fields attached** to it yet! See the following section for adding fields.

### Adding Fields to Content Types

Now that we have a Tripal Content type, we will want to add fields to it. Again, Tripal provides an easy to use function to create field instances and attach them to your entity in the test environment!

The following code snippet shows how to add a single field to an existing content type in the test environment. This should go in your *setUp()* method after the content type is already created.

```
// Create the term used by the field.
// This is the term that would normally get set
// in the form when adding a field through the UI.
$genus_term = $this->createTripalTerm([
  'vocab_name' => 'taxonomic_rank',
  'id_space_name' => 'TAXRANK',
  'term' => [
    'name' => 'genus',
    'definition' => '',
    'accession' =>'0000005',
  ]],
  'chado_id_space', 'chado_vocabulary'
);
// Create the field instance.
$this->createTripalField(
  // The machine name of the content type to attach the field to.
  'organism',
  // The field settings + details.
  [
    // This can be anything.
    'field_name' => 'organism_taxrank_0000005',
    // This is the machine name of the field type you want to create.
    'field_type' => 'chado_string_type_default',
```

(continues on next page)

(continued from previous page)

```
    'term' => $genus_term,
    'is_required' => TRUE,
    'cardinality' => 1,
    // This indicates the base chado table and column to use for this field.
    // You would include anything here that you would normally supply on
    // the storage settings form.
    'storage_plugin_settings' => [
      'base_table' => 'organism',
      'base_column' => 'genus'
    ],
  ]
);
```

If we later want to create content, you will need to create at least all required fields in order for the content to be saved properly. You will also likely need to create tripal terms for any properties that this field has as well.

### Chado Testing Environment

The chado testing environment builds upon the Tripal testing environment. I will describe the chado specific portions in detail below but for more detail on the other steps you should check out the documentation on the *Tripal Testing Environment*.

1. Drupal sets up the testing environment including it's own database and fully functional site.
2. Tripal does not make any new changes to the environment but the Chado Test bases do. Specifically, the add the chado_installations table to the drupal schema and initialize TripalDBX in the test environment.
3. The code inside your tests *setUp()* method is run. The first thing that should be done in the setup for any test interacting with chado is to intialize the chado database.

   ```
   // Initialize the chado instance with all the records that would be present␣
   ↪after running prepare.
   $chado = $this->getTestSchema(ChadoTestBrowserBase::PREPARE_TEST_CHADO);
   ```

   The capitalized portion indicates the type of chado to initialize. In the above example, *PREPARE_TEST_CHADO* indicates the resulting chado schema will have all the records that would be present after running prepare. The options available are:

   - *INIT_CHADO_EMPTY*: creates an empty chado schema for testing. All tables are there but no records.
   - *PREPARE_TEST_CHADO*: creates a chado schema with all tables and all records that would be present after running "prepare" through the UI.
   - *INIT_CHADO_DUMMY*: creates a prepared chado schema with everything that PREPARE_TEST_CHADO and with additional test data. You can see the test data here in tripal_chado/tests/fixtures/fill_chado.sql

4. Finally your test method is called. Note: any services, plugins, etc. that you use here will only have the test environment available. You will not have access to any data in your main site, nor should this long term affect your main site. **That said, we do not recommend running tests on production sites!**
5. Once your test is complete, the *tearDown()* ` method is called to clean the entire development environment up. This includes dropping the development drupal tables including any changes made by your test.

### Retrieving the cvterm ID of a term in your test chado

Often in your setup you will be using Tripal DBX to insert records into your test chado instance. There is a handly function to help you look up the cvterm_id based on the accession:

```
$idspace = 'SO';
$accession = '0000704';
$cvterm_id = $this->getCvtermID($idspace, $accession);
```

The above example retrieves the cvterm_id for the gene term in the test chado database.

### Fields

For fields there are three main components to test:

1. That the data is created, loaded and updated appropriately from the backend data storage.
2. That the field classes match what is expected by the API and return the appropriate values.
3. That the edit form and page display perform as expected when rendered.

The first two lend themselves really well to kernel testing which is much faster than functional testing as it creates a more focused and streamlined test environment. That said, kernel tests are not run with a fully functioning Drupal site, but rather only specific functionality indicated by the test setUp is available. However, the third testing goal needs to interact with a rendered Tripal Content page and thus a functional test is required.

For more information on how to test each of the above goals, see the following tutorials:

### Testing Chado Field storage

> **Warning:** This documentation is still under development and is not complete.

As described in the documentation for how to create fields, Chado Fields depend on the developer to define a number of properties in order to describe to ChadoStorage how to create, load and update the various biological data associated with that field. For example, when creating a field to describe the organism associated with a gene, you will define properties for the genus, species, infraspecific type, infraspecific name, etc. Then ChadoStorage will use the property definitions to pull these data out of Chado and make them available to your field. In this tutorial, we are focusing on testing that the properties you defined in your field, act as you expect and that ChadoStorage is understanding your intent properly.

> **Warning:** All the following will assume you are developing these tests using the TripalDocker, with your module fully installed and mounted within the docker. For instructions on how to set this up see *Install with TripalDocker* documentation. In all the commands with CONTAINERNAME, replace it with the name of your container.

### Creating your testing Class

All tests are encapsulated within their own class as dictated by PHPunit. As such lets create a class skeleton as follows:

In *[yourmodule]/tests/src/Kernel/Plugin/ChadoStorage* create a file with a descriptive name for your test. We recommend naming it using the following pattern *[FieldName]Test.php*. The "Test" suffix is required by PHPUnit so make sure to at least include that in your test name or the test runner will not be able to find your test.

Here is a basic skeleton that you can copy/paste and replace the *TOKENS* with information for your own field:

```
<?php

namespace Drupal\Tests\YOURMODULE\Kernel\Plugin\ChadoStorage;

use Drupal\Tests\tripal_chado\Kernel\ChadoTestKernelBase;
use Drupal\Tests\tripal_chado\Traits\ChadoStorageTestTrait;
use Drupal\Tests\tripal_chado\Functional\MockClass\FieldConfigMock;

use Drupal\tripal\TripalStorage\StoragePropertyValue;
use Drupal\tripal\TripalStorage\StoragePropertyTypeBase;

/**
* Tests that ChadoStorage can handle property fields as we expect.
* The array of fields/properties used for these tests are designed
* to match those in the FIELDNAME field with values filled
* based on a CONTENTTYPE_VALID_FOR_THIS_FIELD content type.
*
* Note: We do not need to test invalid conditions for createValues() and
* updateValues() as these are only called after the entity has validated
* the system using validateValues(). Instead we test all invalid conditions
* are caught by validateValues().
*
* Specific test cases:
*  - TEST CASE DESCRIPTION
*
* @group YOURMODULE
* @group Fields
* @group ChadoStorage
*/
class FIELDNAMETest extends ChadoTestKernelBase {

  use ChadoStorageTestTrait;

  // We will populate this variable at the start of each test
  // with fields specific to that test.
  protected $fields = [];

  protected $yaml_file = __DIR__ . "/FIELDNAME-FieldDefinitions.yml";


  /**
   * {@inheritdoc}
   */
  protected function setUp() :void {
```

(continues on next page)

(continued from previous page)

```
    parent::setUp();
    $this->setUpChadoStorageTestEnviro();

    $this->setFieldsFromYaml($this->yaml_file, "testNAMEOFTESTCASE");
    $this->cleanChadoStorageValues();

    // Any code needed to setup the environment for your tests.
    // For example, you may need to insert records into the test chado
    // here if your field links to existing records.

  }

  /**
   * DESCRIBE YOUR TEST CASE HERE IN PLAIN ENGLISH.
   */
  public function testNAMEOFTESTCASE() {

    // PHPUnit complains if any test doesn't assert something.
    // We will just assert a basic fact here to confirm our test class is
    // setup properly.
    $this->assertTrue(TRUE);
  }
}
```

Now we will run our specific test in order to confirm that it is setup properly:

```
docker exec --workdir=/var/www/drupal9/web/modules/contrib/YOURMODULE \
  CONTAINERNAME phpunit tests/src/Kernel/Plugin/ChadoStorage/FIELDNAMETest.php
```

This will only run the tests in the test file we just setup. If you see errors regarding missing classes, then check that you have the *use* statements for those classes. If no test is found then make sure the class name matches the filename, the classname ends in *Test*, and the method name starts with *test*.

I am going to walk you through creating a test for the ChadoContactDefault field in this tutorial so all future code will show that case. You can see the finished test we are creating in the tripal/tripal Github repository: tripal_chado/tests/src/Kernel/Plugin/ChadoStorage/ChadoContactDefaultTest.php file.

For example, if I were to complete the above instructions to create a *tripal_chado/tests/src/Kernel/Plugin/ChadoStorage/ChadoContactDefaultTest.php* file containing the skeleton template above and execute:

```
 docker exec --workdir=/var/www/drupal9/web/modules/contrib/tripal tripal1587 \
  phpunit tripal_chado/tests/src/Kernel/Plugin/ChadoStorage/ChadoContactDefaultTest.php
```

I would get the following output:

```
PHPUnit 9.6.10 by Sebastian Bergmann and contributors.

Testing Drupal\Tests\tripal_chado\Kernel\Plugin\ChadoStorage\ChadoContactDefaultTest
.                                                                   1 / 1 (100%)

Time: 00:05.203, Memory: 10.00 MB

OK (1 test, 6 assertions)
```

You may find it helpful to include the *–testdox –verbose* parameters to *phpunit* when testing to obtain more verbose output.

### Defining the field instances to be tested

When a field is attached to a specific content type it is called a field instance. When testing fields, we work primarily with a number of test field instances focused on specific content types.

In this example, the field we are testing relates contacts to many other Tripal content types. While most content types have the same style linking table to the contact table, the arraydesign table has a different format. As such, the decision was made to test one random content type (i.e. study) and the arraydesign content type to capture both linking table variations. This is why in the following field instance definitions, you will see two instances: testContactFieldStudy and testContactFieldArrayDesign. The names do not matter so you might as well be descriptive of their purpose. Just ensure they do not contain spaces or special characters.

Field instances for testing are defined in a specific YAML format:

```
[test method which will be using the fields i.e. testNAMEOFTESTCASE]:
  [field name]:
    field_name: [field name]
    base_table: [base table]
    properties:
      [property key]:
        propertyType class: [full class name + namespace]
        action: [action]
        [additional key/value pairs associated with the action]
```

This YAML is stored in its own file in the same directory. In this case that would be tripal/tripal Github repository: tripal_chado/tests/src/Kernel/Plugin/ChadoStorage/ChadoContactDefault-FieldDefinitions.yml file and this file was referred to in the setup function of the test template.

### Methods provided by ChadoStorageTestTrait

The following methods are provided by the test trait and can be used to make testing the functionality a lot easier:

- *$this->chadoStorageTestInsertValues($insert_values)*: attempts to insert the values in the parameter for the specific field.
- *$retrieved_values = $this->chadoStorageTestLoadValues($load_values)*: attempts to load the remaining values specified by the few keys provided as parameters. ChadoStorage is given the values of all properties with drupal_store being TRUE by Drupal itself so those keys are passed in here when testing. The retrieved values are StoragePropertyValue objects so you need to use *getValue()* on them to retrieve the value loaded.
- *$this->chadoStorageTestUpdateValues($update_values)*: attempts to update the values of existing chado records to match the changes in the parameter.

You can see more details about how these are used to test in the finished ChadoContactDefaultTest test. There are also a number of additional examples of this testing in the same directory.

