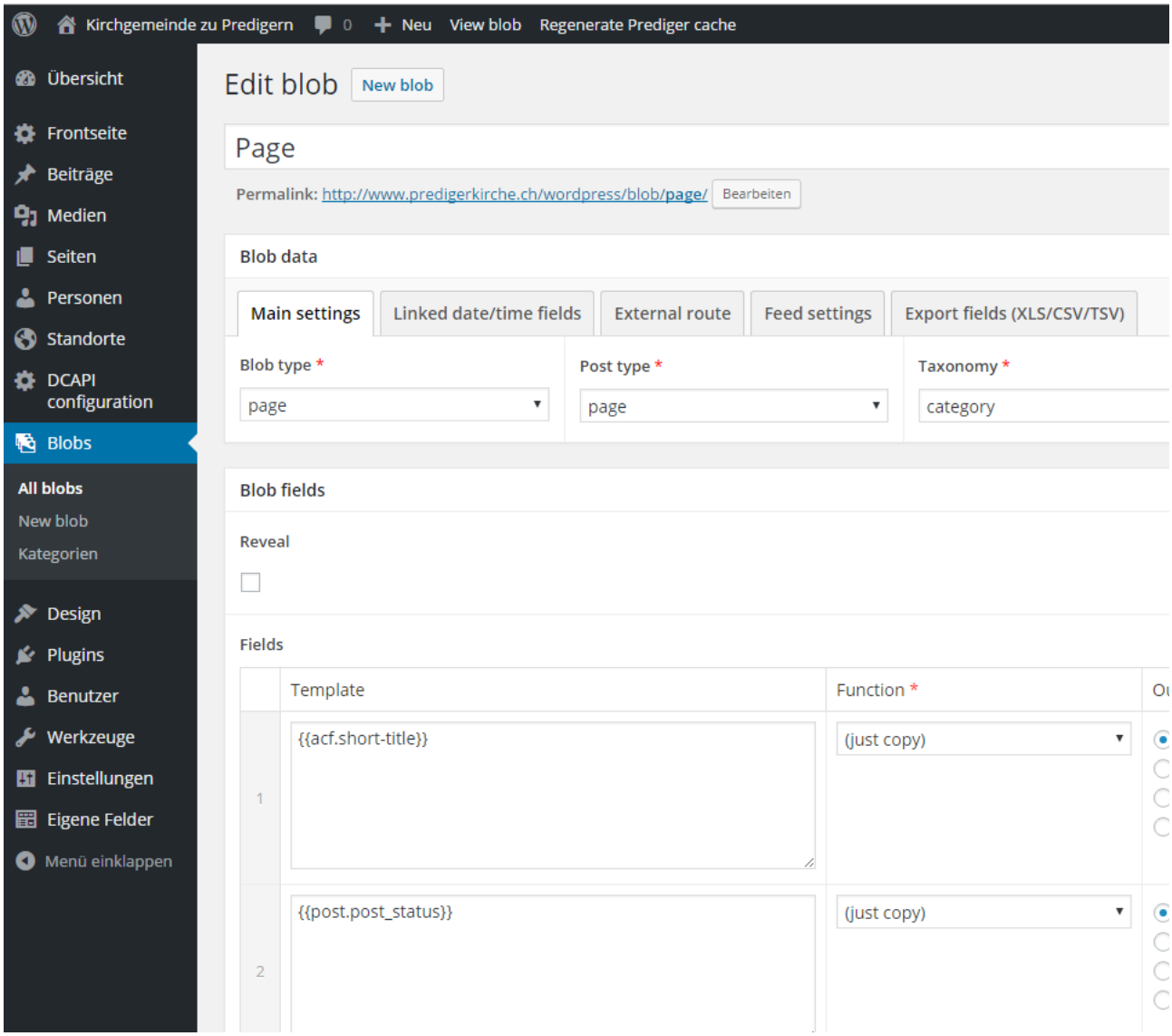
DCAPI – Data-centric API

A plugin for Wordpress

Current DCAPI version V1.2.1, 2016-0813

Document version 2016-08-13



Contents

Summary

This document describes the features of the DCAPI Wordpress plugin.

This plugin provides a cached data-centric view of a Wordpress site enabling Wordpress to be used as a content management system (back-end "model") for a separately implemented user interface (front-end "view" and "controller"). The data is exchanged using JSON structures that are cached in server files and served via a PHP-based API to improve front-end performance.

DCAPI provides a REST-style data-centric interface that serves up the data in a JSON format that can readily be processed by modern web technologies such as Javascript, and the data is cached to provide high performance for a client using HTTP to access the interface. The individual data items presented by the interface are highly configurable, so that Wordpress "internals" can largely be hidden.

DCAPI works using the concept of data "blobs". A blob is basically a transformed version of a Wordpress page, post or custom post. A central part of the design is the idea of a "feed", which is a set of (custom) post blobs selected by means of Wordpress category or custom term settings, where the feed is associated with a particular page blob. The data used for selecting the feed items is read from the page in question, meaning that very flexible feed mechanisms can be built.

DCAPI requires the plug-in Advanced Custom Fields PRO ("ACF Pro" by Elliot Condon) – see http://www.advancedcustomfields.com/pro/ – which provides the facility to extend Wordpress pages and posts with additional data fields in an extremely flexible way. It is not necessary to buy an ACF Pro license to use it with DCAPI. However, if you wish to use the full facilities of ACF Pro, then it is necessary to acquire a developer license. In the rest of this document, "Advanced Custom Fields" is referred to using the abbreviation "ACF".

Using the possibility to create option pages with ACF Pro, DCAPI allows one additional "home page" to be created, based on option page data. This can be useful when creating a front page with different format from normal pages.

DCAPI has the concept of a "search" blob that returns a feed of items selected using a search string. DCAPI does not require the plug-in Relevanssi (by Mikko Saari), but is aware of it and will replace standard Wordpress search query processing transparently if Relevanssi is present. DCAPI can also generate a "config" blob that returns general configuration data about the page and category/term hierarchies.

