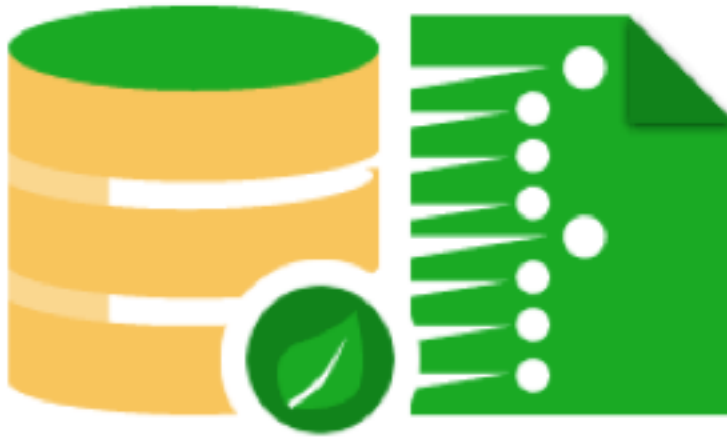

Tripal WS BrAPI Documentation

Tan R, Sanderson LA

Apr 21, 2023

Step-by-step guides:

- 1 Resource 3**
- 1.1 Requirements 3
- 1.2 Configuration 3
- 1.3 Configure Tripal Web Service BrAPI Module 5
- 1.4 Configure Tripal Web Service BrAPI Calls 6
- 1.5 Configure Tripal Web Service BrAPI Version 7
- 1.6 Configure Tripal Web Service BrAPI Search Calls Request Log 8
- 1.7 Tripal WS BrAPI Call Types 8
- 1.8 Setup Tripal WS BrAPI Call 9
- 1.9 Create Tripal WS BrAPI Custom Call 12
- 1.10 Create Tripal WS BrAPI Search Call 13
- 1.11 Create Tripal WS BrAPI Override Call or Call Alias 16
- 1.12 Contributing 17



Tripal Web Service BrAPI

A Drupal module that implements the Breeding API (BrAPI) standardized specifications for Tripal websites that host genotypic/phenotypic data stored in a CHADO Database. BrAPI is an interface for data interchange between crop breeding applications.

Breeding API (BrAPI)	http://docs.brapi.apiar.io
Tripal	http://tripal.info
CHADO	http://gmod.org/wiki/Chado
Drupal	https://www.drupal.org

1.1 Requirements

Tripal WS Brapi requires that you have the following installed or setup in your Tripal website.

Tripal 3 (with Tripal Web Service extension)	http://tripal.info
CHADO	http://gmod.org/wiki/Chado
Drupal 7.x	https://www.drupal.org

Since some database queries involve data from multiple Chado tables, an intermediate knowledge of Structured Query Language or SQL query, particularly using TABLE JOINS and combining conditions is beneficial in creating most BrAPI calls.

1.2 Configuration

TRIPAL_WS_BRAPI module generates a number of system variables that will give end users technical control over its general operation and call request response mechanism. Upon installation of this module, the following table summarizes all system variables created and the default value each one is set to.

FILE: <code>tripal_ws_brapi/includes/config.inc</code>
--

1.2.1 Summary table showing system variables:

Configuration	Use	Default Value
\$config_ws_brapi	Holds the name of the module. To prevent variable name conflict with other modules, all system variables are prefixed with the module name.	Do not change
\$config_version	The default BrAPI version of this implementation.	1.3
\$config_resultset_limit	Limit the number of items/data per page returned by a BrAPI call request	100 items per page
\$config_supported_method	REST request methods supported.	GET and POST
\$config_menu_level	An arbitrary levels may be added to request URL as outlined in BrAPI URL structure specifications. See BrAPI URL Structure	web-services
\$config_hook	External modules can implement a new call or override existing call through the use of Drupal hooks. The string value of this configuration is used to signal this module that a module is implementing calls.	tripal_ws_brapi_call
\$config_allow_override_hooks	In cases where an external module implements a call that is identical in name and version to an existing call, this option will decide which of the two call instances to apply.	no – use the local version of the call.

All configuration variables can be modified as desired, except for the module name. Be sure to save your changes before installing the module. Alternatively, most of these system variables can be altered using the configuration page after the installation routine.

Home » Administration » Tripal » Extensions

Tripal Web Service – BrAPI ▾ CONFIGURE TRIPAL WEB SERVICE - BRAPI BRAPI VERSION BRAPI CALLS SEARCH REQUEST LOG

⚠ This module is set to **allow** external/hook call to override existing call. Please see configure Tripal Web Service BrAPI tab for more information.

Calls

CALL	MODULE
<input type="checkbox"/> v1/calls	tripal_ws_brapi
<input type="checkbox"/> v1/commoncropnames	tripal_ws_brapi
<input type="checkbox"/> v1/crops	tripal_ws_brapi
<input type="checkbox"/> v1/germplasm	tripal_ws_brapi
<input type="checkbox"/> v1/search/germplasm	tripal_ws_brapi
<input type="checkbox"/> v2/commoncropnames	tripal_ws_brapi
<input type="checkbox"/> v2/serverinfo	tripal_ws_brapi
<input type="checkbox"/> v2/germplasm	tripal_ws_brapi
<input type="checkbox"/> v2/search/germplasm	tripal_ws_brapi
<input type="checkbox"/> v1/dbcalls	kp_searches
<input type="checkbox"/> v1/search/dbcalls	kp_searches
<input type="checkbox"/> v1/mycall	kp_searches

Figure 1 – Module configuration page showing all active BrAPI calls and the module each call is hosted. Series of

page tabs show sections of this module that can be configured.

Configuration page allows system administrator to set different values to system variables outlined above. In addition, all active BrAPI calls or simply calls, as well as, search request call logs are summarized in this interface. This page can be accessed using:

A. Drupal administration context menu /Tripal/Extensions/Tripal Web Service – BrAPI

Alternatively, copy and paste this URL into the browser’s location bar.

B . host/ admin/tripal/extension/tripalwsbrapi/configure

Sections are laid out in page tabs and are labelled to indicate which part of this module it covers. Each tab as seen in Figure 1, moving from left to right is summarized in the table below.