### 5.5.3 Additional Resources

- Official Drupal: Testing Documentation
- Official Drupal: PHPUnit file structure, namespace, and required metadata
- Official Drupal: Running PHPUnit Tests
- Official Drupal: PHPUnit Browser test tutorial
- Official Drupal: PHPUnit JavaScript test writing tutorial
- Drupal 8, 9, 10 Functional and Unit Testing (Automation Testing)
- Writing Automated Tests in Drupal 8, Part 4: Kernel tests
- Writing Automated Tests in Drupal 8, Part 3: Unit tests
- Drupal 8: Writing Your First Unit Test With PHPUnit

## 5.6 Hands-On Training

The following lessons provide hands-on training for various aspects of Tripal extension and module development. Each lesson is designed to provide simple, concrete examples for how to accomplish a set of goals. They are not designed to provide extensive background but rather will recommend other documentation for that purpose.

### 5.6.1 How to use Custom Tables in Chado

This lesson describes how to programmatically create and manage *Custom Tables in Chado*.

**Warning:** You should avoid making any changes to existing Chado tables as it could make upgrades to future releases of Chado more difficult and could break functionality in Tripal that expects Chado tabes to be a certain way. Instead, use custom tables!

#### Creating a Custom Table

To create a new custom table, you must first define the table schema which will include the table columns, constraints, default values, and indexes. This design must then be written using the the Drupal Schema API, which is a PHP associative array with key/value pairs that specify the components of the table. The following provides an example table schema array for a custom `library_stock` table that is intended to link records in the `stock` table of Chado with records in the `library` table of Chado:

```
[
    'table' => 'library_stock',
    'fields' => [
        'library_stock_id' => [
            'type' => 'serial',
            'not null' => TRUE,
        ],
        'library_id' => [
            'type' => 'int',
            'not null' => TRUE,
        ],
```

(continues on next page)

(continued from previous page)

```
        'stock_id' => [
            'type' => 'int',
            'not null' => TRUE,
        ]
    ],
    'primary key' => [
        'library_stock_id'
    ],
    'unique keys' => [
        'library_stock_c1' => [
        'library_id',
        'stock_id'
        ]
    ],
    'indexes' => [
        'name' => ['library_id', 'stock_id'],
    ],
    'foreign keys' => [
        'library' => [
            'table' => 'library',
            'columns' => [
                'library_id' => 'library_id'
            ],
        ],
        'stock' => [
            'table' => 'stock',
            'columns' => [
                'stock_id' => 'stock_id'
            ]
        ]
    ]
]
```

Note that in the array structure above, the columns, primary keys, foreign keys, unique keys, and indexes for the table are indicated.

The table can be created by calling the `create()` function of the Tripal Custom Table Service. To create the `library_stock` table defined in the array above we would use the following:

```
// Get an instance of the Custom Table service.
/** @var \Drupal\tripal_chado\Services\ChadoCustomTableManager $ct_service **/
$ct_service = \Drupal::service('tripal_chado.custom_tables');

// Use the service to create the table object by providing it a name.
/** @var \Drupal\tripal_chado\ChadoCustomTables\ChadoCustomTable $custom_table **/
$custom_table = $ct_service->create('library_stock');
```

The code above will create an instance of a `ChadoCustomTable` object but it does not yet create the table in Chado. To do that you must set the table schema in the following way:

```
// Provide the Schema API array defining the table structure.
$success = $custom_table->setTableSchema($schema);
```

In the code above, the `$schema` variable contains the Schema API array defined above. Calling `setTableSchema()`

will automatically create the table in the Chado schema and return TRUE on success. If there are any errors in the structure of the $schema array or any problems creating the table, messages will be logged to Drupal, the attempt will fail and the function will return FALSE.

### Locking a Custom Table

Tripal provides to the site developers an interface by which they can add custom tables. Site developers can see custom tables in the interface which allows them to delete them, rename them or alter them. If you are adding a custom table for use by your extension module and you do not want the site developers to alter it in any way, you can lock the table. Non-custom Chado tables are not available for alteration and custom tables that are necessary for the functioning of a module should not be either.

After creation of your custom table, you can lock the table from the site developers by calling the `setLocked()` function on the `ChadoCustomTable` object and passing `TRUE` as the only argument.

```
$custom_table->setLocked(TRUE);
```

### The Table ID

Every custom table in Tripal is given a unique ID. You can retreive this ID using the `getTableId()` function of the `ChadoCustomTable` object:

```
$table_id = $custom_table->getTableId();
```

Later, you can find the ID of any custom table using its name by calling the `findByName()` function of the Tripal Custom Table Service:

```
// Get an instance of the Custom Table service.
/** @var \Drupal\tripal_chado\Services\ChadoCustomTableManager $ct_service **/
$ct_service = \Drupal::service('tripal_chado.custom_tables');

$table_id = $ct_service->findByName('library_stock');
```

### Finding Custom Tables

After custom tables are created, you will most likely want to work with them in other parts of your module. You will need to have a `ChadoCustomTable` object anytime you want to work with a custom table. There are multiple ways that you can find a table and get a `ChadoCustomTable` object for it: by ID, by name or by iterating through all custom tables.

### Load by ID

If you know the ID of the table you can get a `ChadoCustomTable` object by calling the `loadById()` function the Custom Table Service:

```
// Get an instance of the Custom Table service.
/** @var \Drupal\tripal_chado\Services\ChadoCustomTableManager $ct_service **/
$ct_service = \Drupal::service('tripal_chado.custom_tables');

$custom_table = $ct_service->loadById($table_id);
```

In the code above, the `$table_id` argument would store the known ID of the table. The `$custom_table` variable is now a `ChadoCustomTable` object that can be used to work with the table.

### Load by Name

Custom table names should be unique. So, if you only know the table name, you can get a `ChadoCustomTable` object using the `loadbyName()` function of the Chado Custom Table Service.

```
// Get an instance of the Custom Table service.
/** @var \Drupal\tripal_chado\Services\ChadoCustomTableManager $ct_service **/
$ct_service = \Drupal::service('tripal_chado.custom_tables');

$custom_table = $ct_service->loadbyName('library_stock');
```

### Getting a List of Custom Tables

If you need to get a list of existing custom tables, you can retrieve the names and IDs by calling the `getTables()` function of the Tripal Custom Tables Service;

```
// Get an instance of the Custom Table service.
/** @var \Drupal\tripal_chado\Services\ChadoCustomTableManager $ct_service **/
$ct_service = \Drupal::service('tripal_chado.custom_tables');

$custom_tables = $ct_service->getTables();
```

In the code above, the `$custom_tables` variable will be an associative array where the keys are the table IDs and the values the table names for all custom tables.

### Deleting a Custom Table

You can delete a custom table by calling the `delete()` function on the `ChadoCustomTable` object. You must know the table ID or the table name to do so. Here is example code using the table name:

```
// Get an instance of the Custom Table service.
/** @var \Drupal\tripal_chado\Services\ChadoCustomTableManager $ct_service **/
$ct_service = \Drupal::service('tripal_chado.custom_tables');

$custom_table = $ct_service->loadbyName('library_stock');
$custom_table->delete();
```

### Changing a Custom Table

Suppose you have created a custom table for your module and released the module for others to use. Later you recognize you need to make changes to the custom table for improved functionality. To make changes to the table occur seamlessly for everyone who uses your module, you should create an update hook function in your module's `.install` file. Within the update hook function you should perform the following:

- Create a new version of the table.
- Copy the data from the old table.
- Delete the old table.

- Update your module to use the new table.

Then, when your module is upgraded on a Drupal site to the next version, the table changes will happen automatically.

### Using the Custom Table

After the custom table has been created you can use it the same as any other table in Chado. You can find examples for interacting with Chado tables in the *Tripal DBX: Generic cross database support for Drupal*.

CHAPTER

SIX

# UPGRADING TRIPAL

## 6.1 Upgrading a Tripal 3 site

The upgrade path is still under development. More information will be added here as it becomes available.

**What we know so far:**

- Upgrading from Drupal 7 to 8+ requires a migration.
  - This means you will create a local copy of your current Drupal 7 site, a new Drupal 8+ site and then use the Drupal Migration module to transfer your data from the Drupal 7 site to the new Drupal 8 one.
  - This process ensures unused or old configuration from previous upgrades is not transferred to your new site.
  - More information can be found here: Upgrading from Drupal 6 or 7 to Drupal 8 (and newer)
  - Drupal has provided the following documentation to prepare for a migration: How to prepare your Drupal 7 or 8 site for Drupal 9
- Upgrading from Tripal 3 to 4 will also use the Drupal migration functionality.
- Only Chado 1.3 will be supported in Tripal 4 so you need to upgrade Chado first.
- No re-loading or changing of Chado data will be required by the migration.

## 6.2 Upgrading an Extension Module

This page provides useful short snippets of code to help module developers upgrade their Tripal v3 compatible modules to work with Drupal 10. This list is not comprehensive or complete, but is meant to be an aid.

### 6.2.1 tripal_set_message() and tripal_report_error()

These functions have been upgraded and thus can be used as is. However, the new way is to use a logger service. For example:

```
$logger = \Drupal::service('tripal.logger');
$logger->notice('Hello world');
$logger->error('Hello world');
```

For more detailed information see the *Tripal Logger* documentation.

### 6.2.2 drupal_set_message()

Changelog: https://www.drupal.org/node/2774931

```
use Drupal\Core\Messenger\MessengerInterface;
// if not set by constructor...
$this->messenger = \Drupal::messenger();

// Add specific type of message within classes.
$this->messenger->addMessage('Hello world', 'custom');
$this->messenger->addStatus('Hello world');
$this->messenger->addWarning('Hello world');
$this->messenger->addError('Hello world');

// In procedural code:
$messenger = \Drupal::messenger();
$messenger->addMessage('Hello world', 'custom');
$messenger->addStatus('Hello world');
$messenger->addWarning('Hello world');
$messenger->addError('Hello world');
```

### 6.2.3 format_date()

```
\Drupal::service('date.formatter')->format($time);
```

### 6.2.4 Loading a User Object

To load a user using a known user ID.

```
// Load a user with a known UID in the $uid variable.
$user = \Drupal\user\Entity\User::load($uid);
```

To get the current user:

```
$current_user = \Drupal::currentUser();
$user = \Drupal\user\Entity\User::load($current_user->id());
```

### 6.2.5 Creating Links

To create HTML links the Drupal 7 was was:

```
$link = l('Administration', '/admin')
```

The Drupal 10 approach is:

```
use Drupal\Core\Link;
use Drupal\Core\Url;

$link = Link::fromTextAndUrl('Administration', Url::fromUri('internal:/admin'))
```

Using Links in *drupal_set_message*:

```
$jobs_url = Link::fromTextAndUrl('jobs page',
  Url::fromUri('internal:/admin/tripal/tripal_jobs'))->toString();
drupal_set_message(t("Check the @jobs_url for status.",
  ['@jobs_url' => $jobs_url]));
```

### 6.2.6 Database Queries

#### db_query

The *db_query* function is deprecated in Drupal 9. To perform a database query you will need to rework any calls to the *db_query* function in the following way:

```
// Get the database object.
$database = \Drupal::database();

// Perform the query by passing the SQL statement and arguments.
$query = $database->query($sql, $args);

// Get the result(s).
$job = $query->fetchObject();
```

#### drupal_write_record

The *drupal_write_record* was useful in Drupal 7 for directly working with tables that Drupal was aware of. Here's the replacement:

```
$database = \Drupal::database();
$num_updated = $database->update('tripal_jobs')
  ->fields([
    'status' => 'Cancelled',
    'progress' => 0,
  ])
  ->condition('job_id', $this->job->job_id)
  ->execute();
```

