Database, 2018, 1–11
doi: 10.1093/database/bay099
Database tool

Database tool

Tripal Developer Toolkit

Bradford Condon[1], Abdullah Almsaeed[1], Ming Chen[1,2], Joe West[1] and Margaret Staton[1,*]

[1]Department of Entomology and Plant Pathology, University of Tennessee Institute of Agriculture, 2505 E.J. Chapman Blvd, 370 Plant Biotechnology Building, Knoxville, TN 37996 and [2]Graduate School of Genome Science and Technology, University of Tennessee, Knoxville, M411 Walters Life Science, Knoxville, TN 37996-0840

*Corresponding author: Phone: (865)974-7135; Fax: (865)974-4744; Email: mstaton1@utk.edu

Citation details: Condon,B., Almsaeed,A., Chen,M. *et al.* Tripal Developer Toolkit. *Database* (2018) Vol. 2018: article ID bay099; doi:10.1093/database/bay099

Received 5 June 2018; Revised 6 August 2018; Accepted 27 August 2018

Abstract

Tripal community database construction toolkit utilizing the content management system Drupal. Tripal is used to make biological, genetic and genomic data more discoverable, shareable, searchable and standardized. As funding for community-level genomics databases declines, Tripal's open-source codebase provides a means for sites to be built and maintained with a minimal investment in staff and new development. Tripal is ultimately as strong as the community of sites and developers that use it. We present a set of developer tools that will make building and maintaining Tripal 3 sites easier for new and returning users. These tools break down barriers to entry such as setting up developer and testing environments, acquiring and loading test datasets, working with controlled vocabulary terms and writing new Drupal classes.

Software URL: https://github.com/topics/tripal-developer-tools

Introduction

The rapid advancement in sequencing technology has resulted in a proliferation of genomic data. General, all-purpose databases such as NCBI capture some of this data, but additional support is needed for manual annotation, specialized analyses and data integration, particularly for groups not specializing in bioinformatics. Community-level genomics databases fill this role, hosting a curated set of data from a species or multiple species of interest.

In recent years, however, funding for these resources has been significantly reduced, even for large-scale model organism databases (1). To continue the critical role they play connecting researchers to tailored bioinformatic resources, community databases must formulate a plan to not only keep sites available but also continue to add new data and services.

The biological community database construction toolkit Tripal was created as a multifaceted solution to many of these problems (2). Tripal marries the content management system Drupal (http://www.drupal.org) to the standard biological relational database storage backend Chado (3). Tripal utilizes the same system of modular code units that have

Table 1. Key Drupal vocabulary and concepts

Drupal concept introduced in Tripal 3 (see http://tripal.info/node/347)	Definition	Example	Corresponding Tripal 2 concept
Entity types (Tripal entity type)	A generic container that defines a type of content. All Tripal content is of the TripalEntity type.	TripalEntity type, Page	Node
Bundle (Tripal content type)	An implementation of an entity type to which fields can be attached.	Organism	Node content type
Field	A piece of content that can be used in multiple bundles.	Common name, Genus, Species	
Entity	A particular instance of a Tripal Content Type.	A specific entry, such as the organism *F. excelsior*	Node

An entity type is essentially a generic container that defines a type of content, while a bundle is a more specific subtype that includes fields. For example, Drupal has a page entity type, with bundles for more specific types of pages, for example, a blog post or an article. These bundles include fields that define a smaller unit of content associated with a bundle. In the case of the blog post, fields would include a title, the author, the date it was posted and the body of the article. An entity is a piece of specific content, such as a specific blog post. In Tripal, examples of bundles would include units of content such as gene or organism. Fields would include gene name or genus and species. A single species, such as *Fraxinus excelsior*, would be an entity

made Drupal so successful. Tripal core provides a basic set of common functionality, including content types such as organisms, sequence features and controlled vocabularies. Extension modules can then be developed by any group to extend the core and provide additional functionality. Examples of extension modules include a BLAST tool for users (4), natural diversity genotype data loader and display (5, 6) and elasticsearch to provide fast sitewide searching (7). Tripal websites can reuse and share Tripal extension modules, meaning each website is not coding its own solution to shared biological problems. These advantages make Tripal an attractive option for community database platforms aiming to do more with fewer resources while remaining sustainable.

Tripal was first released in 2009 and has since had numerous improvements (8), the latest of which, Tripal 3 (9), includes improvements to interoperability, data loading and display and semantic web integration. In particular, usage of ontology-derived Controlled Vocabulary terms (CVterms) for all Tripal content will better position Tripal for semantic discovery and processing of biological data (10). Tripal 3 also upgrades many of the Drupal concepts, a necessary task as Drupal's release cycle marches forward. In particular, nodes are replaced with **Bundles**, **Entities** and **Fields** to allow more lightweight, flexible content type solutions (Table 1).

Developer challenges

The Drupal website lists 121 sites using Tripal as of this writing (https://www.drupal.org/project/usage/tripal), a number which continues to grow. As more communities rally behind Tripal, we hope the developer community, which has grown from two to thirteen contributors since its inception, will continue to grow alongside it. While it is advantageous that Tripal sites can be created with a smaller staff, this also introduces its own set of challenges. Creating custom Tripal modules often requires some understanding of biology, as well as significant expertise in computer science and web development. Maintaining the site (collecting, curating and uploading data) requires skills in web development, biocuration and bioinformatics. Development teams may not have the personnel to fulfill all these roles.