CONFIGURE TRIPAL WEB SERVICE BRAPI	Configure overall module operation such as menu levels, HTTP request methods to support and call override strategy.
BRAPI VERSION	Add BrAPI version(s) to support.
BRAPI CALLS	Configure call restrictions or filter conditions used by each call when performing queries.
SEARCH REQUEST LOG	A summary of all search request made. See BrAPI Search Services

1.3 Configure Tripal Web Service BrAPI Module

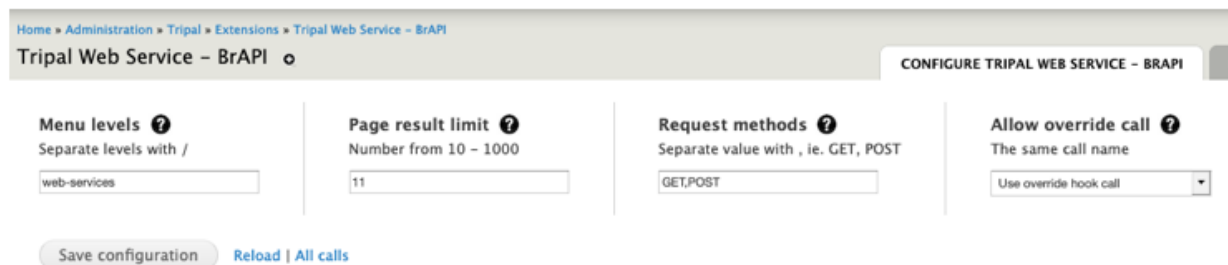


Figure 2 – Configuration page showing module settings.

Most system variables described can be accessed and modified using this form. Each field item (in all sections) can be described with a help or information text by hovering the mouse pointer on to help (question mark) icons. Click Save configuration button to save changes each time an option is modified.

Note: Clear cache each time when setting menu levels options.

Configuration	Use	Default Value
Menu Levels	An arbitrary levels may be added to request URL as outlined in BrAPI URL structure specifications. See BrAPI URL Structure	web-services
Page result limit	Limit the number of items/data per page returned by a BrAPI call request	100 items per page
Request methods	REST request methods supported.	GET and POST
Allow override call	In cases where an external module implements a call that is identical in name and version to an existing call, this option will decide which of the two call instances to apply.	no – use the local version of the call.

1.4 Configure Tripal Web Service BrAPI Calls

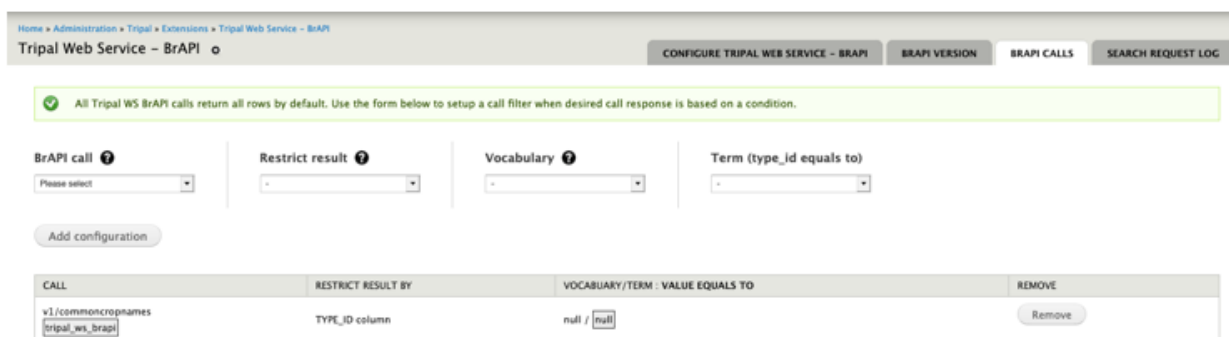


Figure 4 – Configuration page showing call settings.

Every call setup (see setting up calls) that performs a query to a CHADO database table can be configured using this form. This page enables system administrators to create additional restriction or filter criteria based on values stored in columns *CHADO.table – type_id column* and *CHADO.property table*. Each select field contains a summarized values from either of the table columns for quick and easy selections. A summary table below this form outlines all restrictions to a call as shown in Figure 4.

A row in the summary table can be interpreted as (from left to right column)

Note: Call, version X, hosted by Y module, titled ABC, restricts its results by type_id/value column, where its type_id value is of type W cv, equals to H cvterm.

To setup a database query filter/restriction to a call.

1. Select a BrAPI call from the select box.

Note: When a call does not involve querying of data from a database table, such as a custom call, a warning message will pop up instructing user that call cannot implement a query condition.

2. Each call can either use the column type_id or a property table. Restrict select field will analyze data stored and decide if it could support either option. Select an available option.
3. Once a restrict option has been selected, subsequent fields will auto-populate

with relevant values, once again based on values or records stored.

4. Select option when prompted.

Note: Another field labelled Value will present when restrict is set to property table.

5. Click Add configuration button to save.
6. When additional term is required, re-select the same call title. All select field elements will auto-fill with values that have been previously selected for easy and quick selection. Select additional values.
7. All configurations will be summarized in the summary table.

Note: Implement the call restriction created when setting up the call. See Setup Tripal WS Call.

1.5 Configure Tripal Web Service BrAPI Version

The screenshot shows the 'Tripal Web Service - BrAPI' configuration page. At the top, there are navigation tabs: 'CONFIGURE TRIPAL WEB SERVICE - BRAPI', 'BRAPI VERSION', 'BRAPI CALLS', and 'SEARCH REQUEST LOG'. Below the tabs, there are two dropdown menus: 'Major version' (with a help icon) and 'Minor version' (with a help icon). Below these is an 'Add version' button. A table below the form lists the configured versions:

MAJOR VERSION	MINOR VERSION(S)	DEFAULT	RESET	REMOVE
v1	1.3	1.3 <input type="button" value="Set default"/>	<input type="button" value="Reset"/>	<input type="button" value="Remove"/>
v2	2.0	2.0 <input type="button" value="Set default"/>	<input type="button" value="Reset"/>	<input type="button" value="Remove"/>

Figure 3 – Configuration page showing BrAPI version settings.