DCAPI also provides a number of extensions to the Wordpress Back End, enabling useful customisations to be made. It is also possible to use Wordpress as Front End, by making use of the DCAPI shortcode facility, should this be so wished.

DCAPI also provides the possibility to import data from another DCAPI site, enabling content to be shared between multiple sites without duplicate data entry being needed.

Finally, DCAPI allows user-programmed PHP functions to be registered enabling DCAPI field and feed processing to be extended if necessary. Otherwise, no special programming experience is required.

Installation

Copy the following two directories to the Wordpress plugins directory (`../wp-content/plugins/`):

- `advanced-custom-fields-pro`
- `dcapi`

If there is a `cache` subdirectory in the `dcapi` directory, it can be deleted.

In Wordpress, *always* activate the "Advanced Custom Fields Pro" plugin *before* activating the "Data-centric API" plugin. Should unexpected errors occur, *move* the above two directories out of the Wordpress plugins directory to a temporary directory, restart Wordpress, and then move the two directories back, reactivating them in the above order.

Supposing that Wordpress is installed in the top level directory of mysite.com, then the root path for the API is `http://mysite.com/wp-content/plugins/dcapi/` Throughout this document, the shortcut `../dcapi/` is used to refer to the DCAPI root. Depending on the exact Wordpress installation, the path may differ, but it is always derived from the location of the DCAPI plugin.

When DCAPI is activated, two new options appear on the admin dashboard:

- DCAPI configuration – configuration of DCAPI for the whole site is performed here
- Blobs – the various "blobs" are configured here.

The first time DCAPI is activated, it should create the following blobs with basic settings:

- Page
- Post
- Search

If the blobs are not created for any reason, enter `../dcapi/^restore/Default` in a browser.

To check your installation, enter `../dcapi/^info.jsonpp` in a browser. If correctly activated, DCAPI will return basic information about the site:

```
{
    "Site name":"Prediger",
    "GUID":"00a20f56-115a-4a8b-afe8-9b759c70bfcb",
    "DCAPI url":"http:\/\/dev.predigern.ch\/wordpress\/wp-content\/plugins\/dcapi\/",
    "DCAPI version":"1.2.1, 2016-08-12",
    "Wordpress version":"4.5.3",
    "PHP version":"7.0.5",
    "ACF version":"Advanced Custom Fields PRO, V5.3.9.1",
    "Search processor":"Relevanssi",
    "Timeout limit":"20 s",
    "Last modification times (UTC)":{
        "blob":"2016-08-09 19:04:14",
        "config":"2016-08-12 16:57:36",
        "page":"2016-08-11 15:34:08",
        "post":"2016-08-12 16:32:34",
        "term":"2016-08-12 10:38:34"
    },
    "Last cron run (UTC)":"2016-08-13 17:22:31: no updates performed"
}
```

Congratulations! You have accessed data about your site using the DCAPI API!

A more comfortable way to check the data returned by DCAPI is to use a REST Client such as Postman. Postman runs as an application via a Chrome browser. It is very useful for interfacing with REST APIs such as DCAPI, especially as it can "pretty format" JSON data.

Each DCAPI site has its own unique GUID. This is used to inhibit self-referencing via remote site importing, and is not normally of relevance to a DCAPI user. DCAPI is known to run correctly under PHP versions from 5.4 upwards, and has also been tested for compatibility with PHP 7.

DCAPI Cache and Cache Regeneration

DCAPI uses a file-based cache when serving up its data, so that data is normally returned very fast. HTTP cache and compression features are fully supported, so in many cases data is not even sent if it is already in the cache of the requester. Wordpress code is only loaded and run if the data in the cache is stale or missing for some particular reason. Normally, the user can ignore the fact that DCAPI uses a cache. The cache is located in the directory `../wp-content/plugins/dcapi/cache/`. All the files and directories in the cache can freely be deleted and regenerated. To regenerate cache files, the "`?regen`" option is used (see the API documentation below for more details).

If the site uses date/time based feed filters, then it is necessary to regenerate the entire cache at least once per day. This can be carried out by using a cron job service such as setcronjob.com. In addition since changes during the day can affect the cache, it is worth carrying out regular cache updates to update any missing files, using the following actions:

`../dcapi/^clear`	– once per night (preferably after 02:00 in Switzerland so that the UTC day change has also occurred!), to completely clear and regenerate the cache
`../dcapi/^cron`	– regularly during the day, for instance every five minutes or so, to update any missing files

If the `^cron` action cannot be run regularly, then clear the "Cron update" tick box on the Main tab of the DCAPI Configuration screen. This will cause regeneration to take place in the background after each Wordpress request is made.

To see which recent changes have taken place, run the `../dcapi/^log` action.

The directory `../dcapi/backup` is used for storing backed-up blob configuration data. The file `Default.json` stored here contains a copy of the default blob settings and is used for the initial activation of a site.

API

Internal blob routes & "repeated" routes

The standard way to access DCAPI data is to use an "internal" route. This consists of a prefix followed optionally by a forward slash '/' and an ID (which is normally a numeric string), for example `page/237` or `post/1232`. The prefixes and format of internal routes can be configured for each blob individually. The ID is generally the same as the Wordpress page/post ID.

Thus, to access the data of the page with ID 237, the DCAPI call needed is:

```
../dcapi/page/237
```

Posts having a "repeating" field generate multiple routes, one for each value of the "repeating" field. The different output blobs have "^*rrr*" appended to the end of the route (where "`rrr`" is one of the repeating values):

`../dcapi/post/1232^2016-09-14` (here, "`2016-09-14`" is one of the repeating values)

Wildcard routes

To refer to all blobs with a particular prefix, use the star character '*' as wildcard, such as:

```
../dcapi/page/*
```

To refer to all blobs, use:

```
../dcapi/*
```

Internal routes using wildcards are normally only used when regenerating the cache. If no options are provided when using wildcard routes, then the current cache status of all items is returned.

