Tripal BrAPI Documentation

Valentin Guignon, Bioversity International

May 24, 2022

Contents:

This is an implementation of the Breeding API which uses Tripal to expose data in your Chado database.

The Breeding API specifies a standard interface for plant phenotype/genotype databases to serve their data to crop breeding applications. It is a shared, open API, to be used by all data providers and data consumers who wish to participate. Initiated in May 2014, it is currently being actively developed, so now is the time for potential participants to help shape the specifications to ensure their needs are addressed. The listserve for discussions and announcements is at Cornell University. Additional documentation is in the Github wiki. The latest up-to-date specifications and discussions can be found on the git repository and the issue queue.

CHAPTER 1

Usage

Install the module, enable it and adjust the CV settings according to your Chado database instance and the way you store your biological data. You may also setup germplasm attribute settings, add some external BrAPI site references and setup call aggregation if needed.

You can query the BrAPI service through the URL

```
http(s)://<your-drupal-site>/brapi/v1/<service-name>?<query-parameters>
```

For example, `https://www.crop-diversity.org/mgis/brapi/v1/calls`

It will return a JSON structure that can be processed by BrAPI-compliant applications.

1.1 Testing your Instance

To test your instance, you can either use the RestClient plugin for your favorite web browser client from http://restclient.net/ or use the provided BrAPI query interface (path brapi/query). Note that when you use this interface as admin, you will have an additional checkbox which can enable debug mode.

1.2 Javascript & dynamic HTML

If you want to add BrAPI fields on your pages that should be automatically and dynamically populated using external BrAPI site, you can use the following HTML snippet:

```
<form class="brapi-autoquery"
  action="https://BRAPI_SERVER/brapi/v1/SERVICE?PARAMETERS..." method="GET">
  <input type="hidden" name="brapi_html" value="URL_ENCODED_HTML_STRING"/>
  <input type="submit" name="submit" value="Get BrAPI data"/>
</form>
```

where "BRAPI_SERVER" is the BrAPI server name, "SERVICE?PARAMETERS…" is the BrAPI service to query with its optional parameters and values and "URL_ENCODED_HTML_STRING" is the URL-encoded HTML code

to use to replace the form. In this string, not encoded place-holder string will be replaced by properties of the (first) JSON object returned. A place-holder is a the property name as described in the BrAPI specs inside square-brackets. For instance "[germplasmName]" (for the "germplasm-search" call) will be replace by the germplasm name of the first germplasm returned by the call.

Note: Array or object properties can not be used here.

The form can contain additional call parameters using hidden input or select fields wrapped by an HTML element having the CSS class "brapi-query-filter-post"

CHAPTER 2

Set-up

This branch of the module works with both Tripal 2 and Tripal 3.

2.1 Installation

This guide assumes you have a working Tripal site. **Tripal Core is required**.

1. Install as you would normally install a contributed Drupal module. See: https://drupal.org/documentation/install/modules-themes/modules-7 for further information.
2. Enable the module in "Admin menu > Site building > Modules" (path /admin/modules).

Note: This module adds the **Multi-Crop Passport Descriptor (MCPD) Controlled Vocabulary** to your Tripal site. An up-to-date version is required.

Warning: In case you have an obsolete version of the MCPD vocabulary, you may need to update it using the button "Reload Chado MCPD CV" in the MCPD Settings section of admin/tripal/extension/brapi/configuration page.

2.2 Configuration

Configure Tripal-BrAPI in "Administration > Tripal > Extensions > Breeding API > Settings" (path /admin/tripal/extension/brapi/configuration). The settings are organized by sections:

2.2.1 Example value settings

You can specify here which identifiers should be used to demo the calls on the overview page (path brapi/overview). The identifiers correspond to the names inside curly brackets used in call URLs. These settings are optional and do not impact BrAPI behavior.