Most system variables described can be accessed and modified using this form. Each field item (in all sections) can be described with a help or information text by hovering the mouse pointer on to help (question mark) icons. Click Save configuration button to save changes each time an option is modified.

Warning: A version unsupported error message will be returned when attempting to request a call with undefined or not configured version number.

To support multiple versions of BrAPI in a single implementation, this page enables system administrators to plan and set additional version numbers. BrAPI only requires the major version number (leftmost digit also seen in request url – brapi/v1/..) when requesting a call, while calls can be versions 1.2 and/or 1.3 etc.

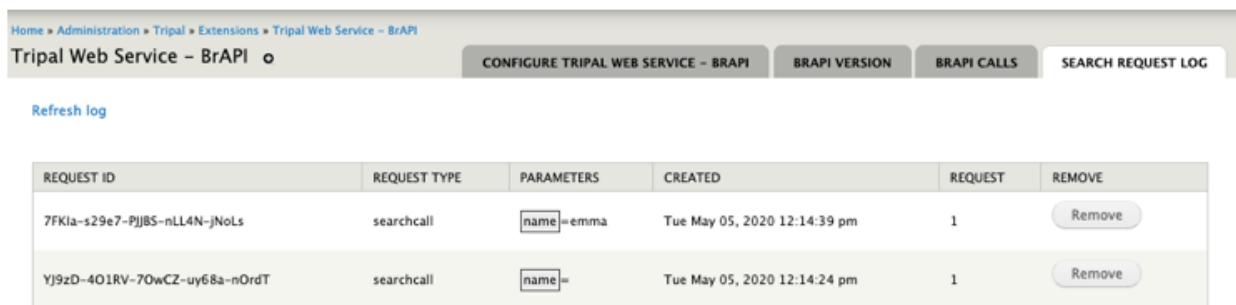
This version construct can be arranged using this page.

1. Select major version number from the list in Major version select field.
2. Select minor version number from the list in Minor version select field.
3. Click Add version button to save.
4. Each version added will be sorted and grouped according to major version number shown in the summary table below the form.

5. To add more version to a major version, re-select the major version number and select a minor version number then click Add version button. Minor version select field keeps track of what has been added, thus making sure no the same minor version number can be added more than once.
6. Additional version will be sorted accordingly as they are added to the group.
7. With multiple version, select a default, among the list of minor versions, to set as the default version of a major version number.
8. Use Reset button to drop all other versions except the default already set.
9. To remove version (including minor versions and default), click Remove button.

Warning: This module can only default to a single version at a given time per major version ie BrAPI 1.3. Other version such as BrAPI 1.2 can be implemented along side BrAPI 1.3, but requires switching from either versions as desired.

1.6 Configure Tripal Web Service BrAPI Search Calls Request Log



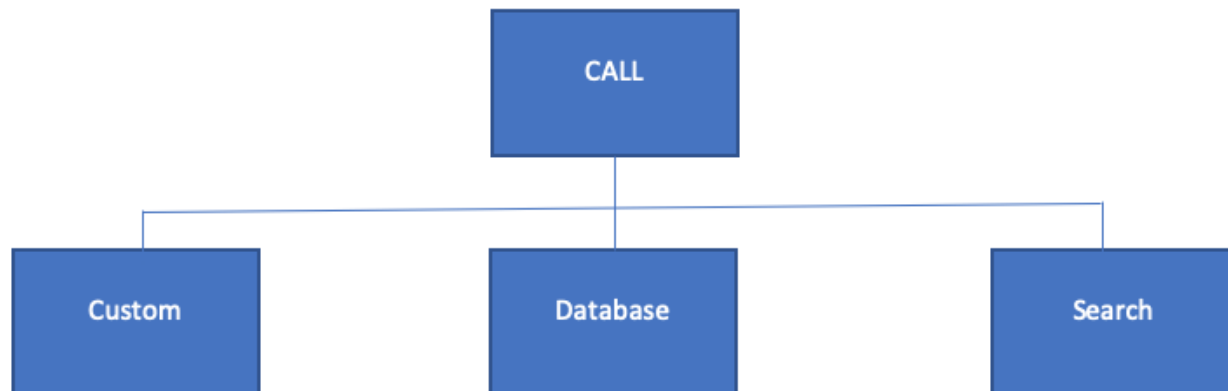
REQUEST ID	REQUEST TYPE	PARAMETERS	CREATED	REQUEST	REMOVE
7FKIa-s29e7-PjJB5-nLL4N-jNoLs	searchcall	<input type="text" value="name"/> =emma	Tue May 05, 2020 12:14:39 pm	1	<input type="button" value="Remove"/>
YJ9zD-4O1RV-7OwCZ-uy68a-nOrdT	searchcall	<input type="text" value="name"/> =	Tue May 05, 2020 12:14:24 pm	1	<input type="button" value="Remove"/>

Figure 5 – Configuration page showing search request log.

As described by BrAPI search services (<https://brapi.docs.apiary.io/#introduction/search-services>), all search call request using POST request method will be saved. This page summarizes all search call history, along with every call details, parameters, date created and number of times it has been requested. A Remove button is available to erase a log item permanently.

1.7 Tripal WS BrAPI Call Types

Tripal Web Service BrAPI Call Types: This module structures call into 3 types namely, **Custom call**, **Database call** and **Search call**.



Custom Call	Data source can be supplied by user either by typing a list or by defining an array of values. No database table involve.
Database Call	Data source comes from a Chado Table and can be configured to restrict result based on the type_id column or value column of the corresponding prop (property) table.
Search Call	Similar to Database call, search call operates like a database call except each request undergoes a two-stage process. First it will post a request that will result in a unique id number then the second stage, using the id number returned will produce the response. A log keeps a record of requests made (see Manage Search Request Log).

This modules has built-in calls namely v1/calls, v1/germplasm, v1/crops, v1/commoncropnames and search call v1/search/germplasm, all of these calls can be viewed by requesting using the BrAPI call url structure.

******For example: Host/web-services/brapi/v1/call or host/web-services/brapi/v2/serverinfo ******

Apart from these predefined calls mentioned, an external module (hosted by the same Drupal website) can implement a call or override a call without necessarily storing call assets in the same directory as this module, which will enable developers to extend functionality.

See Setup Tripal WS BrAPI Call.