External blob routes

DCAPI also allows "external" routes to be configured. These are typically used in the Front End to refer to a given page or post in local language, and may make use of the Wordpress slug (rather than the numeric ID). For example, the external route of page 237 may be configured to be `in-eigener-sache` (the slug of that particular page). To access that page using the external route, use the following syntax (with an additional leading slash character '`/`'):

```
../dcapi//in-eigener-sache
```

The external route of `post/1232` might be `beitrag/1232`, for example. To access by external route, use:

```
../dcapi//beitrag/1232
```

"Repeating" field values may also be appended to external routes:

```
../dcapi//beitrag/1232^2016-09-17
```

A special external route is provided for accessing the home page (if there is no "home" or "home-page" blob, then the front page configured in Wordpress (if any) is returned):

`../dcapi//^home` *or (as shortcut)* `../dcapi//`

Finally, special access to the search blob (if any) is provided without needing to know either the internal or external route that has been configured for that blob. To search for string `sss`, use:

```
../dcapi//^search/sss
```

The configured setting might differ and be `../dcapi/lookup/sss` (internal) or `../dcapi//suchen/sss` (external), for example. A special shortcut version of the local search route is also provided as an action code, as follows: `../dcapi/^search/sss`

Routes for accessing remote site data

Remote site blob data can be accessed via DCAPI, so long as the remote site also uses DCAPI (version V1.2 or greater). To set this up a special "clone" blob is defined on the local site, pointing to a post-type blob on the

current site or a remote site, and specifying a particular source and target term/category. Once this has been set up, the clone blob can be referred to just like a local blob. A locally cached copy of the remote data is made for performance reasons. Because of this, the data returned locally may not be fully up to date with the data on the remote site until the site cache has been fully regenerated.

Special routes

DCAPI also provides the following special internal routes that provide access to key configuration data, and which are not normally needed explicitly by Front End developers.

`../dcapi/^blob`	This route returns a block of data containing the DCAPI configuration for the site.
`../dcapi/^index`	This route returns a block of data that contains full post/page list, a category/term list and other useful indexes.
`../dcapi/`*`bbb`*`/^feed`	This route returns the "feed" contents for all blobs with prefix *`bbb`*

Options

Option settings can be appended to the end of DCAPI routes, as follows.

`?regen`	to regenerate the cache of one or more individual blobs (wildcard routes can be used)
`?clear`	*(can be combined with `?regen`)* to clear the cache first and then regenerate (only for wildcard routes)
`?do`	*(combine with `?regen`)* force regeneration of a set of blobs in the foreground (by default, regeneration takes place in the background) (only for wildcard routes)
`?force`	*(combine with `?regen`)* force regeneration of all blobs – normally only missing or stale files are regenerated (only for wildcard routes)
`?quiet`	suppress "up to date" and "file missing" with `?clear` (only for wildcard routes)
`?format=fff`	sets the output format (where `xxx` can be "`json`" [default], "`jsonpp`" [pretty printed JSON], "`xls`" [XML Spreadsheet 2003], "`csv`" [semicolon separated], "`tsv`" [tab separated] or `xml`). Alternatively, append "`.json`" etc. to the end of the route
`?touch`	to get the last update time of an individual blob

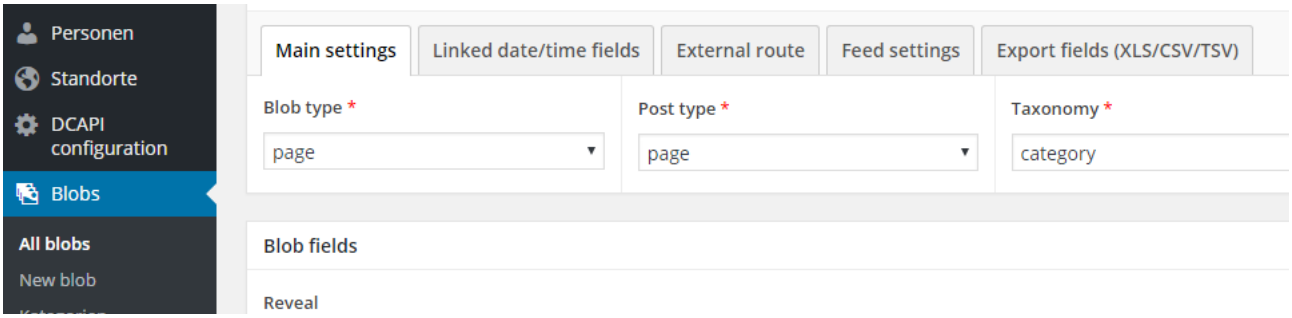
Actions

The DCAPI API provides a number of action verbs, as follows.

`../dcapi/`	equivalent to `../dcapi/^info`
`../dcapi/^backup`	perform blob data backup (stored in `../dcapi/backup/`)
`../dcapi/^backup/`*`path`*	access a blob data backup
`../dcapi/^restore/`*`path`*	restore blob data from a previous backup
`../dcapi/^cache`	list status of all cache items – shortcut for `../dcapi/*`
`../dcapi/^clear`	shortcut for `../dcapi/*?clear;quiet`
`../dcapi/^cron`	similar to `../dcapi/^regen`, but only run if necessary (for cron jobs)
`../dcapi/^info`	get name of site, GUID, DCAPI version, PHP version and ACF version, etc.
`../dcapi/^log`	output the most recent 20 lines of the log file
`../dcapi/^log/`*`nnn`*	output the most recent `nnn` lines of the log file
`../dcapi/^login/`*`user/pwd`*	log into Wordpress (affects visibility of items and fields). NB: password in free text – use with care!
`../dcapi/^logout`	log out of Wordpress
`../dcapi/^regen`	shortcut for `../dcapi/*?regen;do`
`../dcapi/^touch`	to get last cache update timestamp
`../dcapi/^user`	to return currently logged in Wordpress user (if any)

Configuration

The configuration of DCAPI takes place entirely within Wordpress. Wordpress administrators have two additional menu entries on the admin dashboard, “DCAPI configuration” and “Blobs”. There is a “Blob fields” area used for field transformation details on both of these screens.