2.2.2 Storage options

This is the place where you specify how you store your data in Chado and how BrAPI can find it. For instance, if you don't store the common crop name of your stock in the stockprop table, BrAPI can use the organism table instead. In this case, you will change the "Common crop name storage" settings from "Stored in stockprop table" to "Stored in organism table". Then Tripal-BrAPI module will find the common crop name using stock.organism_id –> organism table –> organism.common_name. However, if you do use the stockprop table, then you must ensure you also setup the appropriate cvterm_id for "commonCropName" in the MCPD settings.

All the other parameters have a similar approaches. The "Date storage format" specifies the way your dates are stored and not the way BrAPI will display them (which is in the specifications). As dates are stored as strings in the value field of property, they can be stored in a human-readable manner as well as in timestamp format.

2.2.3 Controlled vocabulary settings

This is where you associate BrAPI terms used by BrAPI calls to corresponding CV terms available in Chado and used by the corresponding field ([table]prop.type_id or [table]_cvterm.cvterm_id). Use the auto-completion feature to find the corresponding terms from the appropriate Controlled Vocabulary (CV). Make sure the numeric value of the identifier of each cvterm, `(cvterm_id)`, is present; the term name can be omitted. In some cases, you may specify several CV terms by separating them with comma.

2.2.4 MCPD settings

BrAPI uses MCPD (Multi-Crop Passport Descriptor controlled vocabulary). If you use your own terms rather than the MCPD ones, you can override these settings here the same way you did for "Controlled vocabulary settings".

2.2.5 Germplasm attribute settings

These settings hold attribute categories and germplasm attributes stored in your Chado database. Category names are case sensitive: make sure you always use the same case. For each attribute you define, you can specify either the Chado Controlled Vocabulary (cv_id) that holds all the terms used by the attribute or a list of Controlled Vocabulary terms (cvterm_id) used. The BrAPI system will look for attributes either in the stockprop table if they have a type_id corresponding to the selected Controlled Vocabulary terms and in stock_cvterm table if they have a corresponding cvterm_id. For stockprop, the value field will be returned as the attribute value while for the stock_cvterm table, the Controlled Vocabulary term name (cvterm.name) will be returned as the value. Only the attributes that you define here will be exposed by BrAPI.

> **Warning:** Category names are case sensitive: make sure you always use the same case.

2.2.6 Call aggregation options

This is where you can setup call aggregation or proxy-ing.

- "Call aggregation" is a way to complete missing fields (null) returned by a given local BrAPI call with field values provided by the same call run on other BrAPI instances.
- Proxy-ing is relaying a BrAPI call to another BrAPI instance and serve its result to the client without having him/her to know about the other BrAPI instance: the client only sees one BrAPI en point which make his/her life easier.

With this Tripal-BrAPI module, you can select which call you wish to aggregate with which other BrAPI instances. If you don't provide any data for a given call which is aggregated, all the field values from the other instance will be used, which is basically "proxy-ing" the call. In order to use aggregation, you need first to record external BrAPI sites in the system. To do so, you will have to go to the "Content > BrAPI Site References" page (path admin/content/brapi_site) and add new references. Then you can go back to the BrAPI setting page and will be able to select one of your referenced BrAPI sites for the call you would like to aggregate or proxy.

2.2.7 Permissions

Configure user permissions in "Administration > People > Permissions" (/admin/people/permissions):

- "Use Breeding API": allows users to access to the Breeding API. Roles having this permission can not alter data but have read access to all the data available through the Breeding API.
- "Update through Breeding API": allows users to modify database content. Roles with this permission can add new entries and update or remove exiting ones.
- "Administer Breeding API": allows users to change the Breeding API settings such as the CV term uses and the default entries to use as example.

2.3 Authentication

The Breeding API Drupal module relies on Drupal user management system to manage its users. Users can authenticate either through the breeding API (https://<your-drupal-site>/brapi/v1/token POST call) or though the Drupal interface (/user). The Drupal user management interface can be used to add or remove users. Users can update their account properties (name, e-mail,...) using Drupal user interface as well. User access levels are managed using Drupal roles and permission system (see "Configuration" paragraph).