1.8 Setup Tripal WS BrAPI Call

1.8.1 A. General information about calls: File structure

All calls (directory and files) must be saved in a calls/ directory of this module. This file structure also applies to external modules implementing a set of calls. TRIPAL_WS_BRAPI can support 2 file structures observed in BrAPI 1.3 and BrAPI 2.0, illustrated in the folder diagram below, labelled as A and B, respectively.

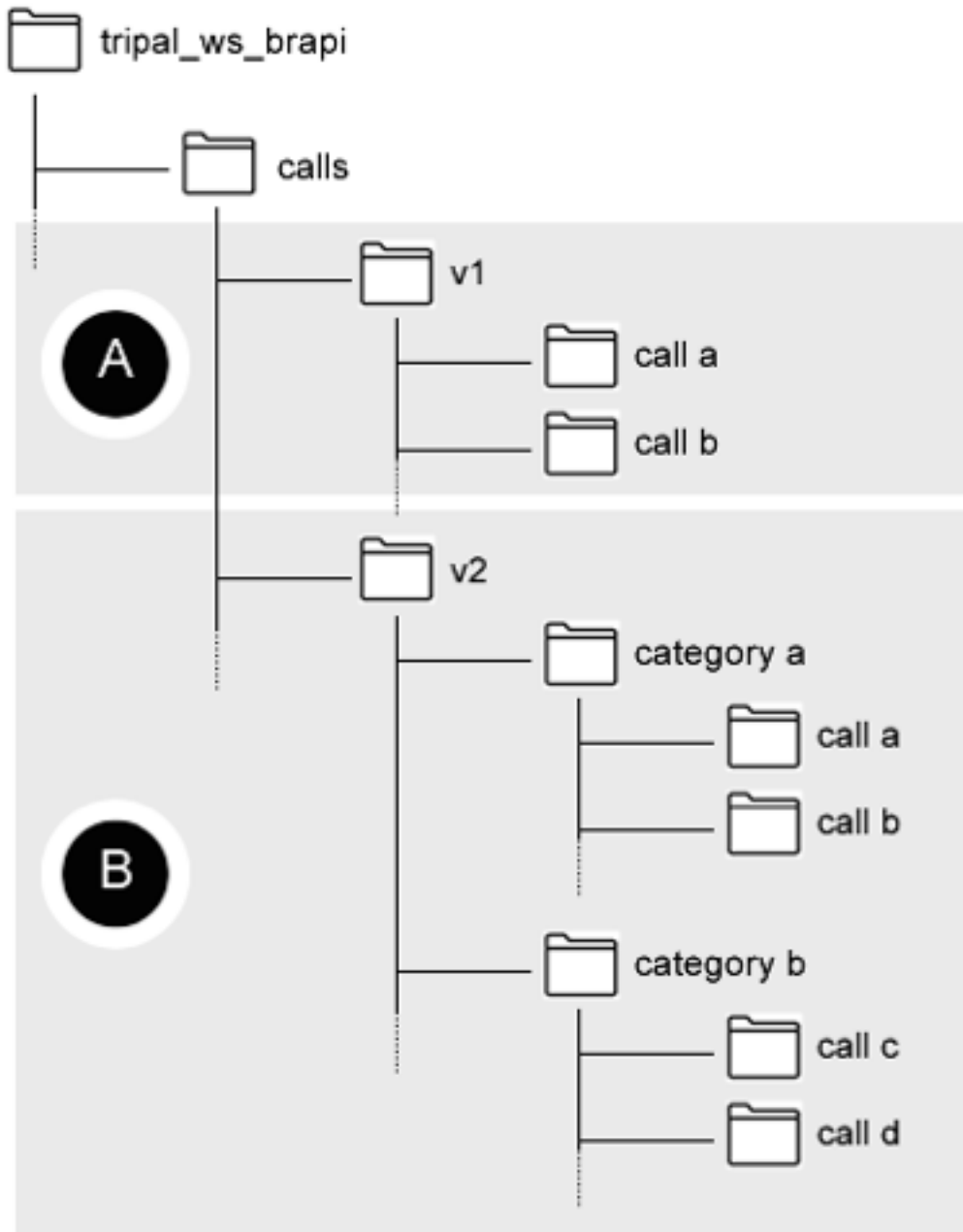


Figure 6 – Call directory structure supported. A: basic structure where calls fall directly under version folder and B: where calls are grouped using a category sub-folder.

Every call directory contains 1 or 2 PHP .inc files: **1. is a base call class file** and **2 search implementation class file**. #2 file is optional when a call does not require search functionality.

See figure below showing an example of a call, v1/germplasm call using file structure A or file structure B:

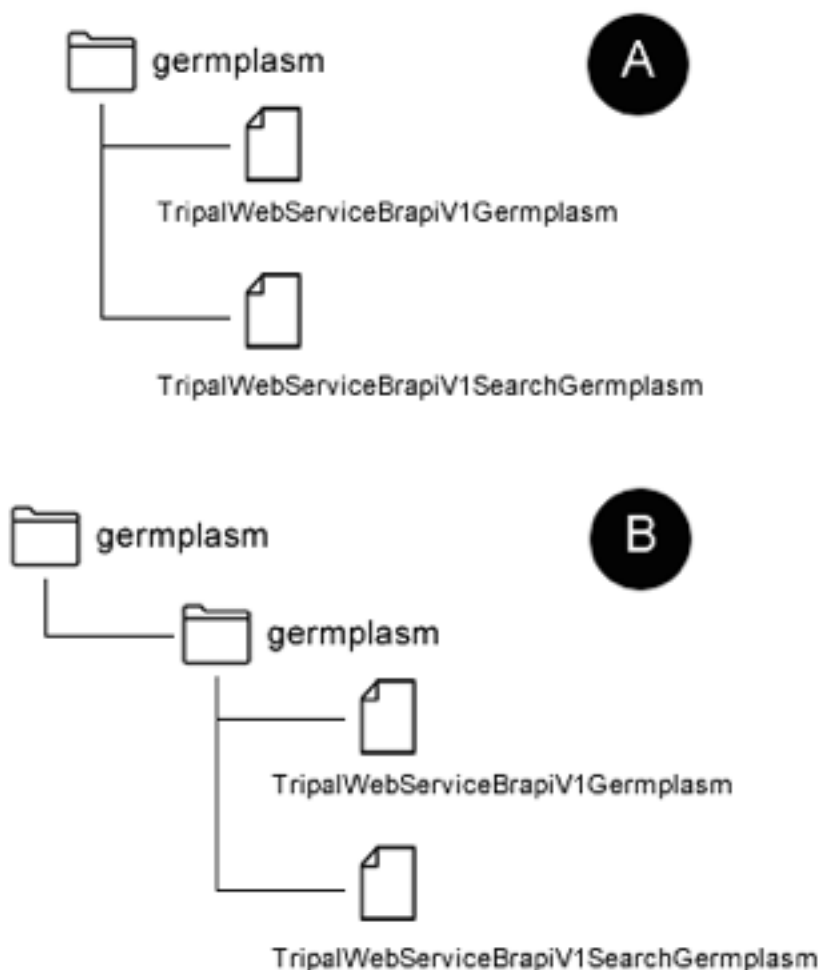


Figure 7 – Call file structures showing both directory structure options.

1.8.2 B. General information about calls: Naming file and class

Call name or title should be the identical call title defined by BrAPI specification. Directory name should match this title with all letters in lowercase form. Filename must be in the following format:

****TripalWebService + V + Major Version Number + Call name or title with the first letter in capitalized form****

****For example:** TripalWebServiceV1Germplasm.inc (to include the file extension which is .inc).**

A search implementation uses the same naming arrangement but has the keyword Search inserted between the version number and the call title.

****For example:** TripalWebServiceV1SearchGermplasm.inc (to include the file extension which is .inc).**

Both of these naming methods apply to when implementing a call that will be hosted outside the module with the exception that each name must be prefixed with the keyword **External**.

****For example:** ExternalTripalWebServiceV1Germplasm.inc or ExternalTripalWebServiceV1SearchGermplasm.inc**

Class name will match the filename created using this method.

1.8.3 C. General information about calls: External module implementing a hook must implement hook using hook string (see configuration)

To register an external module implementing a BrAPI call, use the hook string configuration and implement the following hook in the .module file:

```
/**
 * Implement BrAPI calls.
 */
function HOOK_tripal_ws_brapi_call() {
  // Indicate this module implements BrAPI calls.
  return TRUE;
}
```

Where the HOOK is the module name and tripal_ws_brapi_call is the hook string defined in the configuration. See Configuration.

1.9 Create Tripal WS BrAPI Custom Call

Creating a custom call, or other call types, is as easy as setting up parameters and defining a query or callback function that is responsible for generating relevant data as the response or result from executing a custom call.

Class definition

```
class TripalWebServiceBrapiV1Yourcallname extends TripalWebServiceCustomCall {
  // Parameters allowed in this call which can be included in the call as
  // query string, ie: host/brapi/v1/call?parameter 1=value&parameter 2=value...
  public $call_parameter = [
    // Key : Expected value for this key.
    'parameter 1' => 'data type',
    'parameter 2' => 'data type',
    'parameter ...' => 'data type',
  ];

  // Keyword used to identify result items.
  // data for most call in BrAPI v1.3 and may vary in newer version.
  protected $call_payload_key = 'data';

  // Unit of response for this call.
  // With corresponding minor version a response is for.
  // Fields must match item count in the result.

  // See restriction on multiple version in Configuration.
  public $response_field = [
    '1.3' => ['response field 1', 'response field 2', ' response field ...'],
    '1.2' => ['...'],
    '2.0' => ['...'],
  ];

  // Call parameters as provided in the request url.
  public $call_asset;
  // Class name.
}
```

(continues on next page)

(continued from previous page)

```
public $class_name;

// PREPARE RESULT:
// Callback to create data.
public function getResult() {
    // Define values matching the number of elements defined in $response field.
    return [];
}
}
```

1. Rename the class name using the format defined relating to call class filename (titled Yourcallname in the code snippet above).
2. List the parameters that user can apply to request specific items from the result. Each parameter can be set to a data type which will ensure that only appropriate entries are permitted.

int	(single value) numbers, including 0.
text	(single value) text, alphanumeric value.
array-int	(array, multiple values) elements are numbers.
array-text	(array, multiple values) elements are text value.
hash-code	(single value) xxxxx-xxxxx-xxxxx-xxxxx-xxxxx alphanumeric format.

3. Define the unit of data and its elements in **\$response_field**. Set the key to the target BrAPI version number. ie 1.3 or 1.2.
4. Set the **\$call_payload_key** to a string value. This variable will render as the key in the response. ie data (BrAPI 1.3) and call (BrAPI 2.0) used by /calls and /serverinfo calls, respectively.
5. Define the result in the only method of this class getResult().

Note:

Ensure that the number of items and the order of data returned by getResult() should match the items in the \$response_filed. Include a mechanism to handle each parameters defined in #2. Parameters requested in the url are available in getResult() through the property **\$call_asset**.

```
$this->call_asset['parameters'] property and
$this->call_asset['parameters']['parameter 1'],
$this->call_asset['parameters']['parameter 2'],
$this->call_asset['parameters']['parameter ...'] to access the value.
```

6. Save the file.
7. Test your call using host/web-services/brapi/v + version/yourcallname.
8. Test your call with the parameters set using host/web-services/brapi/v + version/yourcallname?parameter 1=value¶meter 2=value...

1.10 Create Tripal WS BrAPI Search Call

Similar to creating a custom call and database call, a search call uses both **GET** and **POST** request methods to view a query result and request a query, respectively.