### 6.2.7 Views

#### The hook_views_data() function

The *hook_views_data* function is used to expose tables within Drupal to the Drupal Views. The function returns an array that defines how tables can be handled by Views. Fortunately, this is mostly backwards compatible and you can keep the function as is. However, you will need to make the following changes:

1. Where handlers are defined for the field, filter, sort, relationship, argument you must change the key *handler* to *id*.
2. Handler names are now just a single word. The following table provides some common name changes.

| Handler Type | D7 Handler Function | D8/9 Handler ID |
|---|---|---|
| field | views_handler_field | standard (strings) |
| | views_handler_field_numeric | numeric |
| | views_handler_field_date | date |
| filter | views_handler_filter_numeric | numeric |
| | views_handler_filter_string | string |
| | views_handler_filter_date | date |
| sort | views_handler_sort | standard (strings) |
| | views_handler_sort_date | date |
| argument | views_handler_argument_string | string |
| | views_handler_argument_date | date |
| relationship | views_handler_relationship | standard |

You can find additional handlers at these API pages:

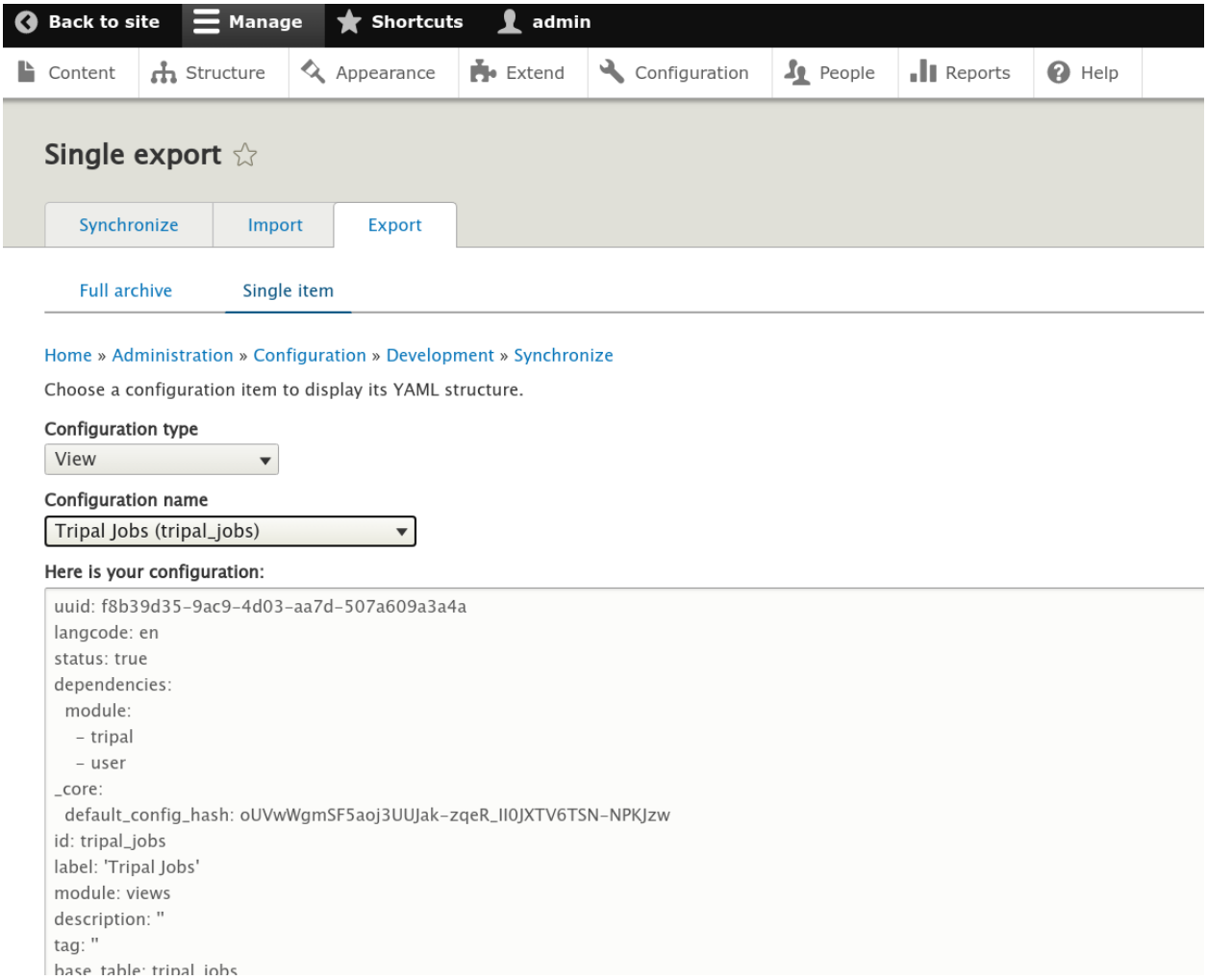
- Fields
- Filters
- Sort
- Arguments
- Relationships

### The hook_views_default_views() function

In Drupal v7 this function was used to provide the set of views that you would like the end-user to see automatically when the module is installed. This function is no longer used neither is the *<modulename>.views_default.inc* file where this hook would be stored. Instead the default views are provided in YML format.

**Step 1: Create the View**: To recreate any views that your module provided in Drupal 7, you must recreate the View using the Views UI interface. No coding is required.

**Step 2: Export the View**: Once the view has been recreated, you can export the YML for the view by navigating to `Admin` >> `Configuration` >> `Configuration Synchronization`. Click the `Export` tab at the top, then click the `single item` link below the tab. In the page that appears you should then select `View` from the `Configuration type` dropdown and then select the name of the view you want to export. The YML code for the selected view will appear in the textarea below. The screenshot below shows an example:




**Step 3: Create the View YML file**: Once you have the YML code for the view, you must create a new file named *views.view.<view_name>.yml* and place the code inside of it. Where *<view_name>* is the machine name of the view. You can safely remove the first *uuid* line. This file must be placed in the *config/install* directory of your module.

**Step 4: Reinstall the Module**: In order for Drupal Views to see this new view you must reinstall the module.

### Embed a View on a Page

In Drupal v7 you could embed a view onto any page by using code similar to the following

```
$view = views_embed_view('tripal_admin_jobs', 'default');
```

In Drupal 8 use code similar to the following to embed a view on a page:

```
$view = \Drupal\views\Views::getView('tripal_jobs');
$view->setDisplay('default');
if ($view->access('default')) {
  return $view->render();
}
else {
  return [
```

(continues on next page)

(continued from previous page)

```
    '#markup' => 'You do not have access to view this page.',
  ];
}
```

### 6.2.8 Attaching CSS

In Drupal 10 CSS files are part of "libraries". Libraries are groups of "assets" such as CSS, JS, or other resources needed for a particular set of pages that the module provides. Libraries are defined in the *<module_name>.libraries.yml* file. For information about preparing your CSS files with drupal see the page about adding css and js files to a module. Once the CSS is setup correctly, you want to add "libraries" to pages that use them. This is done by adding an '#attached' element to the render array returned by a page using the following form:

```
'#attached' => [
  'library' => ['<module_name>/<library_name>'],
]
```

Replace *<module_name>* and *<library_name>* with appropriate values.

CHAPTER

SEVEN

# CONTRIBUTING TO CORE TRIPAL

The following guidelines are meant to encourage contribution to Tripal source-code on GitHub by making the process open, transparent and collaborative. If you have any feedback including suggestions for improvement or constructive criticism, please comment on the Github issue. **These guidelines apply to everyone contributing to Tripal whether it's your first time (Welcome!) or project management committee members.**

**Note:** These guidelines are specifically for contributing to Tripal. However, we encourage all Tripal extension modules to consider following these guidelines to foster collaboration among the greater Tripal Community.

**Note:** Guidelines serve as suggestions ( **should** ) or requirements (**must**). When the word "should" is used in the text below, the stated policy is expected but there may be minor exceptions. When the word "must" is used there are no exceptions to the stated policy.

## 7.1 Guidelines for Contribution to Tripal

The following guidelines are meant to encourage contribution to Tripal source-code on GitHub by making the process open, transparent and collaborative. If you have any feedback including suggestions for improvement or constructive criticism, please comment on the Github issue. **These guidelines apply to everyone contributing to Tripal whether it's your first time (Welcome!) or project management committee members.**

**Note:** These guidelines are specifically for contributing to Tripal. However, we encourage all Tripal extension modules to consider following these guidelines to foster collaboration among the greater Tripal Community.

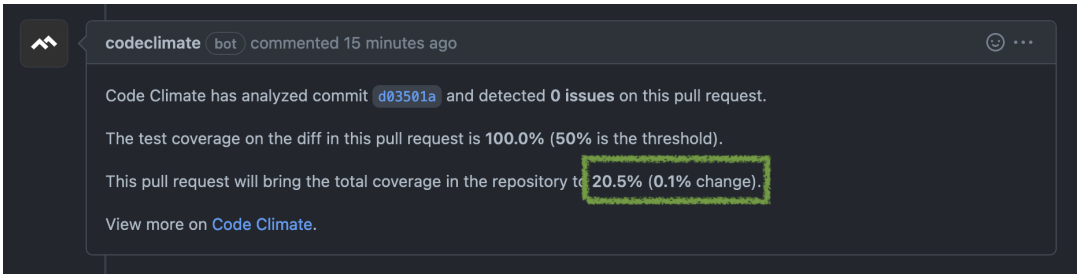
**Note:** Guidelines serve as suggestions ( **should** ) or requirements (**must**). When the word "should" is used in the text below, the stated policy is expected but there may be minor exceptions. When the word "must" is used there are no exceptions to the stated policy.

### 7.1.1 Github Communication Tips

- Don't be afraid to mention people (@username) who are knowledgeable on the topic or invested. *We are academics and overcommitted, it's too easy for issues to go unanswered: don't give up on us!*
- Likewise, don't be shy about bumping an issue if no one responds after a few days. *Balancing responsibilities is hard.*
- Want to get more involved? Issues marked with "Good beginner issue" are a good place to start if you want to try your hand at submitting a PR.
- **Everyone is encouraged/welcome to comment on the issue queue! Tell us if you**
  - are experiencing the same problem
  - have tried a suggested fix
  - know of a potential solution or work-around
  - have an opinion, idea or feedback of any kind!
- **Be kind when interacting with others on Github! (see Code of Conduct below for further guidelines). We want to foster a welcoming, inclusive community!**
  - Constructive criticism is welcome and encouraged but should be worded such that it is helpful :-) Direct criticism towards the idea or solution rather than the person and focus on alternatives or improvements.

### 7.1.2 Pull Request (PR) Guideline

The goal of this document is to make it easy for **A)** contributors to make pull requests that will be accepted, and **B)** Tripal committers to determine if a pull request should be accepted.

- **PRs that address a specific issue must link to the related issue page.**
  - In almost every case, there should be an issue for a PR. This allows feedback and discussion before the coding happens. Not grounds to reject, but encourage users to create issues at start of their PR. Better late than never :).
- PRs **must** be left unmerged for 3 weekdays to give core developers a chance to learn from each other and provide any feedback. Larger or particularly important/interesting PRs should be announced in the Slack #core-dev channel.
- PRs **must** describe what they do and provide manual testing instructions.
- PRs **must not** use any functions deprecated in the currently supported version of Drupal.
- **PRs that include new functionality must also provide Automated Testing.**
  - A PR should not reduce the overall test coverage of the repository. Code Climate will comment on your PR with the total coverage in the repository and include the change caused by your PR. This change must not be negative.
  - 

- PRs **must** pass all automated testing marked as "Required" at the bottom of the PR.
- **Branches must follow the following format:**
  - `tv4g[0-9]-issue\d+-[optional short descriptor]`
  - See the shared repository documentation for more details.
- **Should** follow Drupal code standards

#### How to create a PR

There are great instructions on creating a PR on Digital Ocean: How To Create a Pull Request on GitHub.