Authentication process and calls that require authentication MUST go through the HTTPS protocol (ie HTTP Secure through SSL or TLS encryption). The authentication token must be provided with each BrAPI call that requires authentication (but it can be omitted in other cases). It should[*] be provided by the client application through the HTTP header field "Authorization" and must be of the form:

Authorization: Bearer SESS<some codes>=<some other codes>

Note that "SESS<some codes>" part correspond to the Drupal user session cookie name and "<some other codes>" correspond to the cookie value. [*]: as the module relies on Drupal user management system, the token can be replaced by Drupal regular session cookie. BrAPI client applications that support cookies but not authentication token will support (non-standard) authentication with this BrAPI implementation.

2.4 Troubleshooting

Note: Please note that this module supports a limited number of fields that are part of the Breeding API specifications.

If the Breeding API does not display all the data you expect for a given entry, make sure you associated the appropriate Chado Controlled Vocabulary term with the fields managed by this module. Your data should also be stored in Chado the way the Breeding API Drupal module expects it to be. If you store things differently, consider also storing them the way this module expects them to be stored.

CHAPTER 3

Developers Guide

This guide is meant to help developers extend Tripal BrAPI and/or use BrAPI web services to in their own module.

3.1 Process Overview

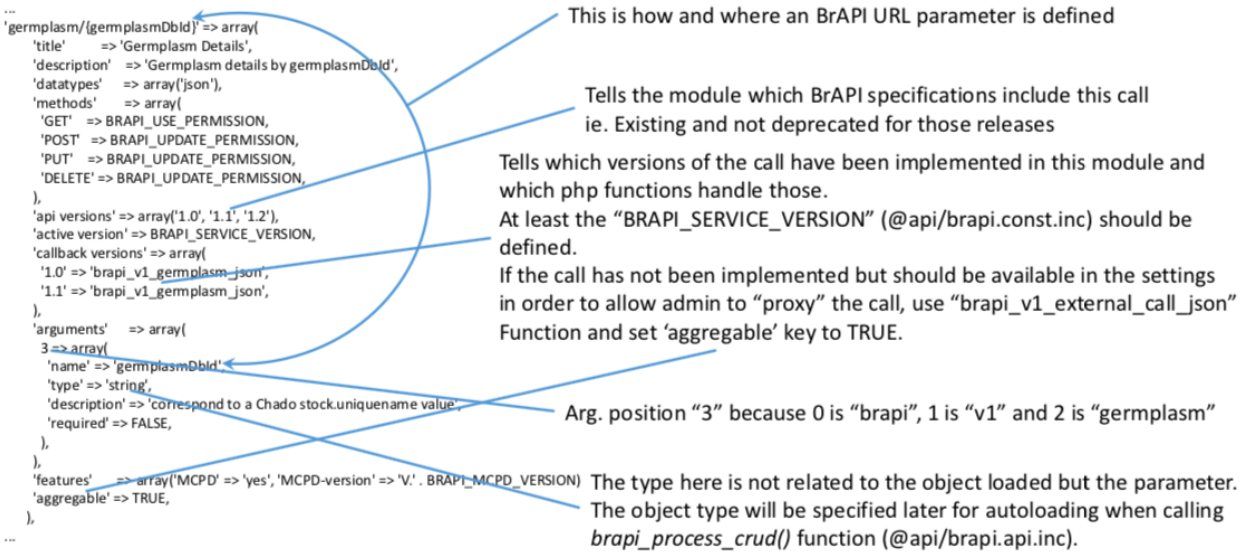
This section attempts to explain how Tripal-BrAPI handles calls.

3.1.1 How BrAPI calls are handled

Example: ‘‘https://www.crop-diversity.org/mgis/brapi/v1/germplasm/01BEL084609‘‘ where `https://www.crop-diversity.org/mgis` is the Drupal site root.

1. Drupal hook_menu called `brapi_menu()` (in `brapi.module`) to resolve URL handling. `brapi_menu()` provides to Drupal supported BrAPI calls as they are defined in `brapi_get_calls()` (in `api/brapi.const.inc`).