Tools that lighten the workload and ease the learning curve should be very welcome for new and veteran Tripal module developers alike. Drupal developer modules exist: the Devel module, for example, adds many helpful development features to your site (11). Generic Drupal tools exist to provide a Developer instance (12), PHPUnit testing (13) and CI (14), Entity management (15, 16) and Field Generation (17). However, because all Tripal content is a custom Tripal content type, few of the features these tools offer would be functional for Tripal and Chado.

Prior to this publication, no such Tripal-specific tools were available. We recognize that the skillset of a given development team may be lacking in biological or computer science expertise. Given that Tripal lives at the union of these disciplines, we identify challenges that a small development team is likely to face deploying a Tripal site and present a suite of developer tools to facilitate their resolution (Table 2), which we have developed in the course of our own work on the Tripal-based site Hardwood Genomics Project (18).

Developer instance. Developers should set up a developer instance to test code before deployment. Building, maintaining and rebuilding developer sites are time consuming and require system administrator knowledge that may be

Table 2. Tools presented in this paper

Name	GitHub repository	DOI	Description
Tripal Alchemist	https://github.com/statonlab/tripal_alchemist	DOI: 10.5281/zenodo.1187120	Convert Tripal Chado entities from one bundle to another.
TripalDock	https://github.com/statonlab/tripaldock	DOI: 10.5281/zenodo.1187125	Build, deploy and manage Tripal docker sites.
Tripal DevSeed	https://github.com/statonlab/tripal_dev_seed	DOI: 10.5281/zenodo.1205522	Lightweight dataset for module and site testing.
Docker images	https://github.com/statonlab/docker-containers	DOI: 10.5281/zenodo.1238033	Rapidly deploy single container pre-built Tripal sites.
TFG	https://github.com/statonlab/fields_generator	DOI: 10.5281/zenodo.1200661	Easily write custom fields with CVterms.
Tripal Test suite	https://github.com/statonlab/TripalTestSuite	DOI: 10.5281/zenodo.1204508	Add PHPUnit and CI testing with a single command

beyond developers with primarily biology backgrounds. A development environment preserves the functionality of your live site for your users, protects your data from accidental corruption and, if configured well, can greatly speed up your development time. For a Tripal site, a developer instance must consist of a **webserver** [apache (19) and nginx (20)] and **database** [PostgreSQL (21)] with Tripal's base dependencies installed (PHP, Drupal and PHP extensions) and configured correctly, as well as any additional services [phppgadmin (22) or elasticsearch (23)]. While the Tripal user's guide provides manual installation instructions (available at http://tripal.info/tutorials/v3.x/installation), this can be an overwhelming experience for new Tripal developers without prior web development experience. Docker containers help developers quickly and easily create shareable development environments, and using them is a developer's best practice (24). **TripalDock** automates and optimizes these tasks.

Data acquisition. Ideally, development sites are lightweight, but they still require biological data to test loading, display and manipulation of real data. Computer-science-focused developers may have difficulty understanding how to generate, format and load Chado with complicated interconnected bioinformatics datasets. Without test data loaded, however, they cannot reliably develop and test their code. Mirroring the dataset available on their live website may be impossible due to the size of genomic datasets, which can result in extremely large databases that take up too much storage space and run too slowly for the rigors of development. Scientific computing packages are sometimes distributed with small example datasets [for example, the Iris dataset in R (25, 26)]. **Tripal DevSeed** is a curated, miniature dataset in one place, covering all core Tripal functionality.

Testing and continuous integration. Code testing is widely appreciated by professional software developers for the many benefits it provides (27). Tests ensure that software produces the intended result given by various inputs, and their use can help discover errors or bugs in the code. Tests also ensure that previously written code is not broken by new additions, and they allow safe code refactoring. The most common testing framework for PHP, the coding language of Drupal and Tripal, is PHPUnit (28). As with most testing frameworks, it uses 'assertions' to verify that code behaves as expected. Consider a method that renames a gene feature. A suite of tests might verify that the given input features are correctly renamed and that the method can gracefully deal with problematic cases such as when the gene does not exist in the database or the new name is already in use. If the rename method is updated with new code, tests are rerun to ensure no new bugs have been inadvertently created by the new additions. Tests are so helpful that some development practices prescribe writing tests before writing the intended code (Test Driven Development), which can increase test coverage and coding efficiency (10).

Despite their importance, a novice Tripal developer may be intimidated by the overhead required for writing reproducible, platform independent tests that interact with Chado, Drupal and Tripal. Continuous integration (CI) is the practice of frequently merging new code during development, then automatically building and testing the new codebase to verify the integrity software. Adding CI testing to software allows projects to release updates more often (29) and to include more code from non-core developers without diminishing code quality (30). One good solution for Tripal developers to leverage is Travis CI, a free service to build and test code hosted at GitHub (31). Testing and CI are a huge topic, and biology-focused developers may not be familiar with it or why it is important to learn and use. The **Tripal Test Suite** encourages developers to learn about these software development practices and eases their initial set up by automatically adding the framework for PHPUnit and Travis CI to a Tripal module.

Entity management. Tripal code developers need to contend with new Drupal content-type concepts, including bundles, entities and fields. Tripal ships with a migration function to convert core node types (an earlier Drupal content type) to entities, but there is no tool to convert entities from one type to another. Additionally, in Tripal 2, many extensions provided custom node types. For example, analysis subtype modules defined BLAST and Interproscan analyses. The Tripal 3 migration process cannot support all custom nodes, so they migrate as analysis entities, and must be converted to custom entities to preserve unique functionality. As site administrators build their sites, they will likely want to further customize content types. **Tripal Alchemist** is an easy-to-use tool for this task.