**The tl;dr version:**

1. Fork the repository or update an existing fork
2. Clone the fork
3. Create a branch specific to your change: `tv4g[0-9]-issue\d+-[optional short descriptor]`
4. Make your changes, committing often with useful commit messages.
5. Push your changes to your fork.
6. Create a PR by going to your fork: target should be `tripal:4.x`. For specifics, see guidelines above.

## 7.2 Code of Conduct

Be nice!

If that's insufficient, Tripal community defers to the Contributors Covenant which is included below.

### 7.2.1 Contributor Covenant Code of Conduct

#### Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, colour, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

#### Our Standards

Examples of behaviour that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behaviour include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behaviour and will take appropriate and fair corrective action in response to any behaviour that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

### Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

### Enforcement

Instances of abusive, harassing, or otherwise unacceptable behaviour may be reported to the community leaders responsible for enforcement at [INSERT CONTACT METHOD]. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

**1. Correction**

**Community Impact**: Use of inappropriate language or other behaviour deemed unprofessional or unwelcome in the community.

**Consequence**: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behaviour was inappropriate. A public apology may be requested.

**2. Warning**

**Community Impact**: A violation through a single incident or series of actions.

**Consequence**: A warning with consequences for continued behaviour. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

**3. Temporary Ban**

**Community Impact**: A serious violation of community standards, including sustained inappropriate behaviour.

**Consequence**: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

**4. Permanent Ban**

**Community Impact**: Demonstrating a pattern of violation of community standards, including sustained inappropriate behaviour, harassment of an individual, or aggression toward or disparagement of classes of individuals.

**Consequence**: A permanent ban from any sort of public interaction within the community.

#### Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.1, available at https://www.contributor-covenant.org/version/2/1/code_of_conduct/.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ. Translations are available on the Contributor Covenant website.

## 7.3 Shared Repository Management

This section describes the current guidelines for committing to the shared Tripal repository. These rules help keep the repository an organized place to work.

### 7.3.1 Branch Naming Conventions

`tv4g[0-9]-issue\d+-[optional short descriptor]`

**Where,**

- `tv4g[0-9]` indicates the roadmap group the branch relates to. You can see the listing of groups here.
- `issue\d+` indicates the issue describing the purpose of the branch. By making a new issue for each major task before we start working on it, we give room for others to jump in and save you time if something is already done, beyond scope, or can be made easier by something they are working on!
- `[optional short descriptor]` can be anything without spaces. This is meant to make the branches more readable so we don't have to look up the issue every time. You are encouraged to **only have one branch per issue!** That said, there are some edge-cases where multiple branches may be needed (i.e. partitioned reviews) where variations in the optional short description can make the purpose of multiple branches clear.

### 7.3.2 Outdated Branches

**It is important to only keep branches that you are actively working on.**

Branches should be deleted as soon as a pull request (PR) has merged them into the main branch.

#### Unable to Finish

If you simply **do not have time to finish** the issue associated with a branch, please do the following:

1. Communicate that in the issue. We all understand that academia is a constantly shifting world of priorities! There is no need to justify the change but we do really appreciate the following:
    - describing where you are currently at progress-wise
    - any difficulties or concerns to ran up against
    - what your plan or design were
2. Make sure all your code is committed, pushed to GitHub and add the last commit hash to your description in the above step. This allows us to recover your branch in case anything goes wrong with the patch created in the next step.
3. Create a patch capturing your current progress and attach that to the corresponding issue. See "How to Save Progress from a Branch" below for detailed instructions.
4. Unless someone has said they will take over the work within the next week, please delete the associated branch.

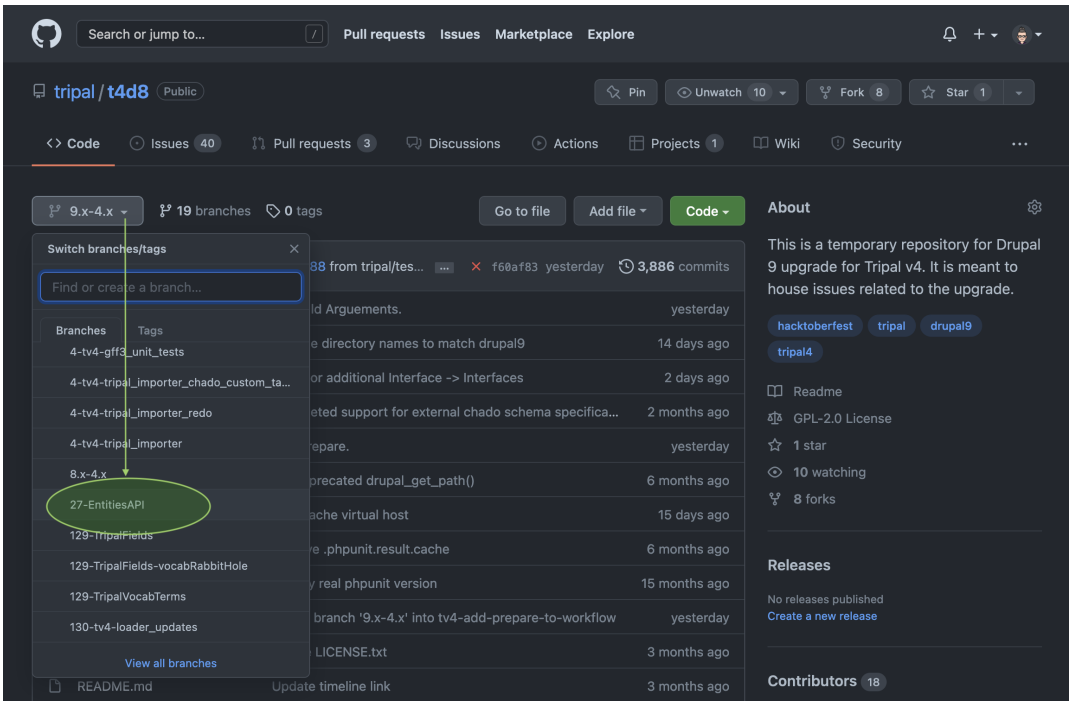
#### Other Reasons for Abandoning a Branch

If you find that you have to **switch gears, try a new approach or otherwise abandon** an open branch, please do the following:

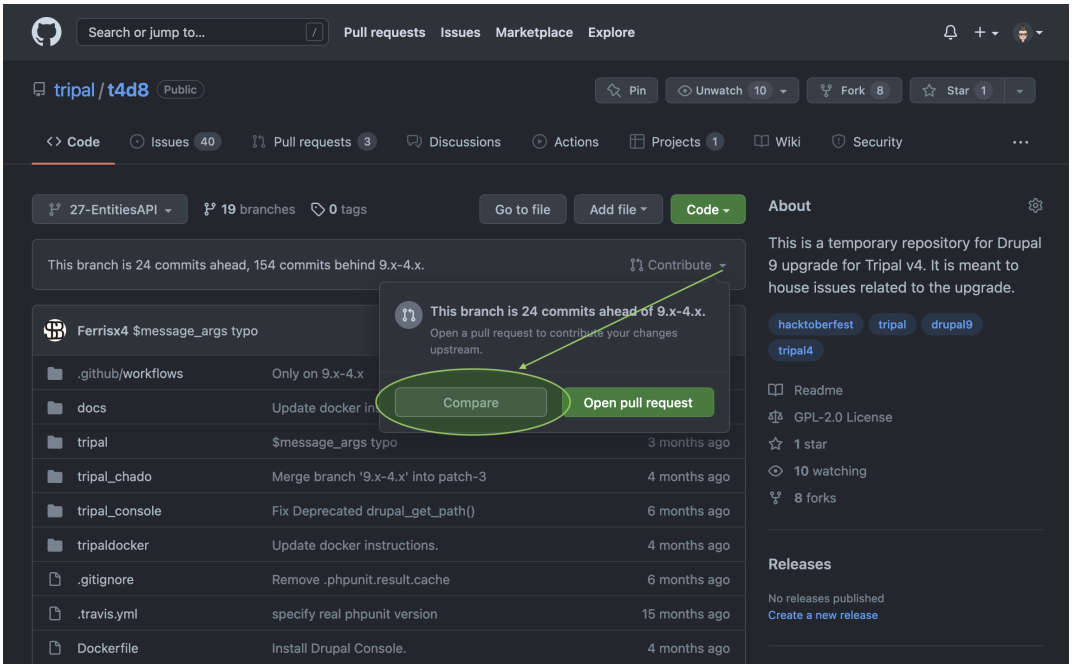
1. Make sure all your code is committed, pushed to GitHub and add the last commit hash to the issue associated with the branch. This allows us to recover your branch in case anything goes wrong with the patch created in the next step.
2. Create a patch capturing the changes in that branch and attach the patch file to the associated issue for the branch. See "How to Save Progress from a Branch" below for detailed instructions.
3. Include your reasoning for abandoning the branch in the issue associated with it. This does not have to be long but should be enough for anyone to understand your reasoning without needing to talk to you.
4. Delete the branch.

#### How to Save Progress from a Branch

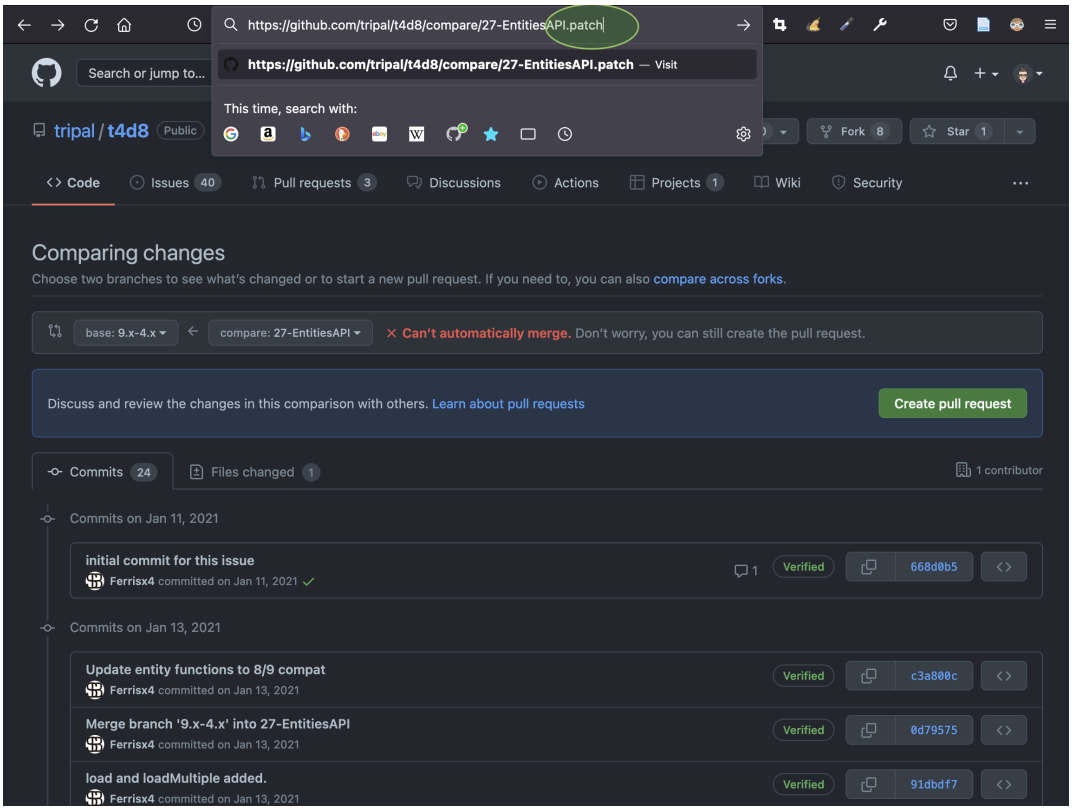
1. Make sure all your code is committed and pushed to GitHub.
2. Go to our repository on GitHub and select your branch from the drop-down list on the left side to switch to that branch.