Here, the system will select the call defined by the `germplasm/{GermplasmDbId}` key of the array returned by brapi_get_calls():



Note: Any call setting defined by `brapi_get_calls()` can be overridden by external module using the hook `hook_brapi_calls_alter(&$brapi_calls)`, `$brapi_calls` being the array returned by `brapi_get_calls()`. Call versions are selected on the BrAPI setting administration interface.

2. For every BrAPI call, Drupal will call `brapi_call_wrapper()` (in `api/brapi.calls.inc`) that will then call the function specified by `brapi_get_calls()` at the `callback versions` key. `brapi_call_wrapper()` will check permissions and select the appropriate call version.

Here, we assume current user has `BRAPI_USE_PERMISSION` and used GET HTTP method: `brapi_v1_germplasm_json()` (in `api/brapi.calls.inc`) function will be called.

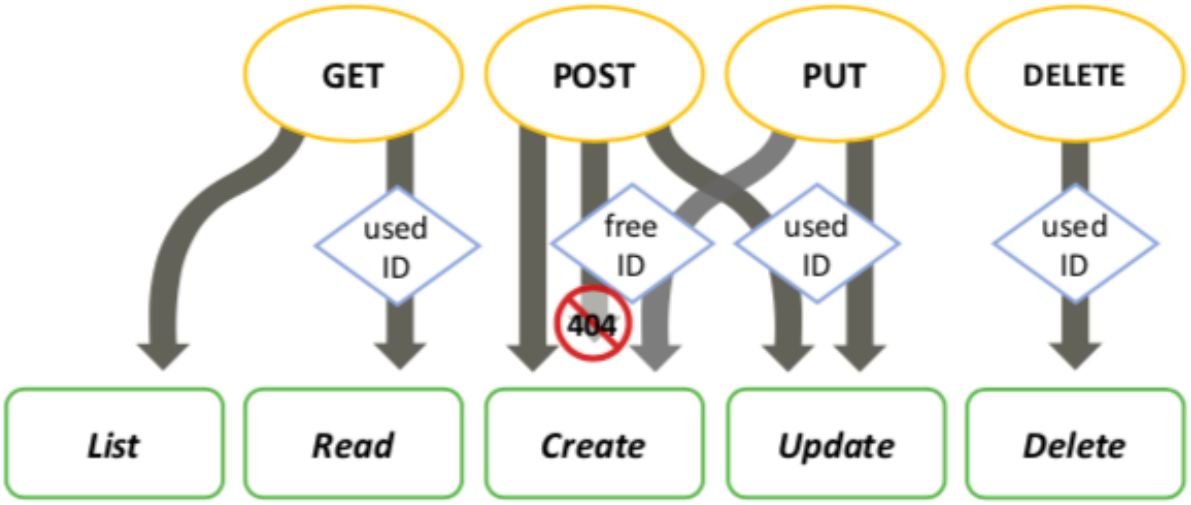
3. Then `brapi_call_wrapper()` will allow external modules to override returned data before being served to the client if they implement `hook_brapi_CALL_FUNC_NAME_alter(&$data, &$context)`. Here, any implementation of `hook_brapi_germplasm_germplasmDbId_alter(&$data, &$context)` will be called. Note, it is also possible to alter call result in case of errors.

4. Finally, `brapi_json_prepare()` and `brapi_json_output()` (in `api/brapi.api.inc`) will be called to format data into a JSON string that will be served by Drupal to the client.

3.1.2 How `brapi_v1_germplasm_json()` works

1. `brapi_call_wrapper()` calls `brapi_v1_germplasm_json()` (in `api/brapi.calls.inc`). As the "germplasm" call deals with a germplasm "resource" (as defined in RESTFull architecture, or "object", or "entity", or whatever you want to call it), it will call `brapi_process_crud()` (in `api/brapi.api.inc`). CRUD stands for Create-Read-Update-Delete and is part of the RESTFull architecture.