DCAPI configuration

The Blob config options panel has the following tabs:

- Main – the main settings for the site are made here
- Visibility – who sees “hidden” items is configured here
- Back End – settings that affect the Back End (Wordpress admin dashboard) are set here

Main tab

The following main settings are made here.

Site name	a short name for this site
Base URL	the base URL for external routes, starting `‘http://’` and ending in a forward slash character. If no value is supplied, then the Wordpress site base URL is used
Timeout limit	maximum processing time allowed for a multiple blob process, in seconds
Cron update	tick if a regular ^cron job (run, say, every 5 minutes) is used to update entries

Visibility tab

The Visibility tab contains settings that control the visibility of “hidden” items on the local site. Individual fields can be hidden (configured in Blob fields), and category/term trees can be hidden.

Reveal to	reveal hidden categories and fields to users with these roles. Hidden categories and fields are always revealed to Administrators
Hidden categories	hide posts with these categories/terms

Back End tab

The Back End tab allows one or two columns to be added to the post and page edit lists in the admin dashboard. If the column is given a title then it will be shown. The content is configured using Blob fields. The content for these columns is only generated when each blob is regenerated, so may be temporarily stale or absent.

Date/Status Column Title	if present, then a Date/Status column is created with the given title which overrides the normal Date column: the column is sortable
Info Column Title	if present, then an additional information column is created with the given title

Blobs

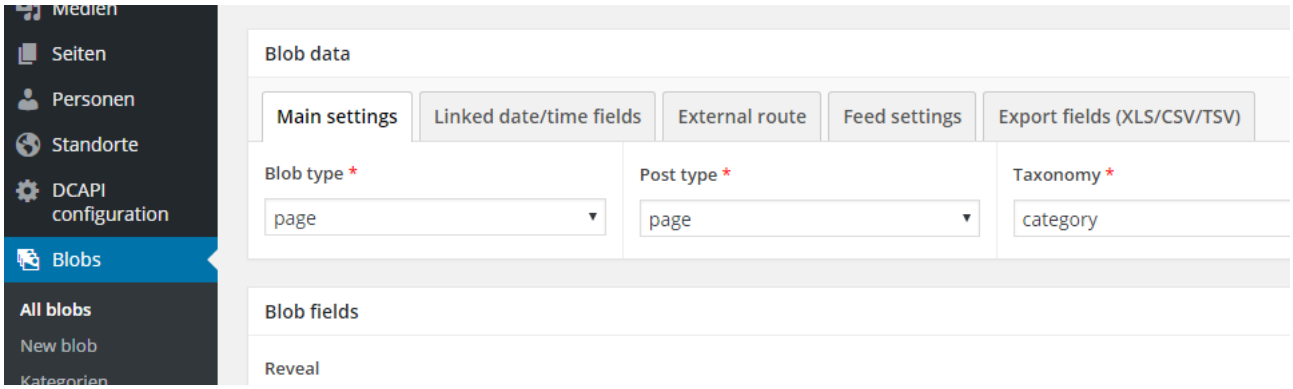
Each blob represents the transformation of a Wordpress page or (custom) post into output data on the DCAPI interface. For example, the "page" blob contains all the configuration information needed to transform Wordpress pages into DCAPI "page" data.

When creating a feed by selecting posts and including them in the output for a given page, the "post" blob is used to transform the post data and the "page" blob the page data. The "page" blob then controls how the feed items are selected.

Special blobs are used to configure home page, a special configuration blob and search output.

Each blob configuration has a number of tabs, as follows:

- Main settings – the main settings for each blob are made here
- Linked date/time fields – these settings enable ACF dates and times to be linked with the Wordpress page/post date/time
- External route – these settings affect the external routes created for the blob concerned
- Feed settings – for blobs with a feed, this tab controls what selection and sorting takes place (only present if the blob has a feed)
- Export fields (XLS/CSV/TSV) – this tab controls transformations used when outputting a blob in `XLS`, `CSV` or `TSV` format (only present if the blob has a feed).



Main settings

The Main settings tab is used to configure key aspects of a blob and to link it with the related Wordpress page/post type and category/term type. Depending on the "blob type", not all these fields (or the related tabs) are shown.

blob type	• `home` = home page (page/0) – the data for the home page comes from an ACF options page • `home-page` = selected existing page • `page` = page • `post` = (custom) post • `config` = configuration blob • `search` = search page • `clone` = clone type blob (copies a local or remote post-type blob) Either "`home`", "`home-page`" or neither may be present. Exactly one "`page`" blob and one "`post`" blob per (custom) post type may be present. There can be at most one "`config`" and one "`search`" blob. Any number of "`clone`" blobs is allowed (with different prefixes)

post type	the Wordpress post type: “page”, “post” or a valid (custom) post type name. “option” is used to refer to menu option pages. Standard fields are accessed in the context of the calling blob
taxonomy	the Wordpress category or taxonomy name to use with this blob type
prefix	the prefix used for internal routes, also returned in field ‘prefix’ in the output data each blob should have a unique prefix, with the exception of blobs of type ‘`home`’ and ‘`home-page`’, which always use the same prefix as ‘`page`’ type blobs
home page	the home page by reference (blob type “`home-page`”)
has feed	this blob type has a feed
repeating field	if this field is set, feed items are iterated across the values of this field, resulting in repeated, separately identified entries in the feed (NB: there is a limit applied to repeating fields to avoid very long feeds)
is cached	this blob type is cached

Linked date/time fields

The Linked date/time fields tab is used to control the behaviour of date/time fields when editing the page/post in Wordpress. This mechanism makes it possible to let the Wordpress user enter all relevant dates and times using ACF fields, and enables the Wordpress publication field block to be ignored.