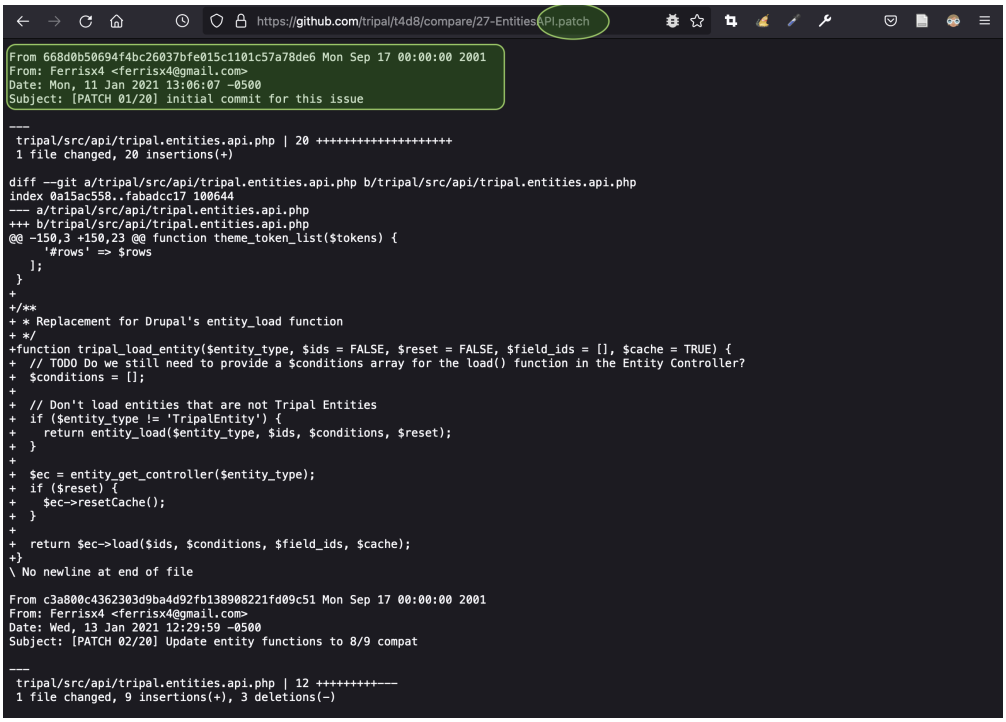

3. Click on “Contribute” on the right side and then “Compare”.




4. Add `.patch` to the end of the URL…




and click enter to see a text patch of all the changes. This will include any new files as well as smaller changes within existing files. Furthermore, it includes attribution to you to ensure that if this work is used in the future, you get credit for it.




5. Use your browser to save this page as a file and attach it to the issue by dragging it into the text area.

#### How to Recover an Previously Deleted Branch

1. Checkout the main branch and pull all changes.
2. Checkout the last commit made in the branch to be recovered. This commit hash should be recorded in the issue comments from when the branch was deleted.

   ```
   git checkout COMMITHASH
   ```

3. Create a new branch pointing to that commit. This branch should follow the naming conventions above.

   ```
   git branch RECOVERED-BRANCH-NAME
   ```

4. Push the new branch to GitHub and carry on like the branch was never deleted!

## 7.4 Creating a Docker for Testing

This section describes the procedure to create and run a docker container from a specific branch from the Tripal repository. This is helpful for testing your own proposed changes, or testing another contributor's proposed changes, since the full install process will be performed on the branch.

### 7.4.1 Testing on the most current development version

If you just want to test functionality of the current development version of Tripal, you can build a docker container as described in the Docker *Quickstart* section. If you need specific software versions or a specific branch, continue reading below.

### 7.4.2 Testing on an unmerged branch

1. Install Docker for your system.
2. Change to a suitable working directory on your local test system.
3. Clone the most recent version of Tripal 4, keeping track of where you cloned it. To keep things organized, you may want to include the issue number, in these examples it is `9999`:

   ```
   git clone https://github.com/tripal/tripal tripal-9999
   cd tripal-9999
   ```

4. If you want to contribute to core, you always want to make a new branch, do not work directly on the `4.x` branch. Use following naming convention for branches: `tv4g[0-9]-issue\d+-[optional short descriptor]`.
   - `tv4g[0-9]` indicates the functionality group the branch relates to. See tags for groups available.
   - `issue\d+` indicates the issue describing the purpose of the branch. By making a new issue for each major task before we start working on it, we give room for others to jump in and save you time if something is already done, beyond scope, or can be made easier by something they are working on!
   - `[optional short descriptor]` can be anything without spaces. This is meant to make the branches more readable so we don't have to look up the issue every time. You are encouraged to only have one branch per issue! That said, there are some edge-cases where multiple branches may be needed (i.e. partitioned reviews) where variations in the optional short description can make the purpose of multiple branches clear.

   Example for creating a new branch for creating a new field. Base your new branch on the main `4.x` branch:

```
git checkout 4.x
git branch tv4g1-issue1414-some_new_field
git checkout -b tv4g1-issue1414-some_new_field
```

Or if you want to test an unmerged pull request, it will be associated with a particular branch. You will see the branch name on the GitHub page for that pull request. Use this branch name in the following procedure. For example, it may appear as


```
git checkout -b tv4g1-issue1449-chadostorage-linkertables
```

If the contributor's branch is in their own repository, checking it out will be slightly different, you will need to include the pull request number. For example, for pull request #1535:

```
git fetch origin pull/1535/head:tv4g2-issue1534-chadoCvtermAutocompleteUpdate
```

5. We will now build the docker **image**, this takes a bit of time to complete. You may want to specify a particular Drupal version, PHP version, or PostgreSQL version. The PHP version is part of the docker file name, the other versions are specified through the `--build-arg` parameters. For example:

```
sudo docker build --tag=tripaldocker:testing-9999 --build-arg drupalversion=
↪"10.2.x-dev" --build-arg postgresqlversion="15" --file tripaldocker/
↪Dockerfile-php8.3 ./
```

6. We will now create a running docker **container** using the **image** we just built. We will map the web port *80* to a value available on the local test system. For example, we will select port `8080`:

```
sudo docker run --publish=8080:80 -tid --name=testing-9999 --volume=$(pwd):/
↪var/www/drupal9/web/modules/contrib/tripal tripaldocker:testing-9999
```

7. And finally we need to start up our PostgreSQL database inside the docker container.

```
sudo docker exec testing-9999 service postgresql restart
```

8. The Tripal site should now be available to evaluate at http://localhost:8080 or whatever other port number you selected.

   For more details about TripalDocker including the site administrator login information and more usage commands see *the install Tripal using Docker usage section*.

9. If you need a shell inside the docker, such as to run a drush command, use

```
sudo docker exec -it testing-9999 /bin/bash
```

10. If at some point you reboot your test system, you can restart this docker container with:

```
sudo docker start testing-9999
sudo docker exec testing-9999 service postgresql restart
```

11. Listing existing containers, include `-a` to show containers that are not running.

```
sudo docker ps -a
CONTAINER ID   IMAGE                                    COMMAND     CREATED      ␣
```

(continues on next page)

(continued from previous page)

```
↪STATUS                    PORTS                                                ␣
↪          NAMES
9e29c051c2ed   tripalproject/tripaldocker:latest   "init.sh"   2 hours ago  ␣
↪Up 2 hours                5432/tcp, 9003/tcp, 0.0.0.0:8080->80/tcp, :::8080->
↪80/tcp   t4
2f7575fe3940   tripaldocker:testing-9999           "init.sh"   3 days ago   ␣
↪Exited (137) 2 days ago                                                        ␣
↪          9999
```

11. Listing existing images.

```
sudo docker images
REPOSITORY                  TAG             IMAGE ID       CREATED          ␣
↪SIZE
tripaldocker                testing-9999    6b09ee09dd54   29 minutes ago   1.
↪61GB
```

12. Cleanup. Stopping the docker container.

```
sudo docker stop testing-9999
```

13. Deleting the docker container and image when you are done with it.

```
sudo docker rm testing-9999
sudo docker rmi tripaldocker:testing-9999
```

CHAPTER

EIGHT

# DESIGN DOCUMENTATION

This is the location for active development of Software Design Documentation (SDD) for Tripal v4. This document is managed by the Tripal Project Management Committee (PMC) but authorship contributions are welcome by anyone involved in core Tripal 4 development.

## 8.1 Design Overview

### 8.1.1 Authors

#### Design Document Authors

The following individuals are primary authors of this document (in alphabetical order). However, contributions are welcomed by anyone.

- Sean Buehler
- Stephen Ficklin
- Lacey Sanderson

#### Tripal v4 Developers

The following individuals are currently contributing to active development of Tripal v4 (in alphabetical order):

- Sean Buehler
- Josh Burns
- Stephen Ficklin
- Valentin Guignon
- Rish Ramnath
- Lacey Sanderson
- Douglas Senalik

We would also like to acknowledge the assistance of other individuals who have contributed to Tripal v4 through various pull requests and activity at Tripal Codefests. The full list of contributors is found here.

## 8.2 Module + File Structure

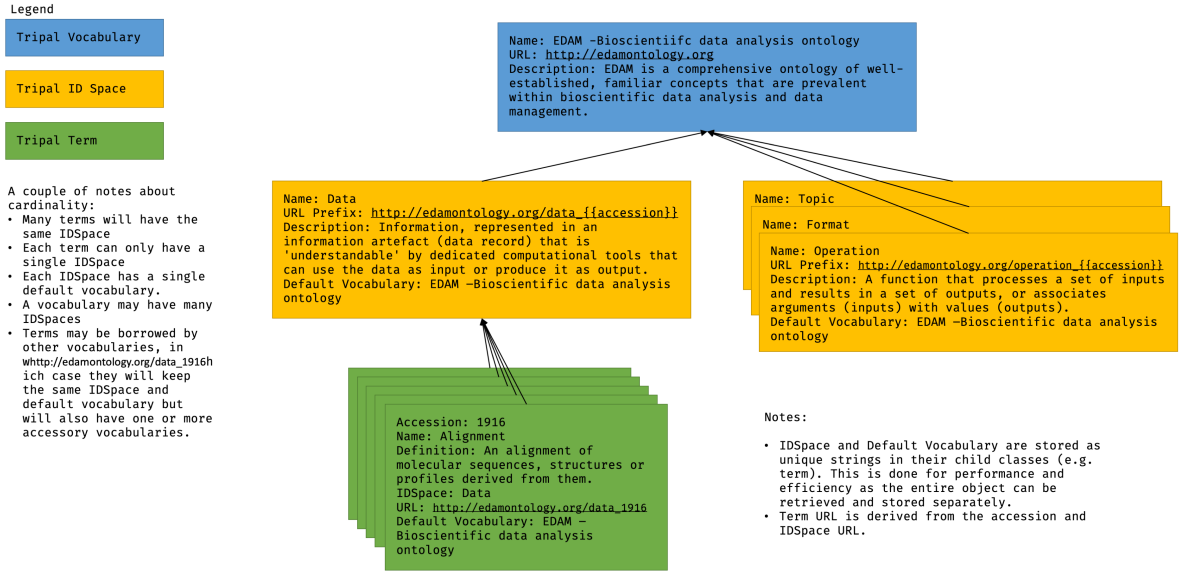
Tripal is a package of multiple Drupal modules (currently 4) with common documentation meant to be used together. We separate our code into multiple modules to allow site developers to choose the functionality they want and to improve maintainability. All modules require the base `Tripal` module but all others should be independent of each other completely and the base Tripal module should be able to be used alone.