2. `brapi_process_crud()` will decide, according to the context (HTTP method used, identifier provided), which action should be done:



`brapi_process_crud()` will load the corresponding resource in case of a used ID provided and will then call the function corresponding to the action to perform. Action functions are provided by the `$action` parameter:

```
$actions = array(
  'create' => 'brapi_v1_create_germplasm_json',
  'read' => 'brapi_v1_read_germplasm_json',
  'update' => 'brapi_v1_update_germplasm_json',
  'delete' => 'brapi_v1_delete_germplasm_json',
  'list' => 'brapi_v1_germplasm_search_json',
);
```

Here, `brapi_v1_read_germplasm_json()` (in `api/brapi.calls.inc`) will be called with a "germplasm" resource of ID `01BEL084609` loaded (using `chado_generate_var()`) as argument.

3. `brapi_v1_read_germplasm_json()` (in `api/brapi.calls.inc`) will prepare and return the data structure that will be converted later into JSON:

```
function brapi_v1_read_germplasm_json($stock) {

  // It initializes the metadata part of the BrAPI call answer.
  // Here we only have 1 element, so we set the pager to 1 element.
  $metadata = brapi_prepare_metadata(1);

  // This can be used to output debugging information into the call.
  $debug_data = array();

  // See step 4 below.
  $germplasm_data = brapi_get_germplasm_details($stock);

  // It initializes the data part of the BrAPI call answer.
  $data = array('result' => $germplasm_data);

  // See step 5 below.
  brapi_aggregate_call($data, $metadata, $debug_data);

  // return the results.
  return array(
    $data,
    $metadata,
```

(continues on next page)

(continued from previous page)

```
    $debug_data,
  );
}
```

4. `brapi_get_germplasm_details()` (in `api/brapi.calls.inc`) is called to generate and fill the structure of a germplasm as defined in the BrAPI specifications. This function is also used when listing germplasm ("germplasm" call without ID) and by the "germplasm-search" call.
5. Then `brapi_aggregate_call()` (in `api/brapi.api.inc`) will check if current BrAPI settings require to also call this BrAPI call on external servers and if so, calls will be made and data will be aggregated (ie. Fields with "null" value returned by `brapi_v1_germplasm_json()` will be filled when a non-null values are provided by external calls for those fields).

3.2 Hook Overview

- `hook_brapi_cv_settings_alter(&settings)`: allows other module to alter CV settings. See also `brapi_get_cv_settings()` documentation in `api/brapi.api.inc`.
- `hook_brapi_data_mapping_alter(&$brapi_data_mapping)`: allows other module to alter BrAPI data mapping settings. See also `brapi_get_data_mapping()` documentation in `api/brapi.const.inc`.
- `hook_brapi_object_selector_alter(&$selector, $context)`: allows other module to alter Chado object loaded by BrAPI when a DbId has been specified for a given call.
- `hook_brapi_calls_alter(&brapi_calls)`: allows other module to alter BrAPI call settings. While you can replace call settings and especially the callback function, you should better use `hook_brapi_CALL_FUNC_NAME_alter` hook to replace BrAPI calls in order to allow other modules to alter the call answer if they need to as well. This hook should be used to change call version settings or supported methods or datatypes or aggregation option only. See also `brapi_get_calls()` documentation in `api/brapi.const.inc`.
- `hook_brapi_CALL_FUNC_NAME_alter(&$data, &$context)`: allows other module to alter the result of a BrAPI call. `$data` contains the result structure currently returned by the call and `$context` contains the metadata and debug strings. See also `brapi_call_wrapper()` documentation in `api/brapi.calls.inc`.
- `hook_brapi_CALL_FUNC_NAME_brapi_error_alter(&$output)`: allows other module to alter error raised by BrAPI (BrAPI exceptions associated to specific HTTP error codes (like 400 bad request, 404 not found, 501 not implemented and such). It is also possible to replace errors by results since $output contains the full JSON structure returned by BrAPI. This may be useful for unimplemented calls raising 501 errors or values not found by current implementation raising 404 if your module can handle those. See also `brapi_call_wrapper()` documentation in `api/brapi.calls.inc`.
- `hook_brapi_CALL_FUNC_NAME_error_alter()`: allows other module to alter other type of errors (not raised by BrAPI, typically PHP exceptions). See also `brapi_call_wrapper()` documentation in `api/brapi.calls.inc`.