ACF Date/Time field	If this ACF field reference is supplied, this date/time value is used to overwrite the post_date and post_date_gmt values (of the specified post_type). If the ACF Time field is supplied, ACF Date/Time is assumed to be a date only and is combined with the ACF Time field to get a date/time Only ACF fields are supported and the “field_xxxx” format must be used when referring to the field. Allowed formats: timestamp, “`yyyy-mm-dd hh:mm`”, “`yyyy-mm-dd hh:mm:ss`”, “`dd.mm.yyyy hh:mm`”, “`dd.mm.yyyy hh:mm:ss`”, “`mm/dd/yyyy hh:mm`” or “`mm/dd/yyyy hh:mm:ss`”. If the ACF Date/Time field is not present, no action is taken (even if the other fields on this tab are not empty!)
ACF Time field	If this ACF field reference is supplied, this time is appended to the ACF Date field value (with an intervening space character) and used to overwrite the post_date and post_date_gmt values (of the specified post_type) Only ACF fields are supported and the “field_xxxx” format must be used when referring to the field. Allowed formats: “`hh:mm`”, “`hh:mm:ss`”
Update ACF Date	If set, then the ACF Date/Time field (if specified) will be overwritten with the current post_date date/time using the PHP date format supplied in this field, when accessing the page/post. For example, “`Ymd`”.
Update ACF Time	If set, then the ACF Time field (if specified) will be overwritten with the current post_date date/time using the PHP date format supplied in this field, when accessing the page/post. For example, “`H:i`”

External route

The External route tab is used to configure how the blob concerned is referred to in the external route tables generated by DCAPI and used in conjunction with `..dcapi//` external route access.

Generate route	If set, generate an external route for this item. A home page always has the special external route `^home` (accessed via `../dcapi//^home` – NB: with a leading forward slash character ‘/’, or the shortcut version `../dcapi//`). There is no external route for config type blobs

Route prefix	External route prefix to use with this item type. If there is an ID or slug present then it is separated from the prefix by a forward slash character. Likewise, the search string is separated from the prefix by a forward slash character
Route style	Style of external route to this item type: *slug-based* or *ID-based* For pages, the slug-based option is hierarchical (with levels separated by forward slashes '/') and has no prefix; for posts, there must be a prefix. The ID-based option always has a prefix

Feed settings

The Feed settings tab is used for setting up feed output (if any). The processing of feeds is carried out after field processing (see below), and "out" fields are available to the standard feed processing routine. A key "out" field that is required is "`feed/type`", the value of which is used to select the relevant line in the Feed settings table. This "out" field must be populated in the Blob fields table, else no feed will be created.

In addition, the "out" field "`feed/search`" is required for search blobs. Normally, this field is set automatically (but can be overridden if necessary by an extra line in the Blob fields table).

Replication of items using the "repeating field" mechanism is applied before the feed is sorted and limited.

If a clone blob is included by the feed selected and the category/term matches, then the contents of that blob are also automatically included in the set of items used for the feed.

Feed type	The value of "out" field "`feed/type`" to be matched
Terms field	Which field provides the feed categories (if any)
Limit field	This field (if any) defines the maximum number of feed entries to output
Source blob(s)	The (custom) post blob(s) and clone blob(s) (if any) which provide the items for the feed
Ordering	• `ASC` = ascending (in the standard feed filter, if ordered by `date` or `param/date`, starting from today forwards inclusive) • `DESC` = descending (in the standard feed filter, if ordered by `date` or `param/date`, starting from today backwards inclusive) • `(as retrieved)` = not sorted
Order by	Which "out" field to use for sorting: `date` / `title` / `menu_order` / `param/date` / `param/order` / `relevance`. The field `relevance` is set automatically as "out" field if Relevanssi searching is being used
Filtering	A filter function called for each selected feed item: the function must return a (unique) sort key, or null if this particular item should be skipped

Export fields (XLS/CSV/TSV)

This tab is used to construct spreadsheet-style output for blobs with a feed (output formats `XLS`, `CSV` and `TSV`).

Column name	Column name to be used in the output spreadsheet. Avoid punctuation and spaces in the name. Each column should have a different name
Template	A handlebars template similar to that used in Fields processing (see below). The context is provided by the blob being output. Use '`blob`' to access the blob level, '`feed`' to access the feed level, or '`item`' to access the item level fields. Use a period to separate levels, eg "`{{item.title}}`". The output values for the current row can be accessed under '`out`'. One row is written out per item in the blob feed. If the feed is empty, then a single row is output

Fields processing ("Blob fields" table)

The fields tab can be compared with spreadsheet processing and controls the transformation of input fields to output fields. Basically, the set of "Blob fields" under the "DCAPI Configuration" tab are processed first, followed by the set of "Blob fields" defined for the current blob.

The algorithm operates as follows. Each row is processed in turn to convert input field values into named output values. The output values are collected in the context "`out`" array with the name given by "Output field". The template and function calls may use "out" values from previous rows, e.g. "`out.feed.type`". Finally, the "out" list values are output according to the "Output type" and then the "Output to" or "Back End" setting.

There are two special cases. For `clone` blobs, the local term setting is automatically copied to the standard "out" field "`terms`". And for `search` blobs, the search string from the current request is automatically copied to the standard "out" field "`feed/search`".

The settings in the fields are as follows.

Template	The first column "Template" contains a "handlebars" template (see http://handlebarsjs.com/ for more details). The context data for the template is as follows (depending on the context!): '`request`' the request (generally, the internal route) '`post`' the current Wordpress post data '`term`' the current Wordpress term/category data '`blob`' the current blob data '`baseUrl`' the base url for external routes '`locale`' the locale as defined in Wordpress '`acf`' the relevant set of ACF fields '`postInfo`' the postInfo data from `^index` (for the current post or page) '`termInfo`' the termInfo data from `^index` (for the current post) '`sourceBlob`' the original post's "blob" output (for `clone` blobs only) '`sourceFeed`' the original post's "feed" output (only `clone` blobs) '`input`' the current entry (for postInfo and termInfo setting) '`out`' the current set of output fields
Function	The output from running the handlebars template is then transformed by an optional "Function" to give an output value. The standard functions provided are as follows: • `(just copy)` – this simply copies the input value to the output • `clean up text` – this is used to clean up text fields by removing any unnecessary HTML tags • `convert terms to names` – given an array of term IDs as input, returns their names as an array • `create route` – this creates the relevant "external" route, given the Wordpress page/post ID as input (`ID` in the input field column) • `get term ID list` – returns a list of category/term IDs for the current item • `get term name list` – given the Wordpress page/post ID as input, returns all referenced categories/terms as a string of names separated by pipe characters '|' • `handle gallery` – this handles an ACF gallery field (array of IDs), returning an array of image data blocks • `handle image` – this returns a block of data for the "featured image" of a page or post, given the Wordpress page/post ID as input • `handle media image` – given a media item, this routine returns image data • `meta data` – produces the standard "meta" data block It is possible to register additional functions in a site plug-in (see below)