Coding fields. Fields are an integral component of the new entity system. In simple cases, new fields can be easily created through the Drupal administrative web pages. However, defining fields programmatically require an understanding of Chado, Controlled Vocabularies and the Tripal field structure. Furthermore, multiple, interconnected files with strictly defined function naming are required, leading to simple mistakes that are difficult to troubleshoot. The **Tripal Fields Generator** (TFG) automates the field creation process.

The tools we have developed to facilitate our own Tripal development are open source and freely available, and we hope they will accelerate the development and sharing of new Tripal modules for others as well.

Materials and methods

Code standards and accessibility

All projects and tools are available on GitHub under an open-source license (GPLv3, located in the project root of each repository), included in each project's repository. Each individual project includes documentation for installation and use, as well as guidelines for contribution.

Tripal DevSeed sequences and annotations

Two hundred coding sequences (CDS) and their corresponding predicted amino acid sequences of the *F. excelsior* genome assembly were selected using the DevSeed minify.sh script and used to seed the developer dataset (32). All scripts used to download, minify and annotate the data are included in the devSeed project repo. The Kyoto Encyclopedia of Genes and Genomes (KEGG) database and software is proprietary, and therefore was annotated using the online webtool (see below).

The dataset was then annotated using the annotate.sh pipeline. CDS sequences were annotated using BLAST+ v2.7.1 (33) against UniProtKB/TrEMBL (downloaded July 2018) and the plant portion of UniProtKB/SwissProt (downloaded July 2018) (34). Amino acid sequences were annotated using InterproScan 5.30-69.0 (35) using with the iprlookup, goterms and pathways flags. Biosamples were randomly generated using the generate_biomaterials.py script, which randomly generates unique and shared key/value pairs and outputs NCBI biosample formatted XML files (36). Expression data was randomly generated for each Biomaterial and CDS pair using the DevSeed generate_expression.py script. The Newick format tree file was generated by aligning CDS sequences using MAFFT v7.402 (37). KEGG annotations were generated using the online KEGG Orthology And Links Annotation (KOALA) tool v2.0 (38).

Results

Developer instance tools

As described in the challenges, developer instance section, the first barrier to creating a Tripal module is configuring the developer environment. We present two platforms for rapid developer instance deployment. The first is a self-contained Docker image, with webserver and database. Docker provides a way to 'containerize' a software program or set of software programs, allowing them to be quickly and easily transferred among computing systems and re-deployed without any new configuration or installation effort. We have created Docker images for base Drupal (no Tripal installed), Tripal 2 and Tripal 3. The Tripal 3 image is also provided pre-loaded with a miniature dataset stored in Chado (described below). These images are designed specifically for Tripal module development. The Drupal image has customization in module locations, and as a result, TripalDock can keep the Tripal submodules in a mapped image separate from the Drupal modules. The Tripal images automatically download and use the latest version of the master branch of the Tripal GitHub account, ensuring the newest version is used when the container is deployed.

For those experienced with Docker, the provided images allow single-command deployment of a functional Tripal site. Because they are already initialized, deployment is extremely fast, taking seconds rather than minutes. For developers that need to build more customized and permanent containers and/or without prior knowledge of Docker, we provide a second solution, TripalDock. As opposed to

Table 3. Commands provided by Tripaldock

Command	Description
new	Create a new container.
up	Start the Tripaldock container.
down	Stop the Tripaldock container.
rm	Remove and destroy the Tripaldock container.
ssh	Interactive shell access to the container.
logs	Access the logs of various services running inside the container.
install	Equivalent to drush pm-enable tripal_module. Allows users to install modules from outside the container. (Drush is a command line shell for Drupal)
drush	Run drush commands from outside the container.

the Tripal images already built and described above, TripalDock is a command line tool that enables developers to create and interact with their own custom Docker containers. It provides simple commands to perform cumbersome Docker tasks and handles the mapping of files, creation of containers, execution of admin commands, dependency management and volume cleanup (Table 3). This tool allows new developers to get started immediately, avoiding the pitfalls of setup and configuration that can stump novice programmers.

Developer dataset

As described in the challenges, data acquisition section, acquiring and loading biological test data for development can be difficult and time consuming. We therefore generated a truncated dataset of 200 genes from the genome assembly of *F. excelsior* (32). Data types appropriate for all of the core and many common extension Tripal modules are available (Table 4) as a set, called **Tripal DevSeed**. Docker images can be downloaded with the data pre-loaded or developers can opt to download clean images and upload only the data relevant for their work. Detailed guides for loading data are included in the GitHub repository. Developers can also use **Tripal TestSuite** (see below) to run the DevSeed database seeder, which will automatically load the Tripal DevSeed dataset into their site.

Tripal DevSeed is accompanied by helper scripts to miniaturize and annotate an input dataset. The user need only provide three files: a set of CDS in FASTA format, a set of polypeptides in FASTA format and a GFF annotation file. The user can also specify a number of features, and the scripts will filter the input data to this number of records. The scripts will trim the input sequences and perform BLAST and InterProScan annotations. The only step that has not been automated is the KEGG KOALA, which must be performed by submitting the sequences through the KEGG website. The result is that developers can easily create their own miniature seed datasets that will be appropriate for their use case. Alternatively, developers can also create small datasets for data types that we do not include. These datasets can, in turn, be easily and granularly loaded using test suite's database seeders.