- `tripal`: contains all generic Tripal functionality with a focus on APIs + Vocabularies, and Entities.
- `tripal_chado`: implements APIs in the `tripal` folder with specifics for supporting Chado. Additionally, this includes many data importers and eventually fields, as well as, Chado-specific APIs.
- `tripal_console`: Tripal implementations of Drupal Console commands. This is focused on making the development of Tripal easier and is meant to include commands for generating Tripal/Chado plugin files and re-writing Tripal 3 field classes in the new Tripal 4 way. Note: Drush commands are still used for the administration of Tripal and should go in the appropriate submodule.
- `tripaldocker`: Provides a Docker image currently focused on Tripal development. There is a plan to make this a production-ready Docker image in the future.
- `docs`: contains our official Tripal 4 ReadtheDocs documentation.
- `.github`: contains GitHub-specific files such as our testing workflow/actions.
- `.gitignore`: includes patterns for files that should not be committed to our repository using git.
- `composer.json`: described our PHP package to Packagist using Composer.
- `composer.lock`: formed when you install Tripal using composer and keeps track of any dependencies. We commit it so you can see versions of dependencies tests were last run using.
- `phpunit.xml`: our PHPUnit test configuration.
- `README.md`: our face to the developer community and the best place to start.

## 8.3 Controlled Vocabulary Design

Tripal is very ontology-focused with public terms forming the basis of our content types and fields. This sets us up to support (1) rich semantic web ready web services, (2) detailed definitions for all content displayed to the user, and (3) intuitive and powerful search filters and facets.

The following figure provides an example of the relationships between Vocabularies, ID Spaces, and Terms in Tripal:




In order to model the above relationships, we developed the following design:

- Vocabularies are collections of ID spaces that are stored using implementations of the Tripal Vocabulary Plugin Type.
- ID spaces are collections of Tripal Terms tied to a single vocabulary that are stored using implementations of the Tripal ID Space Plugin Type.
- Terms will not be plugins and their storage will be handled by their ID Space.

We used the Drupal Plugin API to make it easy to provide different storage backends for controlled vocabularies.

For more in-depth documentation on this design, check out the following pages:

### 8.3.1 Design Requirements

The following are the requirements we took into account for our design. Please let us know if you have requirements not listed in the document.

#### 1. Support multiple data backends

A theme with Tripal 4 is flexible storage for data. We want to ensure our design for vocabularies supports Chado but also has flexibility to be extended (e.g. use Drupal database, graph database, multiple schema in Chado). We also want to make it easier to create custom storage backends than it was in Tripal 3.

### 2. Performance

Vocabularies and their terms are central to the organization of biological data in Tripal 4. We are focusing on performance to reduce barriers to using vocabulary terms extensively throughout your content.

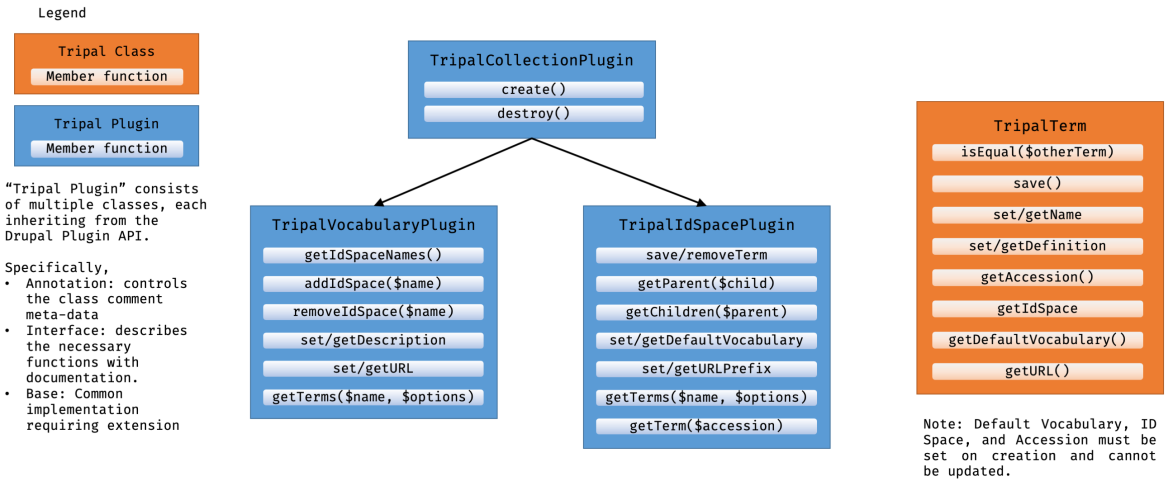
### 3. Support borrowing terms from existing vocabularies

As the available terms increases, we are seeing new ontologies choosing to borrow terms from existing ontologies. As described in Chado#68, Chado has difficulties storing these relationships. We want to ensure that our design takes borrowed terms into account in a Chado agnostic way.

### 4. Model vocabularies intuitively

Chado's storage of vocabularies can be a little confusing and not ideal (e.g. Chado#68). As such we want to design Tripal vocabularies independent of Chado and their storage in general. Specifically, we want to bring in the concept of ID Spaces as described in the OBO Format v1.4.

## 8.3.2 File Structure




The base structure of this API is found in `tripal/src/TripalVocabTerms/`. Specifically, you can find

- The **TripalTerm class**.
- The **base classes** to extend when making your own vocabulary plugin implementation.
- The **interfaces** you should implement are in the Interface directory and describe the methods you must implement in your vocabulary plugin implementation.
- The **annotation classes** describe the metadata needed in the comment header of your implemented plugin class.
- The **plugin managers** are in the PluginManager directory and simply link these plugins to the Drupal API.

**Warning:** Core Implementations for the vocabulary and id space plugins are still underway. The design base is in the main branch but work is not complete.

**All implementations** should be in the `src/Plugin/TripalIdSpace` and `src/Plugin/TripalVocabulary` directory of their respective modules. For example, a Chado implementation would be in `tripal_chado/src/Plugin/TripalVocabulary/ChadoVocabulary.php` and `tripal_chado/src/Plugin/TripalIdSpace/ChadoIdSpace.php`. You do not need to implement the TripalTerm class as all storage is handled by the vocabulary or ID space containing a given term.

## 8.4 Design and Coding Standards

### 8.4.1 Contributing to Design

We welcome anyone who would like to contribute to the Tripal v4 Design. The following policies should be followed by everyone contributing:

#### Verbal Communication

When verbal communication is necessary to work out a details, designers can meet in the following ways:

- Mondays between 16:00-18:00 UTC is the Tripal Developer Meeting in GatherTown. Join the Tripal Slack to get information to join these meetings.
- Zoom or Slack: currently anyone working with the design team can create impromptu scheduled meetings. These only last as long or as often as the designers need. Reach out to those working on a particular item if you are interested in participating.

#### Design Documentation Standards

#### Organization

All documentation for Tripal v4 design can be found either throughout the Developers Guide or in *Design In Progress*.

Designs that are in active preparation should be found in the *Design In Progress* section. Docs that have been approved by the PMC are incorporated throughout the main documentation.

#### How to Contribute Design Documents

#### Adding New Designs

All design documentation must first be added to the *Design In Progress* section. Anyone working on the design can do so. To propose a pending design:

1. Clone or fork the *tripal* repository.
2. Create a new branch for your design.
3. Design documentation is housed in the *docs/design* folder.
4. Only add new documentation into the *docs/design/pending* folder. If you are unsure where to add your contributions please ask.

5. Follow RST markup.
6. Use the consistent RST headers as found in other design documents.
7. Once you have added your documentation, you can submit a pull request for the *4.x* branch.
8. Pending design do not need PMC review and can be immediately merged. However, the act of creating a pull request alerts the PMC that documentation is being prepared.

**Warning:** While you may start implementation of your design prior to formal approval by the PMC please remember that the PMC must approve all designs and the implementation of a design must match the documentation for full inclusion of new code into Tripal 4. For this reason, it is recommended to wait on implementation until the PMC has fully reviewed any design you submit.

### Submitting Designs

The PMC must approve all pending design documentation for it to be officially part of the Tripal v4 design. Designs that get approved are moved into the core documentation. These designs can still be altered but are now considered "official". To submit a pending design do the following:

1. Move your documentation from the *docs/design/pending* section to the Developers Guide (i.e. within *docs/dev_guide*). Ask if you are unsure of where to place it.
2. Submit a pull request requesting review by the PMC.
3. The pull request must stay in the queue for at least 2 days to allow for comment by the community. This is to allow others to have a say if they feel the design is lacking.
4. The PMC merge the pull request if the design passes review or suggest changes if needed.

**Note:** Members of the PMC who are involved in design development will also adhere to the rules for submitting designs for approval in order to allow the community to comment and to support transparency.

### Formatting Design Documentation

### Structure of the Document

### Headers

Use the following for headers

- # Page Titles
- =, for sections
- -, for subsections
- ^, for sub-sub sections
- ", for sub-sub-subsections

## 8.5 Design In Progress

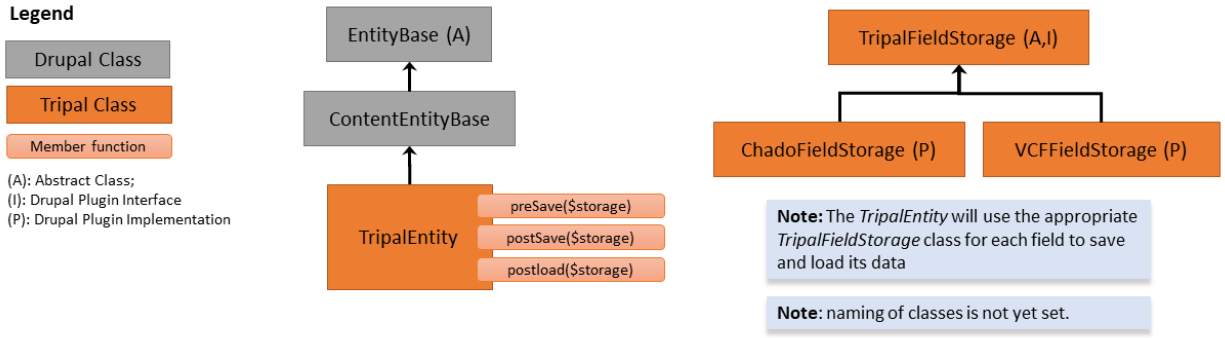
### 8.5.1 Entities and Fields Design

This design document is attempting to describe our current design process for Entities and Fields in Tripal 4 utilizing the new Drupal 10 APIs.

#### Entity/Field Design Summary

**Note:** The names of classes described below are not officially set and design will be updated as this evolves.

The following figure gives a high-level overview of the planned classes and their relationship to the Drupal API:




While the design is not complete at this point, here is a brief summary of the overall plan.

- The Drupal ContentBaseEntity will be extended to further support biological data and multiple data sources.
- We will override the ContentBaseEntity::preSave(), ::postSave() and ::postLoad() methods to move storage handling out of the entity and into a per field implementation.
- These overridden methods will call the appropriate TripalFieldStorage plugin implementation(s) for the fields attached to a given entity.
- Each field will indicate it's preferred storage plugin and administrators will have the ability to change the storage plugin used on their site.
- All TripalFieldStorage plugins will return data using a well documented data array. Controlled vocabularies will play a critical role.
- Biological data will not be duplicated in the Drupal database.
- We will create Drupal Console commands to upgrade old Tripal3 Fields to the new architecture.