Output type	This column specifies how to process the particular field: 'normal' normal output processing 'hidden' output is hidden from normal users (see "Visibility" settings) 'repeat' output reflects each value of the "repeating" field (if any) 'template' store this template as a "partial", e.g. "{{>partial ...}}" 'postInfo' used for generating "postInfo" in the special ^index blob 'termInfo' used for generating "termInfo" in the special ^index blob 'back end' output is used in the Back End admin screen column specified 'clone' clones the full "blob" or "feed" data to the output
Output field	Fields should preferably be named alphanumerically using Camel case, and underscore '_' characters are allowed. Hierarchical fields can be output by separating the individual level names with ".": "defaults.page.image" creates the field "image" contained in the field "page" contained in the field "defaults". Up to three levels are supported Each time the same name is used in the output field column, the "out" list value will be overwritten: this enables overriding of standard field settings to take place
Output to	'blob' output to the blob itself 'feed' output to a special feed item list used when creating feeds
Back End	'date/status' output to the back end "Date/Status" column 'info' output to the back end "Info" column

The "hidden" field can be used to hide rows in the table (the initial setting for "standard" fields), to make it easier to scroll a long list.

The following handlebars helpers are provided:

if	{{#if value}} true-part {{/if}} *or* {{#if value}} true-part {{else}} false-part {{/if}}
unless	{{#unless value}} false-part {{/unless}} *or* {{#unless value}} false-part {{else}} true-part {{/unless}}
each	{{#each value type}} interating part {{/each}} where type (if present) signifies special processing: • if it is "cloneKey" then the output array uses the same key value • if it is "cloneValue" then the output array uses the input value as key • if it is "Map" then the output array is a map from value to input key
with	{{#with array}} content-in-array-context {{/with}}
lookup	{{lookup array index}}
log	{{log value}} – output (echo) value (useful for testing)
br *(DCAPI helper)*	{{br}} – insert a newline break tag
remove *(DCAPI helper)*	{{remove}} – inserts a "remove this attribute" signal, which signals that the repeat/hidden field should be removed rather than overwritten
retain *(DCAPI helper)*	{{retain}} – inserts a "retain this attribute" signal, which signals that the repeat/hidden field should be retained in the output (e.g. overwritten)
wordpress *(DCAPI helper)*	{{wordpress "option_name"}} – output Wordpress option value
postmeta *(DCAPI helper)*	{{postmeta "key"}} – returns a "single" post meta value, given the key
acf *(DCAPI helper)*	{{acf "key"}} – returns a "single" ACF option value, given the key
route *(DCAPI helper)*	{{route "ID"}} – returns a "route" block, given a (post) ID, else uses context to look up route

nextColor *(DCAPI helper)*	`{{nextColor "root"}}` – returns the next color (only used in termInfo processing!). If root is specified then it must match the current root term name
map *(DCAPI helper)*	`{{#map input_value type}} ... {{/map}}` where '...' are lines with format `test=>output_string` with the input value given (iterates if an array and sets `{{@value}}`) The first match of `input_value` returns `output_string`. The `test` can be a regex in the form `/.../p` and where `type` (if present) signifies special processing: • if it is "cloneKey" then the output array uses the same key value • if it is "cloneValue" then the output array uses the input value as key • if it is "Map" then the output array is a map from value to input key
eval *(DCAPI helper)*	`{{#eval input_value type}} ... {{/eval}}` evaluates the content "..." as if PHP code (with a limited set of functions), with the input value given (iterates if an array and sets `{{@value}}`) The first match of `input_value` returns `output_string`. The `test` can be a regex in the form `/.../p` and where `type` (if present) signifies special processing as for the `map` helper
format *(DCAPI helper)*	`{{format value fmt="Y-m-d" sep=" "}}` – multiple arguments are concatenated with spaces (or use sep parameter). Format a date-time using the standard PHP date settings, plus special format strings "@l" / "@D" to return day of week in German, and "@F" / "@M" to return month in German
list *(DCAPI helper)*	`{{list value sep=" " pre="" post="" tag=""}}` – multiple arguments are concenated with sep (default single space). `tag` creates a matching pair of HTML tags around the content. `pre` and `post` are optional prefix and postfix strings The tags are only output if field content is present. This general processing applies for most HTML tag codes, except in the following special cases: • In the special case of `"br"`, a single ` ` tag is output after the field, but only if the field is not empty • In the special case of `"a"`, the field value is output as the href attribute of the `<a>` tag group, and a "right arrow" character is displayed: `<a href='field_value'>&rArr;</a>` • In the special case of `"img"`, the field value is output as the src attribute of an `<img>` tag: `<img src='field_value' />` • Heading tags `"h1"` to `"h9"` are also treated specially. If the heading level output remains the same on following interations, then it is not output, until a change in a higher-level heading resets the processing for lower-level headings.

The following special handlebars field values are provided in iterative helpers:

- @first – true if this interation is the first
- @last – true if this interation is the last
- @key – the current key
- @index – the current array index
- @value – the current interation value

The DCAPI implementation of handlebars is analogous to handlebars.js, with the following restrictions:

- no processing of quadruple "raw" mustaches
- paths using embedded '/' are not processed
- @../index etc. paths are not supported
- "as" parameters are passed into handlers as hash parameters with the names "`asParam-`*n*" (*n* = 1..)
- the `log` helper echoes to the screen and there is no "`level=`" support provided

See http://handlebarsjs.com for more information.