Table 4. Developer dataset contents

Name	Corresponding module	Format	Description
sequences	Core	FASTA	200 CDS sequences and their corresponding polypeptide sequences.
gff	Core	GFF	GFF3 describing gene, mRNA, exon, CDS and five/three prime UTR regions for landmarks containing the above 200 CDS sequences.
blast	tripal_analysis_blast	XML	BLAST annotations for CDS sequences against Swiss-Prot and TrEMBL database.
interproscan	tripal_analysis_interpro	XML	Interproscan annotations of polypeptide sequences.
kegg	tripal_analysis_kegg	TSV	KEGG annotations generated by the KEGG KOALA tool.
biosamples	tripal_analysis_expression	ncbi xml	Randomly generated NCBI XML formatted biosamples.
expression	tripal_analysis_expression	TSV	Randomly generated expression data for the above biosamples.
database_backups	NA	SQL	PostgreSQL database dump of all loaded biomaterials, facilitating one-step loading of a database.
tree	Core	Newick	Newick format tree generated from MAFFT-aligned CDS

Testing and CI

Testing is an essential step in software development and particularly important for shared community code development, yet many projects do not adopt a formal testing structure, typically because developers are pressured to provide results and feel they cannot afford the time spent learning a framework and writing tests. **Tripal Test Suite** lowers the barrier to entry for writing PHPUnit tests for Tripal: installing the package allows single-command set up of the necessary directory structure, environmental variables and database connection (see Table 5 for a full list of command line utilities and testing features). Command line utilities are used to set up the initial Travis and testing environment, create new seeders and tests and run database seeders. The listed features are for inclusion in tests and seeds, greatly simplifying interacting with Chado and Drupal within your tests. For example, test classes can wrap all tests in a database transaction and be rolled back automatically, preserving your database. **Data factories** allow users to easily add biological data into Chado that can be altered in the test and rolled back after it finishes. **Database seeders** are provided to allow developers to quickly load datasets into their Tripal site for long-term use. A database seeder comes pre-installed to load the **Tripal DevSeed** dataset, allowing developers to populate their sites with data with a single command, rather than manually submitting forms and loading each file. Additionally, by having a generic set of db seeder functions in Test Suite, any developer wishing to provide a new set of seed data for other data types (phenotypes, genotypes, etc.) could create their own small files and db seeder functions, mimicking the current DevSeed functionality for any type of data.

The package also automates the configuration of **Continuous Integration** with Travis CI: the user need only to push their code to GitHub and integrate their repository on Travis CI. Out of the box CI will ensure the module can be installed on a Tripal 3 site and that all PHPUnit tests pass. Tripal Test Suite also provides support to automate the process of inserting and removing test data into the database using database seeders, which make test data available only during the testing process (Note that ‘auto increment’ columns are not reset to their previous state). This concept is useful because it allows tests to manipulate the database without causing side effects by automatically

Table 5. Tripal Test suite Features

Command line	
Shell command	Description
init	Generates folders and files for PHPUnit testing and Travis CI. The PHPUnit environment bootstraps your Drupal site, tests and provides access to the TripalTestCase class. The .travis.yml file bootstraps a Tripal site in the Travis environment, enables the module and runs tests.
make:test	Creates a new namespaced Tripal Test in /tests/
make:seeder	Creates a new database seeder class. Seeders allow a new site to be quickly populated with a consistent dataset.
Db:seed [seeder]	Seeds the database with example data defined by seeder classes. If no seeder argument is specified, all seeds will run.

Test helper methods		
Feature	Usage (PHP)	Description
Transactions	use DBTransaction;	A PHP trait that wraps each test in the class in a transaction, undoing all changes to the database. This ensures that each test has a ‘known’ database state and that the developer’s site is not modified when testing.
Factories	$features = factory(’chado.feature’, 100)->create(); $this->publish(’feature’);	Creates Chado records. Arguments can be passed to specify the values of specific columns, or how many records to create. Combined with transactions, developers can easily define transient Chado data for their tests. The publish method generates entities for the specified records.
HTTP Requests	$response = $this->get(’/’);$response->assertStatus(200)->assertSee(’My Site’);	Tripal Test Suite’s HTTP methods allow you to call site URLs and confirm that the response and page content is as expected. Supported methods are GET, POST, PUT, PATCH and DELETE.
Silencing console	$output = silent(function() {drupal_json_output([’key’ => ’value’]); return true;}); $output->assertSee(’value’)->assertJsonStructure([’key’])->assertReturnEquals(true);	Tripal commands often output status messages to the console. When running large suites of tests, it is desirable to silence these messages so that test failures are not lost in status messages from passing tests. The silence method allows developers to prevent console messages, while providing methods to verify the messages are output.