Class definition

```

class TripalWebServiceBrapiV1SearchYourcallname extends TripalWebServiceSearchCall {
  // Parameters allowed in this call which can be included in the call as
  // query string. See example below using CURLOPT_POSTFIELDS option.
  public $call_parameter = [
    // Key : Expected value for this key.
    'parameter 1' => 'data type',
    'parameter 2' => 'data type',
    'parameter ...' => 'data type',
  ];

  // Keyword used to identify result items.
  protected $call_payload_key = 'data';

  // Unit of response for this call.
  // With corresponding minor version a response is for.
  // Fields must match item count in the result.

  // See restriction on multiple version in Configuration.
  public $response_field = [
    '1.3' => ['response field 1', 'response field 2', ' response field ...'],
    '1.2' => ['...'],
    '2.0' => ['...'],
  ];

  // Chado table, source data.
  public static $chado_table = 'chado.table ie stock'

  // Call parameters as provided in the request url.
  public $call_asset;
  // Class name.
  public $class_name;

  // PREPARE RESULT:
  // Callback to create data.
  public function getResult() {
    // Define values matching the number of elements defined in $response field.
    return [];
  }
}

```

1. Rename the class name using the format defined relating to call class filename (titled Yourcallname in the code snippet above).
2. List the parameters that user can apply to request specific items from the result. Each parameter can be set to a data type which will ensure that only appropriate entries are permitted.

int	(single value) numbers, including 0.
text	(single value) text, alphanumeric value.
array-int	(array, multiple values) elements are numbers.
array-text	(array, multiple values) elements are text value.
hash-code	(single value) xxxxx-xxxxx-xxxxx-xxxxx-xxxxx alphanumeric format.

3. Define the unit of data and its elements in \$response_field. Set the key to the target BrAPI version number. ie 1.3 or 1.2.
4. Set the \$call_payload_key to a string value. This variable will render as the key in the response. ie data (BrAPI 1.3) and call (BrAPI 2.0) used by /calls and /serverinfo calls, respectively.

5. Define the result in the only method of this class getResult().

Note:

Ensure that the number of items and the order of data returned by getResult() should match the items in the \$response_filed. Include a mechanism to handle each parameters defined in #2. Parameters requested in the url are available in getResult() through the property \$call_asset.

```
$this->call_asset['parameters'] property and
$this->call_asset['parameters']['parameter 1'],
$this->call_asset['parameters']['parameter 2'],
$this->call_asset['parameters']['parameter ...'] to access the value.
```

- 6. Save the file.
- 7. Test your call using host/web-services/brapi/v + version/yourcallname.
- 8. Test your call with the parameters set using host/web-services/brapi/v + version/yourcallname?parameter 1=value¶meter 2=value...

Note: This class extends to a different class than the one used in defining database calls and custom calls and it is important to specify the source table (\$chado_table property). Class name now contains a Search keyword as described in naming class section. The class this class extends to handles both POST (log search request) and GET requests.

Search call operates differently compared to other calls – custom call and database call. Search call needs to POST the call (with parameters) and at this stage a hash code is returned. A call can then be requested using the code to view the result or response.

POST Search Request	GET Search Request
<pre>Sample request. ... code-block:: php \$ch = curl_init(); // Search call curl_setopt(\$ch, CURLOPT_URL, "host/web- services/brapi/v1/search/searchcall"); curl_setopt(\$ch, CURLOPT- LOPT_RETURNTRANSFER, TRUE); curl_setopt(\$ch, CURLOPT_HEADER, FALSE); curl_setopt(\$ch, CUR- LOPT_POST, TRUE); // Parameter curl_setopt(\$ch, CUR- LOPT_POSTFIELDS, [{"parameter" : ["value"]}]); curl_setopt(\$ch, CURLOPT_HTTPHEADER, ["Content- Type: application/json"]); \$response = curl_exec(\$ch); curl_close(\$ch); var_dump(\$response);</pre>	<pre>host/web-services/brapi/v1/search/searchcall?searchResultDbId=7FKIa- s29e7-PJJBS-nLL4N-jNoLs</pre>
Add parameters in // Parameter line. Parameter in JSON format.	Using the hash code returned, get the call response by adding a the parameter searchResultDbId=HASH CODE to the call.
RESPONSE: hash code 7FKIa-s29e7-PJJBS-nLL4N-jNoLs	Call response JSON.

A copy of the POST request and the hash code can be accessed in the configuration page. To perform the same search request, use the same hash code to GET request call to retrieve the same response. This call request and its parameters can be accessed multiple times so long as the log entry is not deleted.

1.11 Create Tripal WS BrAPI Override Call or Call Alias

In some cases, calls from different versions implement the same process carried out by an existing call. To eliminate the need to copy and paste codes, call can point to an existing call and function identical to the source call.

Class definition

```
class TripalWebServiceBrapiV2Serverinfo extends TripalWebServiceCustomCall {
  // Call parameters as provided in the request url.
  public $call_asset;

  // Override the base class payload key.

  // Keyword used to identify result items.
  // data for most call in BrAPI v1.3 and may vary in newer version.
  protected $call_payload_key = 'call';

  // Call existing and identical call already setup.
  // Declare existing call.
  //
  // module name / char v + major version number / call name.
  //
  // See table below for the actual alias of existing calls in this module.
  public static $is_alias_of = 'tripal_ws_brapi/v1/calls';
}
```

1. To override the \$call_payload_key of the source call, set the value of the property **\$call_payload_key**.
2. Extend the class identical to the type of call of the source call.
3. Finally, set the source call by specifying the module it is hosted followed by the version number then the call title. For example, the call – mygermplasm that mimics germplasm call, located in tripal_ws_brapi module:

tripal_ws_brapi/v1/germplasm

In cases when call wants to point to a search call, add search/ level between version number and the call title.

tripal_ws_brapi/v1/search/germplasm

Note: Ensure correct override configuration settings when implementing an override, by using the exact same call title but is hosted by module external to tripal_ws_module.

Note: When implementing from an external module, prefix the class and class filename with **External** keyword (see naming class section).

For example:

```
class ExternalTripalWebServiceBrapiV2Serverinfo extends TripalWebServiceCustomCall
```

Summary table of existing Tripal WS BrAPI Calls when implementing a call Alias and/or Override:

tripal_ws_brapi Calls	Call alias property - \$is_alias_of:
v1/calls	tripal_ws_brapi/v1/calls
v1/commoncropnames	tripal_ws_brapi/v1/commoncropnames
v1/crops	tripal_ws_brapi/v1/crops
v1/germplasm	tripal_ws_brapi/v1/germplasm
v1/search/germplasm	tripal_ws_brapi/v1/search/germplasm
Other calls	TO DO.

1.12 Contributing

We're excited to work with you! Post in the issues queue with any questions, feature requests, or proposals.