Wordpress Shortcodes

DCAPI provides shortcode support to enables Wordpress to be used as Front End while still taking advantage of DCAPI's feed processing and importing of data from remote sites. The shortcode processing uses the same handlebars template approach used elsewhere in DCAPI.

As an example, the following shortcodes, entered as content on the Wordpress page, result in the feed contents from that particular page's blob being inserted into the Front End page output:

```
[dcapi]{{list item.monthYear tag="h2"}}<div id="first-line"><span class="all-
caps">{{item.weekday}}</span> {{item.date}}  {{item.time}}  &nb
sp; <span class="all-caps">{{item.ort}}</span></div>{{list item.link tag="a"}}
{{list item.title tag="b"}}{{br}}{{item.detail1}}{{br}}{{item.detail2}}[/dcapi]
```

The feed processing is triggered by a matched pair of shortcodes: **`[dcapi]...[/dcapi]`**, marked bold in **red** above. The block of characters within the dcapi bracket is repeated for each feed item. Handlebars templates, only valid within the dcapi shortcode block, shown above in **dark green**, are used to output blob feed fields. For example, `{{item.detail2}}` causes the value of the blob feed field "detail2" to be inserted.

Formatting information can be handled using the `list` helper. In the example of `{{list item.title tag="b"}}`, the blob feed field "title" is output surrounded by `<b>..</b>` HTML tags. It is possible to refer to general blob fields (outside the feed block), by prefixing the field name with '`blob.`', and to refer to feed fields (outside the feed item data), by prefixing the field name with '`feed.`'.

Finally, using a **`[dcapi blob]...[/dcapi]`** block, the context is set to the blob level and no feed iteration takes place. Similarly, a **`[dcapi feed]...[/dcapi]`** block sets the context to be at the feed level. A **`[dcapi item]...[/dcapi]`** block sets the context to be at the item level (default).

Information for Programmers

Registering Filter Functions

It is possible to register PHP filter functions to extend blob processing. The following possibilities are currently provided:

- Transformation function
- Feed filter

Transformation function

To register a transformation function, use the following PHP code:

```
global $DCAPI;                                                    – if needed
$DCAPI->register_transformation($name, $function);
```

where `$name` is the name to use in the Fields table and `$function` is the (globally defined) function to call.

The signature of the function is as follows:

```
function copy_item($blob, $post, $input_value, &$out)
```

where `$blob` is the incoming blob configuration, `$post` is the incoming current post, `$input_value` is the value generated by the field template, and `&$out` is the output 'out' list. The return value is the value to set the output field to. It is also possible to set other output fields as follows:

```
$out[$out_field] = value;
```

Note that such individually set output values will not be output unless there is at least one line on the Blob fields sheet with the necessary Output field value with the given name (`$out_field`).

Feed filter

To register a feed filter function, use the following PHP code:

```
global $DCAPI;                                                    – if needed
$DCAPI->register_feed_filter($name, $function);
```

where `$name` is the name to use in the Feed settings table and `$function` is the function to call.

The signature of the function is as follows:

```
function feed_filter($out, $order, $orderby, $itemDetails)
```

where `$out` is the 'out' list array, `$order` is the ordering value, `$orderby` is the name of the field to order by, and `$itemDetails` is a collection of the 'item' and 'details' field settings for the current feed item.

The function should return a (unique) sort key for the item, or `null` if this particular feed item should be skipped. If the sort key returned is not unique, then the latest feed item will override an existing feed item with the same sort key.

Example Output

This section gives some examples of DCAPI output from a live site, formatted using the `.jsonpp` option. Elision marks "..." show where content has been omitted for the sake of clarity.

Post blob


```
{
  “meta”: {
    “route”: {
      “intRoute”: “post/1232”,
      “extRoute”: “/beitrag/1232”
    },
    “self”: “http://dev.predigern.ch/wordpress/wp-content/plugins/dcapi/post/1232”,
    “copyright”: “\\u00A9 2016, Kirchgemeinde zu Predigern [DEV]. Authorized use only”,
    “lastModified”: “2015-11-24 06:20:48”,
    “cacheTimestamp”: 1470274229
  },
  “prefix”: “post”,
  “ID”: 1232,
  “type”: “post”,
  “title”: “Morgenmeditation”,
  “date_status”: “08.01.2015 07:00\n <em>veröffentlicht</em>”,
  “status”: “publish”,
  “slug”: “morgenmeditation”,
  “fullSlug”: “beitrag/1232”,
  “subtitle”: “Tagzeitengebet”,
  “dateTimeType”: “weeklyRepeat”,
  “repeatStartTime”: “07:00”,
  “fromDateTime”: “1420696800”,
  “fromDT”: “2015-01-08 07:00”,
  “toDateTime”: “1483164000”,
  “toDT”: “2016-12-31 07:00”,
  “weeklyRepeat”: [
    “4”
  ],
  “nextEvents”: [
    “2016-08-04”,
    “2016-08-11”,
    “2016-08-18”,
    “2016-08-25”,
    “2016-09-01”,
    “2016-09-08”,
    “2016-09-15”,
    “2016-09-22”,
    “2016-09-29”,
    “2016-10-06”,
    “2016-10-13”,
    “2016-10-20”,
    “2016-10-27”,
    “2016-11-03”,
    “2016-11-10”,
    “2016-11-17”,
    “2016-11-24”,
    “2016-12-01”,
    “2016-12-08”,
    “2016-12-15”
  ],
  “content”: “Singend und schweigend in den Tag</p><p>Jeweils donnerstags 7.00 bis 7.45 Uhr<strong><br /> </strong>”,
  “terms”: [
    {
      “term_id”: 89,
      “taxonomy”: “category”,
      “name”: “Tagzeitengebet”,
      “level”: 1,
      “parentTerms”: [
        2
      ],
      “useAsSubtitle”: true,
      “linkedTo”: {
        “intRoute”: “page/0”,
        “extRoute”: “/^home”
      }
    }
  ],
  “location”: {
    “ID”: 1246,
    “type”: “location”,
    “route”: {
      “intRoute”: “location/1246”,
      “extRoute”: “/standort/1246”
    },
    “slug”: “predigerkirche”,
    “title”: “Predigerkirche”,
    “content”: “<p>Die Predigerkirche ist am Zähringerplatz in Zürich</p><p>Das Turmzimmer befindet sich im 2. Stock im Turm – Eingang durch
die Seitentür oder neben dem Präsenzdienst. Ein Lift ist vorhanden</p>”,
    “terms”: [
      91,
      90
    ],
    “coordinate”: {
      “address”: “Predigerkirche, Zähringerplatz, Zurich, Switzerland”,
      “lat”: “47.37388”,
      “lng”: “8.545094000000063”
    }
  },
  “contributors”: [
```

```
    {
      “ID”: 1006,
      “type”: “person”,
      “route”: {
        “intRoute”: “person/1006”,
        “extRoute”: “/person/1006”
      },
      “slug”: “renate-von-ballmoos”,
      “role”: “Leitung”,
      “name”: “Pfrn. Renate von Ballmoos”
    }
  ],
  “categories”: [
    {
      “ID”: 89,
      “type”: “category”,
      “title”: “Tagzeitengebet”,
      “slug”: “tagzeitengebet”
    }
  ],
  “image”: {
    “ID”: “1684”,
    “type”: “attachment”,
    “mimeType”: “image/jpeg”,
    “url”: “http://dev.predigern.ch/wordpress/wp-content/uploads/2015/03/early-butterfly-e1425827213873.jpg?1469371587”,
    “thumbnailUrl”: “http://dev.predigern.ch/wordpress/wp-content/uploads/2015/03/early-butterfly-e1425827213873-150x150.jpg”
  },
  “excerpt”: “Singend und schweigend in den Tag”
}
```