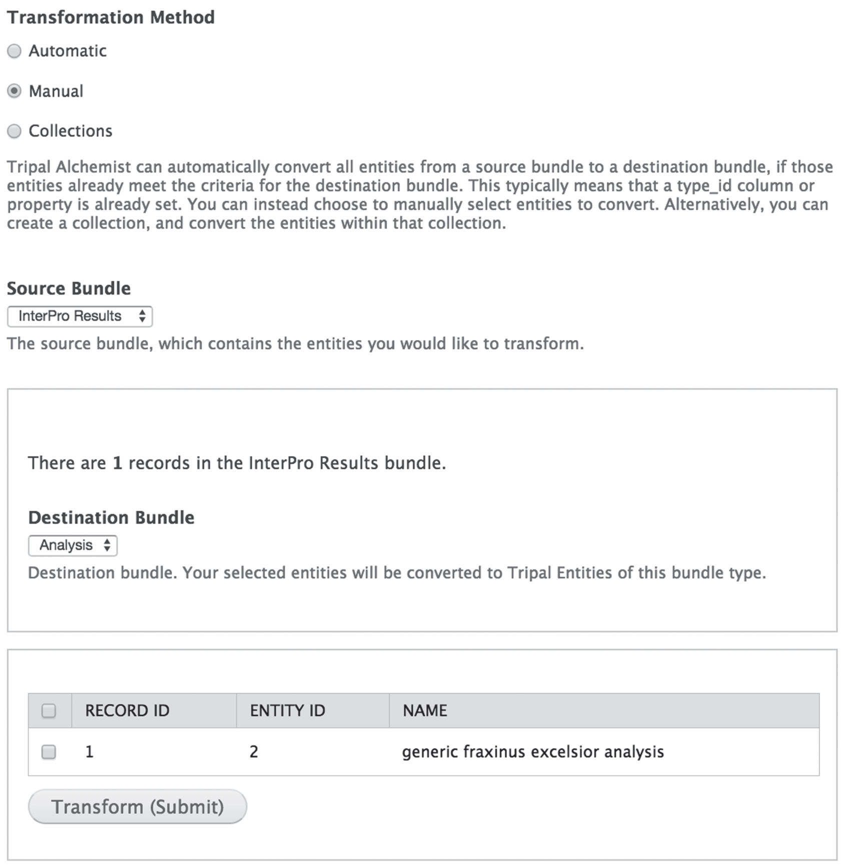
rolling back the database state upon the completion of tests or if a fatal error occurs during the testing runtime. By secluding data providers from tests, database seeders also make it possible to share code that populates the database with mock data with other modules that depend on the same data.

The Tripal Test Suite is now used within the Tripal Core repository itself. Using the same tool for modules and core makes it easier to write and structure tests.

Entities, bundles and fields

Tripal 3 introduces the concepts of entities, bundles and fields. The resulting content is more flexible, and, because bundles and fields are associated with CVterms, more discoverable and interoperable. This is another concept to master to successful development in Tripal 3 and choosing CVterms can be difficult, especially for non-biologists. We present **Tripal Alchemist** and **Tripal Fields Generator** as essential tools to help developers work with bundles/entities and fields, respectively.

Tripal Alchemist. **Tripal Alchemist** is an administrative tool that facilitates converting Tripal entities representing Chado records from one bundle type to another (Figure 1). The module provides three avenues for converting entities. 'Automatic' conversion will identify records whose type matches a different bundle. Analysis records, for example, migrate during the Tripal 2 to Tripal 3 upgrade process as a generic analysis bundle type, but they may have a specific type property. When the site administrator creates an appropriate bundle (BLAST annotation analysis type for example), matching entities will automatically be converted. 'Manual' conversion provides a table for users to select entities to convert, overwriting their type. This is useful when creating a new bundle type that applies to existing records, which do not have the matching property. 'Collection' conversion functions similarly to 'manual' but acts on a collection of entities that the administrator can define elsewhere. This method is ideal for converting large subset of entities. For example, we wished to convert the mRNA records for some (but not all) species to messenger ribonucleic acid (mRNA) contigs, allowing us to customize the fields displayed for these records.


Transformation Method

Automatic [✓ Manual] Collections

Tripal Alchemist can automatically convert all entities from a source bundle to a destination bundle, if those entities already meet the criteria for the destination bundle. This typically means that a type_id column or property is already set. You can instead choose to manually select entities to convert. Alternatively, you can create a collection, and convert the entities within that collection.

Source Bundle

InterPro Results

The source bundle, which contains the entities you would like to transform.

There are **1** records in the InterPro Results bundle.

Destination Bundle

Analysis

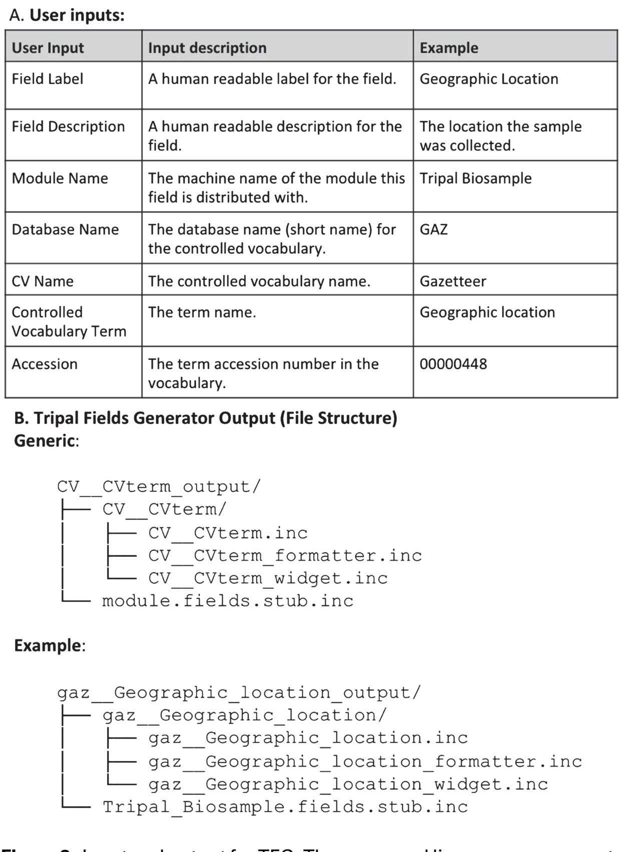
Destination bundle. Your selected entities will be converted to Tripal Entities of this bundle type.