3.3 How to Implement Calls

For each call you would like to implement:

1. Create a function in `api/brapi.calls.inc` before `brapi_v1_external_call_json()` function and ideally near related functions with a name like:

`brapi_v1_[call_name]_[call_version]_json` (replace non-word characters in call name by underscores and lowercase all the name). For calls that may receive IDs, add a parameter with a NULL default value. for example, `brapi_v1_germplasm_germplasmdbid_13_json($germplasm_id = NULL)` for BrAPI call `germplasm/{germplasmDbId}` following BrAPI v1.3 specifications. The function will return an ordered PHP array `array($result, $metadata, $debug)` that will reflect the JSON array returned by BrAPI. 2. Update brapi_get_calls() in `api/brapi.const.inc` in order to make the `callback versions` subkeys of the call you are implementing to point to your new function. 3. Then do one of the following depending on the call:

- Call on a resource (ie. a resource ID may be provided)
- Search call
- Other type of call
- Override existing implementation

3.3.1 Call on a resource

In the `brapi_get_data_mapping()` function (in `api/brapi.const.inc`), find the `$brapi_data_mapping` associative array and make sure the resource exists in that array (as first level key). For instance the `marker` key will hold data mapping for markers.

Fill the corresponding resource mapping if it is not completed yet. All is documented in the `brapi_get_data_mapping()` function code documentation. The `fields` key holds the definition of both the BrAPI resource fields and internal fields. Internal fields might be used to compute BrAPI resource fields. Each field can be defined as one of the following:

- a column of the Chado table associated with the resource,
- a column of another Chado table linked to the table associated with the resource,
- a function that will handle the field value and operations on it (create, read, update, delete or a subset of those operations)
- an alias for another field.

Some BrAPI field values may be stored in several ways in Chado, depending how people use Chado. Such fields should always be defined as alias and the various ways of storing the valus should use internal field names.

For instance, germplasm 'species' field could be stored as a stockprop (plain text), a stock_cvterm (in a CV holding species names), in the organism table (as species name), in the phylonode table (when the linked organism correspond to a sub-taxa rather than the species) or maybe in another creative way. Therefore, for each of those case, an internal field is created (let's say 'speciesProp', 'speciesCVTerm', 'speciesOrganism', 'speciesPhylonode' and 'speciesMyWay') and the main field 'species' used by BrAPI will be an alias to one of those internal fields. That alias can be modified by the administrator to make it point to the right internal field on the BrAPI administration interface. The alias default value will be then overridden by the admin settings right after the definition of the `$brapi_data_mapping` array.

Also note that deprecated field names, for instance 'germplasmSpecies' may also be set as aliases to the non-deprecated fields replacing them (here 'species') that might be an alias as well (ie. alias > alias > internal field).

The implementation of your `brapi_v1_[call_name]_[call_version]_json($resource_id)` that will handle CRUD operations will look like this:

```
 function brapi_v1_[call_name]_[call_version]_json($resource_id)
{
   $actions = array(
```

(continues on next page)

(continued from previous page)

```
    'create' => 'brapi_v1_create_[call_name]_[call_version]_json',

    'read'   => 'brapi_v1_read_[call_name]_[call_version]_json',

    'update' => 'brapi_v1_update_[call_name]_[call_version]_json',

    'delete' => 'brapi_v1_delete_[call_name]_[call_version]_json',

    'list'   => 'brapi_v1_search_[call_name]_[call_version]_json',
  );

  return brapi_process_crud('[call_name]', $actions, '[resource_type]', $resource_
↪id);
 }
```

Note: You may only need to implement a subset of CRUD. In this case just remove unimplemented operations from the $actions array.

Then you will have to define each function that will handle a CRUD operation. The following provides an example of a read operation:

```
function brapi_v1_read_[call_name]_[call_version]_json($resource) {

  $metadata = brapi_prepare_metadata(1); // We return 1 element.

  $debug_data = array();


  // We delegate data loading to a specialized function.

  $resource_type_data =
  brapi_get_[resource_type]_[call_version]_details($resource);
  $data = array('result' => $resource_type_data);


  brapi_aggregate_call($data, $metadata, $debug_data);

  return array(

    $data,

    $metadata,

    $debug_data,

  );
}
```

The function returns an array of 3 elements used in the BrAPI JSON answer.