1.12.1 Quickly setting up a Testing/Demo Environment using Docker

The following commands clone this repository and then start a Drupal7Docker container with the the current directory mounted. It also initializes the Tripal site by starting postgresql, installing Tripal and Chado, and preparing the database.

```
git clone https://github.com/UofS-Pulse-Binfo/tripal_ws_brapi.git
cd tripal_ws_brapi
docker pull laceysanderson/drupal7dev
docker run --publish=8888:80 --name=tdocker -tid --env-file=tests/example.env --
↳volume=`pwd`: /var/www/html/sites/all/modules/tripal_ws_brapi laceysanderson/
↳drupal7dev:latest
docker exec -it tdocker /app/init_scripts/startup_container.sh
```

Next we need to install this module and add test data to the site. The following commands accomplish this given the above setup.

```
docker exec tdocker /var/www/html/vendor/bin/drush en tripal_ws_brapi tripal_ws_brapi_
↳testdata -y
docker exec --workdir=/var/www/html tdocker ./vendor/bin/drush php-script sites/all/
↳modules/tripal_ws_brapi/tripal_ws_brapi_testdata/drush-scripts/loadTestData.php
```

Now you can proceed by running the automated tests:

```
docker exec --workdir=/var/www/html/sites/all/modules/tripal_ws_brapi tdocker_
↳composer up
docker exec --workdir=/var/www/html/sites/all/modules/tripal_ws_brapi tdocker vendor/
↳bin/phpunit
```

Or manually testing it through <http://localhost:8888/web-services>.

1.12.2 Automated Testing

This module uses Tripal Test Suite. To run tests locally:

```
cd MODULE_ROOT
composer up
./vendor/bin/phpunit
```

This will run all tests associated with the Tripal WS BrAPI extension module. If you are running into issues, this is a good way to rule out a system incompatibility.

Warning: It is highly suggested you **ONLY RUN TESTS ON DEVELOPMENT SITES**. We have done our best to ensure that our tests clean up after themselves; however, we do not guarantee there will be no changes to your database.

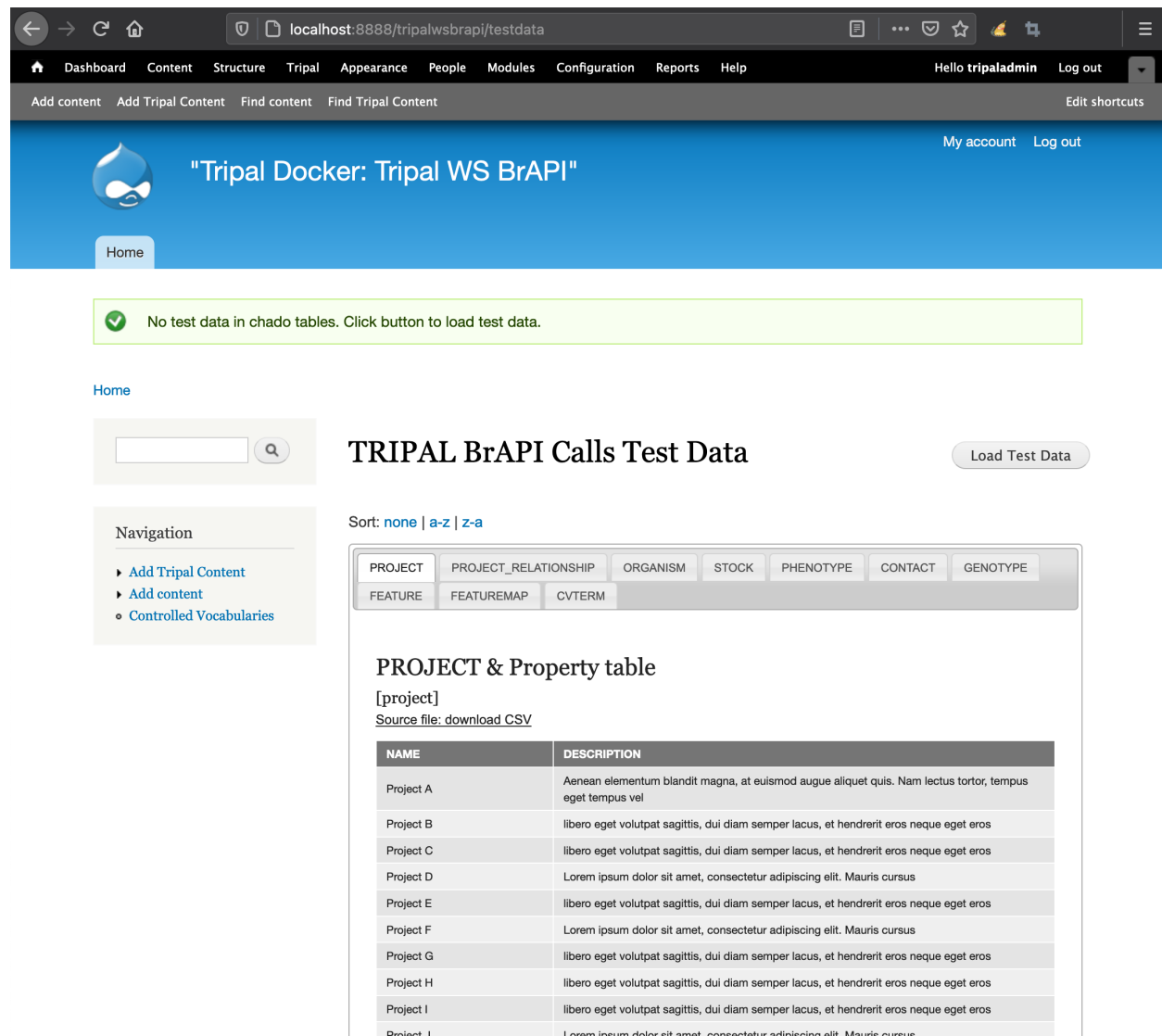
1.12.3 Manual Testing (Demonstration)

Using the Testing Helper Module

First enable the helper module through the UI or using Drush:

```
drush en tripal_ws_brapi_testdata
```

Then go to the provided user interface at [https://\[{}yourdrupalsite{}\]/tripalwsbrapi/testdata](https://[{}yourdrupalsite{}]/tripalwsbrapi/testdata)



Home

My account Log out

✔ No test data in chado tables. Click button to load test data.