☐	RECORD ID	ENTITY ID	NAME
☐	1	2	generic fraxinus excelsior analysis

Figure 1. The Tripal Alchemist administrator tool. The Tripal Alchemist administrator interface. To convert entities, the administrator must first select a transformation method (top). In this example, choosing the manual method allows conversion of selected entities from a source type to a destination type (middle). The available source entities are then presented in a table (bottom), which the user can select from and submit.

Any information a developer wants displayed on the website must correspond to a field. A properly coded field should handle querying and retrieving the data from the storage backend (Chado, if a Tripal Chado field), providing an interface for the user to enter data into the field when creating or ending the bundle it is attached to, as well as formatting the data for both web services and end user consumption. Furthermore, fields are CV-centric: each field must correspond uniquely to a CVterm.

Tripal Fields Generator. We created the **Tripal Fields Generator** tool to make coding fields significantly faster and easier. TFG is a command line PHP package easily installed with the PHP dependency manager Composer. It asks a series of questions about the CVterm usage of your field and generates stub files with base class methods and instructions for both TripalFields and ChadoFields (Figure 2). Three files are generated to properly define the field, with standard functions already named and left empty for the developer to add code: the class file, the formatter file and the widget file. Additionally, a stub file is generated demonstrating how to declare instances of the new field within the module. Because the cross-file references are generated automatically, this significantly reduces errors that are difficult to debug. Furthermore, TFG will connect to your Drupal site's database automatically and verify that the CVterm exists in your database.



A. **User inputs:**

User Input	Input description	Example
Field Label	A human readable label for the field.	Geographic Location
Field Description	A human readable description for the field.	The location the sample was collected.
Module Name	The machine name of the module this field is distributed with.	Tripal Biosample
Database Name	The database name (short name) for the controlled vocabulary.	GAZ
CV Name	The controlled vocabulary name.	Gazetteer
Controlled Vocabulary Term	The term name.	Geographic location
Accession	The term accession number in the vocabulary.	00000448

B. Tripal Fields Generator Output (File Structure)
Generic:

```
CV__CVterm_output/
├── CV__CVterm/
│   ├── CV__CVterm.inc
│   ├── CV__CVterm_formatter.inc
│   └── CV__CVterm_widget.inc
└── module.fields.stub.inc
```

Example:

```
gaz__Geographic_location_output/
├── gaz__Geographic_location/
│   ├── gaz__Geographic_location.inc
│   ├── gaz__Geographic_location_formatter.inc
│   └── gaz__Geographic_location_widget.inc
└── Tripal_Biosample.fields.stub.inc
```

Figure 2. Input and output for TFG. The command line program prompts the user to provide a series of inputs (**A**) and checks the database to determine if the specified controlled vocabulary term exists. If it does, the software produces a set of files in the correct directory structure to generate the field (**B**). An example is given using the 'Geographic location' term from the Gazetteer vocabulary for a mock module named Tripal_Biosample (**C**). This demonstrates how the user's inputs are structured into the file names. These inputs are also used to build all the necessary code variables and functions in each file, leaving only the essential custom coding for the function bodies for the developer to finish.

Use cases

To illustrate how a Tripal developer might utilize these tools in their own work, we have developed a set of three use cases, each of which highlights how one or more of the dev tools can be used. Each use case is derived from our actual experience developing Tripal modules for use on our website, HardwoodGenomics (18), and can be used for other Tripal sites as well.

Upgrading from Tripal v2 to Tripal v3. A developer is tasked with upgrading an existing Tripal 2 site to Tripal 3. The site includes BLAST and InterProScan annotations for millions of genes and differential expression data tied to NCBI Biological Samples. After performing the built-in migration, all analyses are now using the same generic analysis bundle, and all analyses are displayed the same way. The developer wishes to customize certain analyses by adding unique fields for each. For example, the developer wants the BLAST and InterProScan analyses to list the number of genes annotated. However, this wouldn't make sense for a differential gene expression analysis. For that type of analysis, the developer wants to display the number of significantly differentially expressed genes. The developer will accomplish this in Tripal by defining new bundles for these three types of analyses. Each bundle will be defined by using a CVterm for the analysis type [such as BLAST evidence (eco:0000206), Match to InterPro member signature evidence (eco:0000029) and Gene expression profiling (operation:0314)]. For each unique bundle, the developer can then add the unique fields that make sense for that particular analysis. At this point the developer is still left with the generic analyses that need to be converted to the new specific analyses. **Tripal Alchemist** provides a way to do this conversion automatically. The developer can use the module to take all analyses with the CVterm for BLAST evidence (eco:0000206) and automatically convert them to their new BLAST-specific analysis bundle.