3.3.2 Search call

As for the call on a resource, the data mapping must be defined. The search function should look like this:

```
function brapi_v1_search_[call_name]_[call_version]_json() {

  $cv_settings = brapi_get_cv_settings();


  $parameters = array(

    'selectors' => array(
      [chado_filter]

    ),

    // We provide the function that will load the resource BrAPI fields.
  'get_object_details' => 'brapi_get_[resource_type]_[call_version]_details',

  );


  // This function will magically do the searching, handle pagination
  // and user filters for you.

  return brapi_v1_object_search_json('[resource_type]', $parameters);

}
```

The `selectors` key is used to set a default filter on which other filters will be later added. We use the same format for `[chado_filter]` as is required by `chado_generate_vars`. For instance, if your resource is germplas, you will want to return only germplasm from the Chado stock table. This requires filtering by stock.type_id to ensure only stocks with a type specified in the configuration are returned. Thus, in the code snippet above [chado_filter] would be `type_id => [cvterm_ids entered in config form]`.

Both resource create, read, update and list/search operations use the following specialized function:

```
function brapi_get_[resource_type]_[call_version]_details($resource) {

    $resource_type_data = array();


    $fields = array(

     [list_of_brapi_field_names_to_output],

    );


    // Some additional fields may be requested in the query;
    // auto-add them to the list.

    $fields = brapi_merge_fields_to_include($fields);


    // Load/fetch each requested field value.

    foreach ($fields as $field_name) {

      // All is magically handled by the data mapping defined earlier.
  $resource_type_data[$field_name] =
```

(continues on next page)

(continued from previous page)

```
    brapi_get_field('[resource_type]', $resource, $field_name);

    }


    return $resource_type_data;

}
```

3.3.3 Other type of call

Any BrAPI call function should do something similar to:

```
function brapi_[call_name]_[call_version]_json() {

  $debug = array();

  $metadata = brapi_prepare_metadata([element_count_or_1_or_nothing]);

  $data = array();


  // compute result data.


  // Optional: aggregate data from external calls.
brapi_aggregate_call($data, $metadata, $debug);


  // Return the BrAPI array in PHP that will be transcribed into a JSON string.

  return array(
    array('result' => array('data' => $data)),

    $metadata,

    $debug,

  );

}
```

3.3.4 Override existing implementation

Existing implementations car be overridden from another Drupal module using the following hooks:

- `hook_brapi_calls_alter(&brapi_calls)` to replace the functions used by calls. Use this method of you don't need to use existing call functions or if you need to implement your own handler for the way you store your data.
- `hook_brapi_CALL_FUNC_NAME_alter(&$data, &$context)` to keep existing call function (and setting selection) but just alter the default output result. This can be useful if you just want to remove (access restriction) or add data.

CHAPTER 4

Maintainers

Current maintainers:

- Valentin Guignon (vguignon) - https://www.drupal.org/user/423148

CHAPTER 5

Sponsors & Partnerships

The Breeding API Drupal implementation has been sponsored by Bioversity International, a CGIAR Research Centre. The Breeding API project has been sponsored by the Bill and Melinda Gates Foundation which funded the breeding API hackathon in June 2015 in Seattle and in July 2016 in Ithaca.

Partners of the Breeding API project are:

- Bioversity International http://www.bioversityinternational.org/
- BMS http://www.integratedbreeding.net/breeding-management-system/
- BTI (Cassavabase, Musabase) http://bti.cornell.edu/
- CIMMYT http://www.cimmyt.org/
- CIP http://cipotato.org/
- CIRAD http://www.cirad.fr/
- GOBII Project http://gobiiproject.org/
- IRRI http://irri.org/
- The James Hutton Institute http://www.hutton.ac.uk/
- WUR http://www.wur.nl/