### Entity/Field Design Requirements

The following are the requirements we are taking into account for our design. Please let us know if you have requirements not listed in the document.

#### 1. Multiple data backends per Content Type

Tripal needs to **support multiple data backends on a single Tripal Entity Type; specifically, on a per field basis.**. For example, a SNP entity type should be able to have data from Drupal (application-specific), Chado (biological metadata-specific), VCF (genotypic data), genetic map files, GWAS-related files, etc. This ensures that biological data can be stored in the format which most makes sense for that data whether that be a flat-file format or a database. Furthermore, it reduces data duplication by allowing support for original file formats rather then requiring all data to be imported into a single database.

Unfortunately this requirement is in direct conflict with the new Drupal 9+ paradigm of requiring a single data backend per entity type. All existing extension modules are in keeping with this Drupal paradigm (including External Entities). Since this assumption is interwoven in many of the Entity and Content Entity classes, we cannot simply extend the core classes for our design.

#### 2. Tripal Fields control their own data load + save

As mentioned above, each field should be able to determine it's own data storage. This functionality was available in Tripal 3 and supports easy overriding of data storage through the creation of new fields. For example, multiple groups can create genotypic data fields for genetic marker pages which cater to their specific storage paradigm. Furthermore, this allows fields to only load the data they want to display which is more efficient then the entity needed to load all data to support all fields.

#### 3. Entities + Fields should be vocabulary-focused

Tripal 3 ensured that all Tripal Entity Types and Tripal Fields needed to be associated with a controlled vocabulary term (preferably from a published ontology). This supports better cross database communication through the use of standard ontologies. Additionally, it provides important information for semantic web services by ensuring all data in Tripal is highly typed and these types are well-described with definitions and relationships.

#### 4. Low data duplication

Biological data can be quite large, especially when important metadata for each data point is included. As such, we would like to duplicate as little data as possible in order to keep database size manageable.

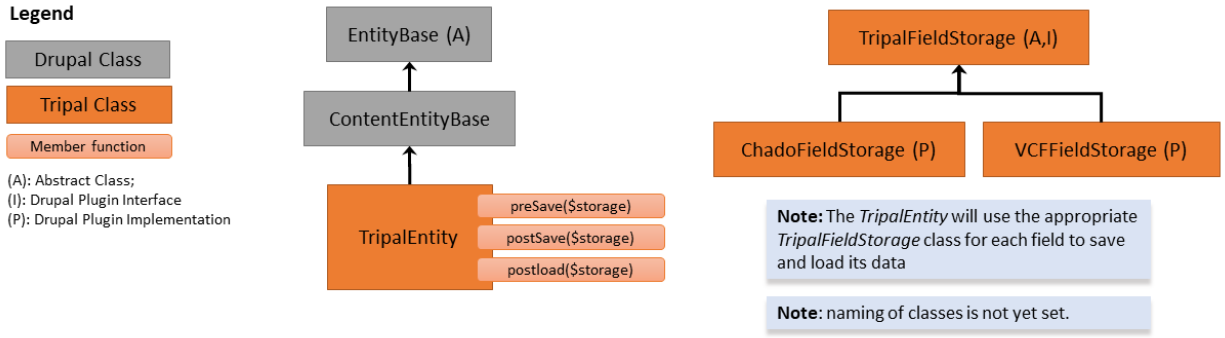
#### 5. Tripal Fields are easy to extend

The data types and display for each Tripal site can be extremely diverse depending on their audience. For example, the data needed for a metabolic focused community is quite different from that of a breeding focused community. As such, fields need to be very easy to extend to allow Tripal sites to support their individual communities data needs.

### 6. Upgrade path from Tripal 3

It needs to be as easy as possible to upgrade Tripal 3 fields, widgets and formatters to Tripal 4. Based on the data diversity we mentioned above, we have seen a huge number of Tripal fields developed for Tripal 3. In order to ease upgrade to Tripal 4 we need to take into account the sheer volume of fields being upgraded and ease the process as much as possible. For reference, here is the Tripal 3 Field documentation.

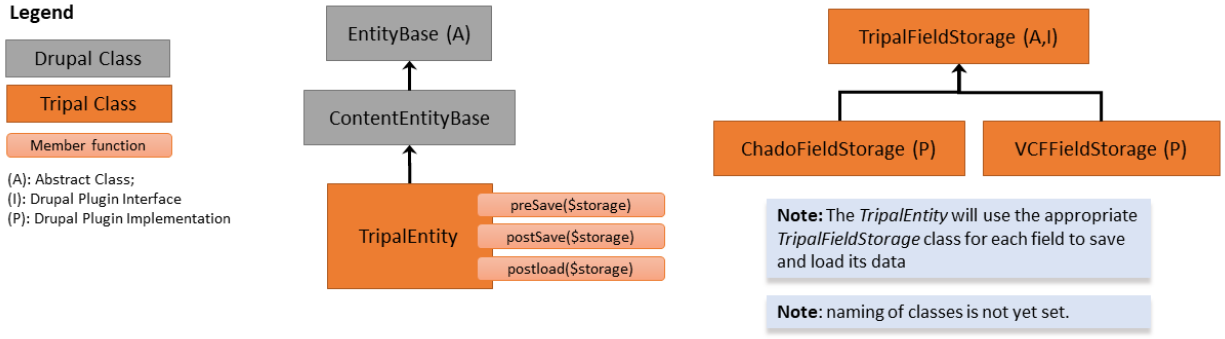
#### TripalEntity




The TripalEntity class is an instance of a ContentBaseEntity interface. It inherits the functionality of a Drupal Entity but allows us to provide Tripal specific customizations. In particular the following functions will be overridden:

- ***preSave()***
  - Cache the biological field data provided by the user.
  - Remove the biological data so Drupal doesn't store it using it's own *EntityStorageInterface*. If Drupal stores anything it will be context information for the TripalFieldStorage plugin implementations.
  - Note: Caching and then removing the biological data prevents Drupal from duplicating it.
- ***postSave()***
  - Pulls the biological field data from the cache.
  - Determines the proper TripalFieldStorage implementation that is needed for each field
  - Calls the proper TripalFieldStorage instance for each field to save the data.
  - We are exploring performance improvements by passing multiple fields with the same storage at once.
- ***postLoad()***
  - Determines the proper TripalFieldStorage implementation that is needed for each field.
  - Passes the identifying context information Drupal saved for each field to the correct TripalFieldStorage plugin implementation based on the field definitions.
  - TripalFieldStorage instances will load the data for each field and add it to the entity.
  - We are exploring performance improvements by passing multiple fields with the same storage at once.

### TripalFieldStorage




Tripal will have a *TripalFieldStorage* abstract class which is a Drupal plugin interface. It will provide methods to support the following functionality

- *load*: for loading data from the underlying data store
- *save*: for inserting or updating data in the data store
- *delete*: for removing data in the data store

We are ensuring this plugin is completely agnostic to the details mentioned regarding the TripalEntity class. This ensures that this class will be unaffected if the design of entities changes. It also allows this plugin to be more intuitive and easier to implement then alternate data storage in Tripal3.

---

**Note:** More functionality may be added to this class but for now, the design is focused on these methods.

---

### Example Usage: Chado

The *TripalFieldStorage* class is data store agnostic. However, implementation of this class will be data store specific. The *ChadoFieldStorage* class is one such possible implementation where it is responsible for interacting with Chado in a PostgreSQL database. We currently expect that all interactions with Chado in such a class would occur using the new *BioDB* API that is currently being proposed by Valentin.

---

**Note:** As a note, we currently have the Chado API (flat functions) and the ChadoRecord class for interacting with Chado. While these will remain for backwards compatibility we anticipate they will be deprecated in favour of the *BioDB* API as it is matured.

---

## 8.5.2 Tripal Biological Database layer and Chado

This document describes the biological database API and the new Chado API.

### Introduction

At its very beginning, Tripal was created to enable the use of Chado schema under Drupal CMS. Since its version 3, Tripal design changed in order to be ontology driven (like Chado) but database agnostic. However, Tripal version 3 was only able to support the Chado database schema. With version 4, a new biological database layer has been added: the *Biological Database* layer. This layer provides a database API extending Drupal database API that enables the use of other database schemas in Drupal. While it is currently limited to PostgreSQL database type, it has no number or type of schema limit. It means that it supports querying more than one schema at a time, and it is not limited to Chado or Drupal.

Since the Biological Database layer is an API, it does not work on it own but is rather a basis to build other extensions that will work on proprietary schema definitions like Chado. Therefore, the Tripal Chado extension is provided within Tripal package as an implementation of the Biological Database Layer for Chado.

To sum up:

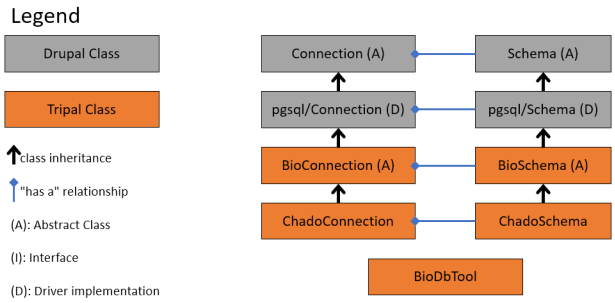
- The Biological Database API provides an abstraction layer to support any biological database schema.
- It supports multiple schemas and cross-schema queries.
- Tripal Chado extension is an implementation of the Biological Database API to support Chado schema.

### BioDB Design Summary

The Biological Database API provides an API for biological databases but also an API to manage concurrent data manipulation tasks: a Task API. This API is composed by a Task interface (and base implementation) and an extension of Drupal Lock API that supports shared locks. Finally, a set of exception classes has been added in order to finely manage thos new API exceptions.

### Biological Database API

The following figure gives a high-level overview of the classes provided by the Biological Database API and their relationship to the Drupal API:




This next figure shows which types of methods are provided by which class.


As shown above, BioConnection and BioSchema classes inherit from Drupal classes. You can find inherited methods from Drupal official documentation:

- Connection class
- Schema class

Many other methods are available and described in the source code of BioConnection, BioSchema and BioDbTool.

### Task API

The purpose of the task API is to provide a common class interface for all database-related tasks. Thus, all tasks can be managed from a same user interface or command line tool the same way.

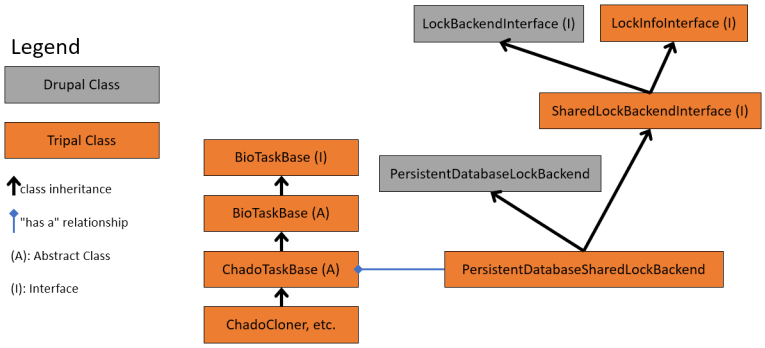
Obviously, sometimes, some database task may require more than a couple of seconds to perform their job. During that time, no other task should be allowed to modify the database concurrently to avoid data corruption. That's why a Lock API has been added in order to lock a database schema during a task and avoid data corruption issues.