Next, the developer wants to bring their Tripal 2 custom functionality to Tripal 3. They are working with a custom module called Tripal Sample, and they want to include a custom pane that displays the geographic location of a sample on a map. They use **Tripal Fields Generator** to define a new field, picking a CVterm for their field that exists in an ontology on their site. They pick the EDAM term 'Geographic Location', but forget that the accession is data:3720, using EDAM:3720 instead. Without Fields Generator, they would have struggled getting their field to work or inserted a bad term into their site. However, **Tripal Fields Generator** checked the database and informed them of their mistake before it was made. The tool creates the three field stub files for them (data__geographic_location.inc, data__geographic_location.formatter.inc and data__geographic_location_widget.inc) and the code to declare the new field (tripal_sample_module.fields.inc), greatly reducing the possibility of making a very-difficult-to-troubleshoot mistake (Figure 2).

Developing a new module. A developer is tasked with adding new functionality to their site: a genetic cross viewer to display F1 and F2 progeny phenotypes and genotypes. The first thing they might do is use **TripalDock** to create a new development site, so that they can work locally and not accidentally bring down their development site as they work. The developer is interested in displaying their cross data in the context of other data: phenotypes might link to genes and expression data. They therefore use **DevSeed** to quickly load in an organism complete with a miniature genome, biomaterials and expression data.

The developer begins in earnest on their module. They use **Tripal Test Suite**'s init method to automatically set up a PHPUnit testing environment and CI. As the developer starts to work on an importer for cross data, they write unit tests for functionality in the importer. The developer might work with a biologist at this stage to define the expected constraints on the data, such as 'Can an individual appear in the same cross multiple times?', 'Do all progeny always have both a phenotype and a genotype?'. These constraints can be included in tests to ensure the biologist and the developer both achieve consistent and expected functionality. Each test can be wrapped in a database transaction, so new methods being tested don't alter the database for older, working tests. When the importer is completed, the developer can easily refactor their code, simply re-running the unit tests after each refactor to ensure functionality.

The developer starts coding a field to display the cross data. They use **Tripal Fields Generator** to identify a meaningful CVterm (for example, Genetic Mapping-operation:0282) and to create the base field class, widget and formatter.

Training a new developer. A new developer has joined the team with some experience in bioinformatics, but no knowledge of Drupal, Chado or Tripal.

The junior developer's first task is to install a personal site using **TripalDock**. The site serves as a sandbox that can be easily destroyed and rebuilt, allowing them to learn and make mistakes without disrupting the deployed live site. It also allows them to begin learning Tripal quickly, without wrestling with system administration tasks (creating a virtual machine, webserver, etc.). Next, they would practice loading data into their Tripal site using **DevSeed**. The fully documented dataset will teach them how Tripal imports different types of data and how Chado stores it as well as how to navigate the administrative pages of a Tripal website. Once they are familiar with how to load data manually into Tripal, they will switch to the automatic loading provided by data seeders in **Tripal Test Suite**.

Once their site is set up, the new developer is tasked with creating a simple module to learn the Tripal framework. They use the **Tripal Test Suite** setup command to give their project a functional PHPUnit testing and Travis CI environment on GitHub, without spending weeks understanding how to structure their tests, access their Drupal site within the tests or create a Travis build with a functional Drupal. When they contribute code to other modules or to Tripal core, the test structure will match their learning module. Their learning module would use **Tripal Fields Generator** to build their first fields, automatically creating the necessary files in the correct location, with the required methods to fill out.

As the developer learns, they may load data into the development or production site as the wrong subtype. Developers with a computer science background might be overwhelmed with the number of chado feature types to keep track of: mRNA, mRNA_contig, polypeptide, CDS, 5′ UTR, 3′ UTR, exon, intron and gene. **Tripal Alchemist** makes correcting these mistakes trivial: any entities using the same base table can be easily converted, saving the team from deleting the offending entries and reloading.

When attempting to fix a problem on an existing Tripal module the team maintains, they introduce a crippling bug on the release branch. The CI that the team set up with **Tripal Test Suite**'s notifies the team that the module no longer installs and tests do not pass, and the team is able to revert the change. They instruct the new member on using GitHub best practices [branches, pull requests, (39)] to contribute, and CI tests on branches and pull requests allow the developer to submit functional code. Their own learning module used the same structure and framework for tests, so they will be able to write their own tests for their contribution.

Conclusion

The release of Tripal 3 has unlocked new possibilities for community genomics websites. As with many projects, the limiting factor for a Tripal site is funding and, relatedly, personnel. With smaller development teams, tools that bridge skill set gaps, either in the biological or computer sciences, can greatly enhance productivity. We have introduced a suite of utilities that will facilitate working on Tripal 3 sites and developing Tripal 3 modules. These tools encourage best programming practices for module and site development and ease the Tripal and Chado learning curves. The tools and use cases serve as a model for other open-source bioinformatics software communities for implementing best programming practices for developers with both biological and computational backgrounds.

Acknowledgements

We acknowledge the entire Tripal software developers community, particularly Stephen Ficklin (Washington State University) and Lacey-Anne Sanderson (University of Saskatchewan) for their leadership of the core Tripal infrastructure and the community of Tripal software developers.

Funding

National Science Foundation (#1443040 and #1444573).

Conflict of interest. None declared.