Home

TRIPAL BrAPI Calls Test Data Load Test Data

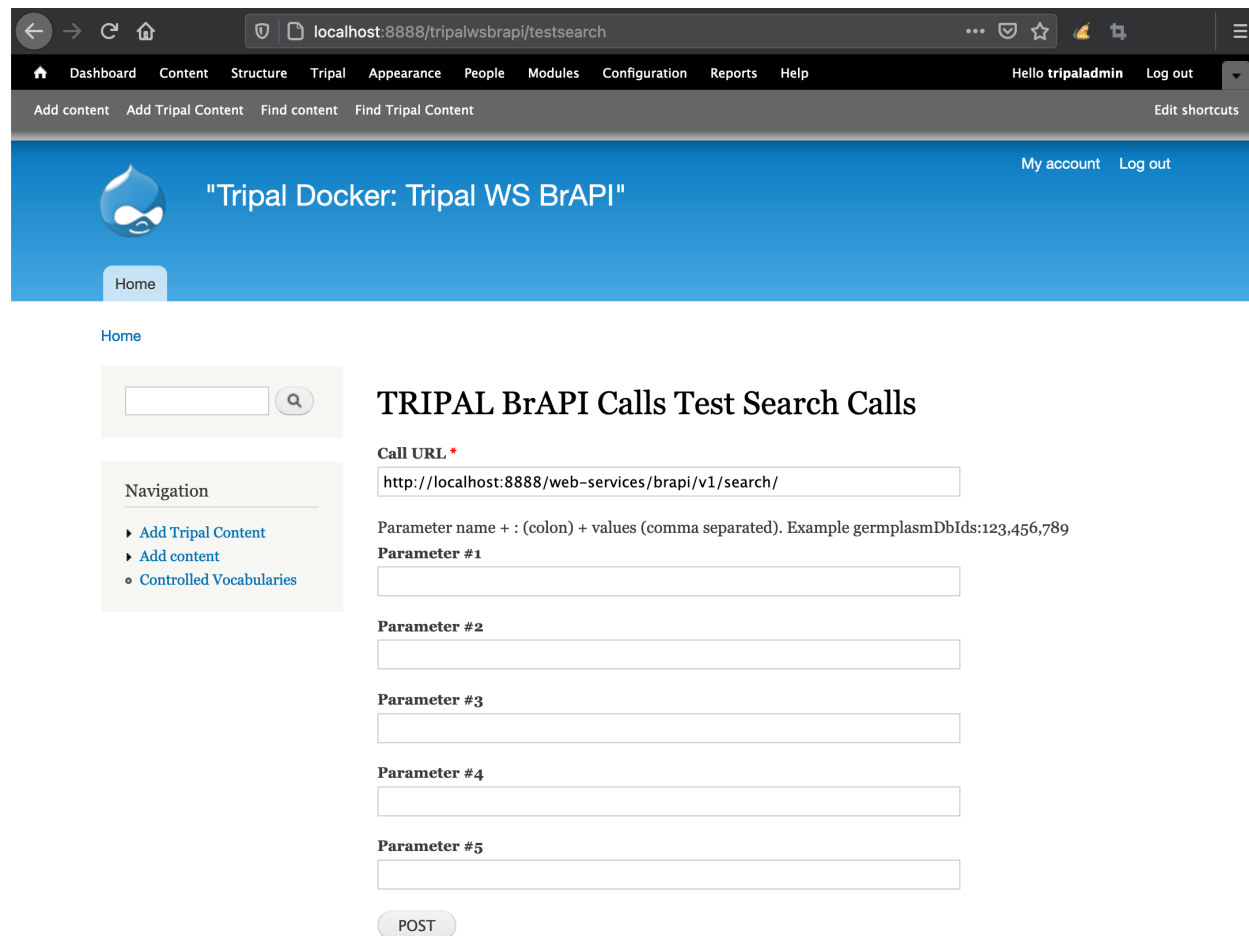
Sort: none | a-z | z-a

PROJECT	PROJECT_RELATIONSHIP	ORGANISM	STOCK	PHENOTYPE	CONTACT	GENOTYPE
FEATURE	FEATUREMAP	CVTERM				
PROJECT & Property table						
[project]						
Source file: download CSV						
NAME	DESCRIPTION					
Project A	Aenean elementum blandit magna, at euismod augue aliquet quis. Nam lectus tortor, tempus eget tempus vel					
Project B	libero eget volutpat sagittis, dui diam semper lacus, et hendrerit eros neque eget eros					
Project C	libero eget volutpat sagittis, dui diam semper lacus, et hendrerit eros neque eget eros					
Project D	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris cursus					
Project E	libero eget volutpat sagittis, dui diam semper lacus, et hendrerit eros neque eget eros					
Project F	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris cursus					
Project G	libero eget volutpat sagittis, dui diam semper lacus, et hendrerit eros neque eget eros					
Project H	libero eget volutpat sagittis, dui diam semper lacus, et hendrerit eros neque eget eros					
Project I	libero eget volutpat sagittis, dui diam semper lacus, et hendrerit eros neque eget eros					
Project J	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Mauris cursus					

The tabbed table shows you all the test data which will be loaded into your site when you click the “Load Test Data” button.

You can edit the test data by editing the CSV files distributed with this helper module. More tables can be added by editing the array of chado table names in the tripal_ws_brapi_testdata.module file.

There is also an interface provided for testing POST calls. This is available at [https://\[yourdrupalsite\]/tripalwsbrapi/testsearch](https://[yourdrupalsite]/tripalwsbrapi/testsearch)



Using the Database Seeder

We have provided a [Tripal Test Suite Database Seeder](#) to make development and demonstration of functionality easier. To populate your development database with fake germplasm data:

1. Install this module according to the instructions in this guide.
2. Run the database seeder to populate the database using the following commands:

```
cd MODULE_ROOT
composer up
./vendor/bin/tripaltest db:seed BrAPIDatabaseSeeder
```

4. To access the web services go to [https://\[your drupal site\]/web-services](https://[your drupal site]/web-services) assuming the default configuration.

Warning: NEVER run database seeders on production sites. They will insert fictitious data into Chado.