While using an exclusive lock on a schema for a task that modifies the data, to prevent other tasks accessing that data, perfectly makes sense, tasks that just require the data to remain unchanged while they are completing their job could share a same "read-only" lock. Therefore, the lock API provides 2 lock flavors:

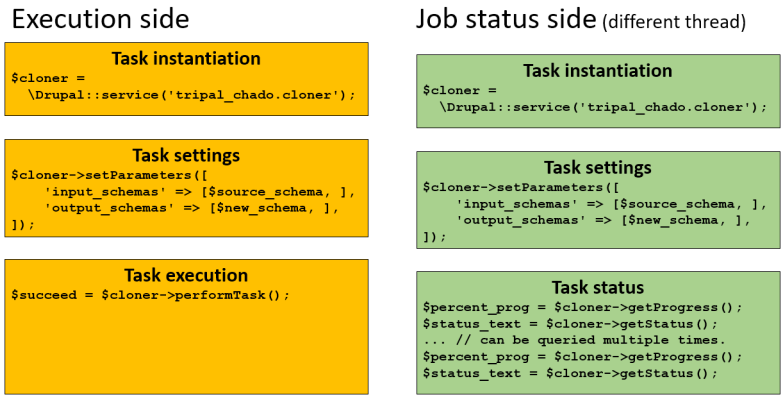
- exclusive locks for modified schemas
- shared locks for schemas only used for reading

Those two flavors are provided by the PersistentDatabaseLockBackend class through the methods ::acquire() and ::acquireShared().

The following figure gives an overview of the classes provided by the Task API:




The task life cycle is displayed in the figure below with the example of a database schema cloner task.




On the left side (yellow), the life cycle of a cloner task execution is described as follow:

1. First, a cloner task object is instanciated using Drupal services.
2. Then, its parameters are set. `$source_schema` would be the name of the schema to clone which will be locked with a shared lock (read only). `$new_schema` would be the name of the new schema that will receive a copy of `$source_schema`. `$new_schema` will be locked for exclusive use and no other task would be able to use it until the exclusive lock is released.
3. Finaly, the task is performed and an execution status is returned.

On the right side (green), the life cycle of a cloner task status tracking. In that part, a task is created and setup using the exact same steps 1. and 2. of the executed task. However, the third step differs since we just want the status of another task with the same parameters currenlty running. Two methods can be used:

- getProgress: to get the percent of task completion (returns a float number between 0 and 1).
- getStatus: to get a human readable text status of the current task.

### Code Examples

The following code can be used in modules or tested directly using `drush php`.

1. Dynamic query example on feature table in default Chado schema.

```
$biodb = \Drupal::service('tripal_chado.database');
$query = $biodb->select('feature', 'x');
$query->condition('x.is_obsolete', 'f', '=');
$query->fields('x', ['name', 'residues']);
$query->range(0, 10);
$result = $query->execute();
foreach ($result as $record) {
  echo $record->name . "\n";
}
```

See also Drupal dynamic queries.

2. Static query example on feature table in default Chado schema.

```
$biodb = \Drupal::service('tripal_chado.database');
$sql_query = 'SELECT name, residues FROM {1:feature} x WHERE x.is_obsolete = :obsolete␣
↪LIMIT 0, 10;';
$results = $biodb->query($sql_query, [':obsolete' => 'f']);
foreach ($results as $record) {
```

(continues on next page)

(continued from previous page)

```
  echo $record->name . "\n";
}
```

See also Drupal static queries and result sets.

3. Cross schema queries.

```
$biodb = \Drupal::service('tripal_chado.database');
$biodb->setSchemaName('chado1');
$biodb->addExtraSchema('chado2');
$sql = "
  SELECT * FROM
    {1:feature} f1,
    {2:feature} f2,
    {node_field_data} fd
  WHERE fd.title = f1.uniquename
    AND f1.uniquename = f2.uniquename;";
$results = $biodb->query($sql);
```

4. Chado installation task example.

```
$parameters = [
  'input_schemas' => [],
  'output_schemas' => ['chado'],
  'version' => '1.3',
];
$installer = \Drupal::service('tripal_chado.installer');
$installer->setParameters($parameters);
$success = $installer->performTask();
if (!$success) {
  echo "Chado installation failed. See logs for details.\n";
}
```

5. Chado cloner task example.

```
$parameters = [
  'input_schemas' => ['chado'],
  'output_schemas' => ['chado2'],
];
$cloner = \Drupal::service('tripal_chado.cloner');
$cloner->setParameters($parameters);
$success = $cloner->performTask();
if (!$success) {
  echo "Failed to clone schema. See logs for details.\n";
}
```

6. Chado upgrader task example with status tracking.

Execution thread:

```
$parameters = ['output_schemas' => ['chado'],];
$upgrader = \Drupal::service('tripal_chado.upgrader');
$upgrader->setParameters($parameters);
$success = $upgrader->performTask();
```

(continues on next page)

(continued from previous page)

```
if (!$success) {
  echo "Failed to upgrade schema. See logs for details.\n";
}
```

Status tracking thread (using the same parameters):

```
$parameters = ['output_schemas' => ['chado'],];
$upgrader = \Drupal::service('tripal_chado.upgrader');
$upgrader->setParameters($parameters);
$progress = $upgrader->getProgress();
$status = $upgrader->getStatus();
echo "Currently at " . (100*$progress) . "%\n" . $status;
```

7. Some random code.

```
// Get the BioDatabase tool.
$biotool = \Drupal::service('tripal_biodb.tool');

// Get Drupal schema name.
$biotool->getDrupalSchemaName();

// Test if a user-provided schema name is valid and not reserved.
if ($issue = $biotool->isInvalidSchemaName($schema_name)) {
  throw new Exception();
}
// If we want to check a reserved schema name.
$biotool->isInvalidSchemaName($schema_name, TRUE);

// Temporary reserve a new schema pattern to avoid its use by other modules.
$biotool->reserveSchemaPattern('mytests_*', 'Reserved for my tests.');

// Permanently reserve a pattern.
$config = \Drupal::service('config.factory')
  ->getEditable('tripal_biodb.settings')
;
// Warning: to not free other reservations, don't forget to get current
// config first and modify that array! Don't create a new one.
$reserved_schema_patterns = $config->get('reserved_schema_patterns') ?? [];
$reserved_schema_patterns['mytests_*'] = 'Reserved for my tests.';
$config->set('reserved_schema_patterns', $reserved_schema_patterns)->save();

// Get a new Chado connection using default Chado schema.
$biodb = \Drupal::service('tripal_chado.database');

// Create a new schema 'new_chado': 2 possible methods.
// - Method 1, using BioDatabase Tool:
$biotool->createSchema('new_chado');
// - Method 2, unsing a Chado connection: first we need to set the schema name
//   and then create it.
$biodb->setSchemaName('new_chado');
$biodb->schema->createSchema();
```

(continues on next page)

(continued from previous page)

```
// Copy some feature values from 'chado1' to 'chado2':
$biodb->setSchemaName('chado1');
$biodb->addExtraSchema('chado2');
$sql = "
  INSERT INTO {2:feature} f2
    (organism_id, name, uniquename, residues, seqlen, md5checksum, type_id)
  SELECT o2.organism_id, f1.name, f1.uniquename, f1.residues, f1.seqlen, f1.md5checksum,␣
↪c2.cvterm_id
  FROM {1:feature} f1
    JOIN {1:organism} o1 ON o1.organism_id = f1.organism_id
    JOIN {1:cvterm} c1 ON c1.cvterm_id = f1.type_id,
    {2:organism} o2,
    {2:cvterm} c2
  WHERE o2.species = o1.species
    AND c2.name = c1.name
    AND f1.uniquename LIKE 'NEW_%'
  ;";
$results = $biodb->query($sql);

// By default, ->select, ->insert, ->update, ->delete and other similar
// dynamic query methods of BioConnection will use Drupal schema. In fact, it
// is because those methods generate SQL queries using the table notation with
// simple curly braces (ie. "{some_table_name}") which will use Drupal table
// for backward compatibility with Drupal. It is possible to change that
// default to the selected biological schema. In order to use a Chado schema
// as default for those methods in other modules, the class or an instance
// must register itself as willing to use the biological schema by default:
$some_object = new SomeClass();
// Register any instance of the class:
$biodb->useBioSchemaFor(SomeClass::class);
// Another way would be to just register a specific class instance:
$biodb->useBioSchemaFor($some_object);
// Now calls to BioConnection dynamic query methods will work on the
// biological schema by default (until thread ends or call to
// ::useDrupalSchemaFor method).
// Note: if static queries in any of the registered classes need to use Drupal
// tables, instead of using the simple curly braces notation, the Drupal
// schema index must be specified explicitly. So "{some_table_name}" must be
// turned into "{0:some_table_name}".

// Execute a set of SQL commands on a given biological schema from an SQL file
// that may containt "SET search_path = ...":
// - Case 1: automatically remove any "SET search_path":
$biodb->executeSqlFile($sql_file_path, 'none');
// - Case 2: replace some schema names by others in every "SET search_path":
//   Here we replace every 'chado' by 'my_chado'.
$biodb->executeSqlFile($sql_file_path, ['chado' => 'my_chado']);

// Get the list of table in a biological schema.
$tables = $biodb->schema()->getTables(['table', 'view']);
$stock_table = $tables['stock'];
```

(continues on next page)

(continued from previous page)

```
// Get table definition with a simple array structure.
$biodb->schema()->getTableDef('stock', []);
// Get table definition from file version 1.3 in Drupal database API format.
$biodb->schema()->getTableDef('stock', ['source' => 'file', 'format' => 'drupal',
↪'version' => '1.3']);
// Get table definition from database as SQL DDL.
$biodb->schema()->getTableDef('stock', ['source' => 'database', 'format' => 'sql']);

// Clone 'chado' schema into 'chado2'.
// Method 1: using Biological Database API.
$biodb->setSchemaName('chado2');
$biodb->schema()->cloneSchema('chado');
// Method 2: using cloner service.
$parameters = [
  'input_schemas' => ['chado'],
  'output_schemas' => ['chado2'],
];
$cloner = \Drupal::service('tripal_chado.cloner');
$cloner->setParameters($parameters);
$success = $cloner->performTask();
if (!$success) {
  echo "Failed to clone schema. See logs for details.\n";
}


// Get Chado test schema base name.
$test_schema_base_names = $config->get('test_schema_base_names') ?? [];
$chado_test_base_name = $test_schema_base_names['chado'];

// Write tests for Chado operations.
use Drupal\Tests\tripal_chado\Functional\ChadoTestKernelBase;
class MyFunctionalTest extends ChadoTestKernelBase {
  // Get a temporary schema name.
  $biodb = $this->getTestSchema(ChadoTestKernelBase::SCHEMA_NAME_ONLY);

  // Create a temporary schema with dummy data.
  $biodb2 = $this->getTestSchema(ChadoTestKernelBase::INIT_DUMMY);

  // Create a temporary empty Chado schema with no data.
  $biodb3 = $this->getTestSchema(ChadoTestKernelBase::INIT_CHADO_EMPTY);

  // Create a temporary empty Chado schema with some dummy data.
  $biodb4 = $this->getTestSchema(ChadoTestKernelBase::INIT_CHADO_DUMMY);

  // ... test stuff ...

  // Once done, don't forget to free all used schemas.
  // If you forget, there is a garbage collecting system that will remove
  // unused schemas but warnings will be raised.
  $this->freeTestSchema($biodb);
  $this->freeTestSchema($biodb2);
  $this->freeTestSchema($biodb3);
```

(continues on next page)

(continued from previous page)

```
  $this->freeTestSchema($biodb4);
}
```