References

1. Kaiser,J. (2016) Funding for key data resources in jeopardy. *Science*, **351**, 14.
2. Ficklin,S.P., Sanderson,L.-A., Cheng,C.-H. *et al.* (2011) Tripal: a construction toolkit for online genome databases. *Database*, **2011**, bar044. https://doi.org/10.1093/database/bar044.
3. Mungall,C.J., Emmert,D.B. and FlyBase Consortium (2007) A Chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, **23**, i337–i346.
4. tripal_blast. https://github.com/tripal/tripal_blast (14 May 2018, date last accessed).
5. nd_genotypes. https://github.com/UofS-Pulse-Binfo/nd_genotypes (29 May 2018, date last accessed).
6. tripal_analysis_unigene. https://github.com/tripal/tripal_analysis_unigene (14 May 2018, date last accessed).
7. Chen,M., Henry,N., Almsaeed,A. *et al.* (2017) New extension software modules to enhance searching and display of transcriptome data in Tripal databases. *Database*, **2017**, bax52. doi: 10.1093/database/bax052.
8. Sanderson,L.-A., Ficklin,S.P., Cheng,C.-H. *et al.* (2013) Tripal v1.1: a standards-based toolkit for construction of online genetic and genomic databases. *Database*, **2013**, bat075–bat075.
9. Tripal 3. https://github.com/tripal/tripal (doi:10.5281/zenodo.1302070) 2018.
10. Hancock,J.M. (2014) Editorial: biological ontologies and semantic biology. *Front. Genet.*, **5**, 18.
11. Devel. https://www.drupal.org/project/devel (30 July 2018, date last accessed).
12. ddev. https://github.com/drud/ddev (30 July 2018, date last accessed).
13. PHPUnit. https://www.drupal.org/project/phpunit (30 July 2018, date last accessed).
14. Drupal - Travis Integration. https://www.drupal.org/project/drupal_ti (30 July 2018, date last accessed).
15. Node Convert. https://www.drupal.org/project/node_convert (30 July 2018, date last accessed).
16. Entity Type Clone. https://www.drupal.org/project/entity_type_clone (30 July 2018, date last accessed).
17. Field Type Generator. https://www.drupal.org/project/ftg (30 July 2018, date last accessed).
18. Almsaeed,A., Condon,B., Chen,M. *et al.* FAIRsharing.org: HWG; Hardwood Genomics Project. https://fairsharing.org/FAIRsharing.srgkaf (14 March 2018, date last accessed).
19. The Apache HTTP Server Project. https://httpd.apache.org (27 July 2018, date last accessed).
20. NGINX | High Performance Load Balancer, Web Server, & Reverse Proxy. https://www.nginx.com/ (27 July 2018, date last accessed).
21. PostgreSQL. https://www.postgresql.org/ (27 July 2018, date last accessed).
22. phpPgAdmin. http://phppgadmin.sourceforge.net/doku.php (27 July 2018, date last accessed).
23. Open Source Search & Analytics Elasticsearch. https://www.elastic.co/ (27 July 2018, date last accessed).
24. Boettiger,C. (2015) An introduction to Docker for reproducible research. *Oper. Syst. Rev.*, **49**, 71–79.
25. Fisher,R.A. (1936) The use of multiple measurements in taxonomic problems. *Ann. Eugen.*, **7**, 179–188.
26. R-core. Rdocumentation: Edgar Anderson's Iris Data. https://www.rdocumentation.org/packages/datasets/versions/3.5.1/topics/iris (5 August 2018, date last accessed).
27. Pan, J. Software Testing. http://users.ece.cmu.edu/~koopman/des_s99/sw_testing/ (26 July 2018, date last accessed).
28. PHPUnit – The PHP Testing Framework. https://phpunit.de/ (14 May 2018, date last accessed).
29. Hilton,M., Tunnell,T., Huang,K. *et al.* (2016) Usage, costs, and benefits of continuous integration. In: Open-source Projects, Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, New York, NY, USA. pp. 426–437.
30. Vasilescu,B., Yu,Y., Wang,H. *et al.* (2015) Quality and productivity outcomes relating to continuous integration in GitHub. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering; ESEC/FSE 2015. ACM, New York, NY, USA. pp. 805–816.
31. Travis CI - Test and Deploy Your Code with Confidence. https://travis-ci.org/ (14 May 2018, date last accessed).
32. Sollars,E.S.A., Harper,A.L., Kelly,L.J. *et al.* (2017) Genome sequence and genetic diversity of European ash trees. *Nature*, **541**, 212–216.
33. Altschul,S.F., Madden,T.L., Schäffer,A.A. *et al.* (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.

34. Schneider,M., Lane,L., Boutet,E. *et al.* (2009) The UniProtKB/Swiss-Prot knowledgebase and its plant proteome annotation program. *J. Proteomics*, **72**, 567–573.
35. Quevillon,E., Silventoinen,V., Pillai,S. *et al.* (2005) InterProScan: protein domains identifier. *Nucleic Acids Res.*, **33**, W116–W120.
36. Barrett,T., Clark,K., Gevorgyan,R. *et al.* (2012) BioProject and BioSample databases at NCBI: facilitating capture and organization of metadata. *Nucleic Acids Res.*, **40**, D57–D63.
37. Katoh,K. and Standley,D.M. (2013) MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Mol. Biol. Evol.*, **30**, 772–780.
38. Kanehisa,M., Goto,S., Sato,Y. *et al.* (2012) KEGG for integration and interpretation of large-scale molecular data sets. *Nucleic Acids Res.*, **40**, D109–D114.
39. GitHub best practices. https://resources.github.com/videos/github-best-practices/ (27 July 2018, date last accessed).