Page blob having a “feed”


```
{
  “meta”: {
    “route”: {
      “intRoute”: “page/148”,
      “extRoute”: “/gottesdienste”
    },
    “self”: “http://dev.predigern.ch/wordpress/wp-content/plugins/dcapi/page/148”,
    “copyright”: “\\u00A9 2016, Kirchgemeinde zu Predigern [DEV]. Authorized use only”,
    “lastModified”: “2015-05-03 19:56:27”,
    “cacheTimestamp”: 1470279800
  },
  “prefix”: “page”,
  “ID”: 148,
  “type”: “page”,
  “title”: “Gottesdienste”,
  “date_status”: “16.01.2015 18:30\n <em>veröffentlicht</em>”,
  “status”: “publish”,
  “slug”: “gottesdienste”,
  “fullSlug”: “gottesdienste”,
  “parentPage”: {
    “isCached”: true,
    “route”: {
      “intRoute”: “page/0”,
      “extRoute”: “/^home”
    },
    “title”: “Kirchgemeinde zu Predigern”,
    “shortTitle”: “Frontseite”
  },
  “childPages”: [
    {
      “isCached”: true,
      “route”: {
        “intRoute”: “page/150”,
        “extRoute”: “/gottesdienste/sonntagsgottesdienste”
      },
      “title”: “Sonntagsgottesdienste”,
      “shortTitle”: “Sonntags”
    },
    {
      “isCached”: true,
      “route”: {
        “intRoute”: “page/152”,
        “extRoute”: “/gottesdienste/werktagsgottesdienste”
      },
      “title”: “Werktagsgottesdienste”,
      “shortTitle”: “Werktags”
    },
    {
      “isCached”: true,
      “route”: {
        “intRoute”: “page/158”,
        “extRoute”: “/gottesdienste/andere-gottesdienste”
      },
      “title”: “Gottesdienste im Jahreskreis”,
      “shortTitle”: “Im Jahreskreis”
    },
    {
      “isCached”: true,
      “route”: {
        “intRoute”: “page/2358”,
        “extRoute”: “/gottesdienste/im-pfrundhaus”
      },
      “title”: “Im Pfrundhaus”
    }
  ],
  “image”: {
    “ID”: “1751”,
    “type”: “attachment”,
    “mimeType”: “image/jpeg”,
```

```
    “url”: “http://dev.predigern.ch/wordpress/wp-content/uploads/2015/03/2015-03-08_7761-e1426283649562.jpg?1469371581”,
    “thumbnailUrl”: “http://dev.predigern.ch/wordpress/wp-content/uploads/2015/03/2015-03-08_7761-e1426283649562-150x150.jpg”
  },
  “content”: “Gemeinsam feiern und beten - auf die göttliche Stimme hören, in Stille, Musik und durch die biblischen Traditionen.”,
  “excerpt”: “Gemeinsam feiern und beten - auf die göttliche Stimme hören, in Stille, Musik und durch die biblischen Traditionen.”,
  “feed”: {
    “type”: “scheduledPosts”,
    “termList”: [
      2
    ],
    “title”: “Kommende Gottesdienste”,
    “count”: 45,
    “items”: [
      {
        “route”: {
          “intRoute”: “post/5358”,
          “extRoute”: “/beitrag/5358”
        },
        “prefix”: “post”,
        “ID”: 5358,
        “type”: “post”,
        “title”: “Sommervesper III - Musik aus aller Welt”,
        “post_date”: “2016-08-05 18:30:00”,
        “status”: “future”,
        “slug”: “sommervesper-lll-musik-aus-aller-welt”,
        “fullSlug”: “beitrag/5358”,
        “subtitle”: “Freitagsvesper”,
        “dateTimeType”: “dateTime”,
        “startDateTime”: “1470414600”,
        “startDT”: “2016-08-05 18:30”,
        “location”: {
          “ID”: 1246,
          “type”: “location”,
          “route”: {
            “intRoute”: “location/1246”,
            “extRoute”: “/standort/1246”
          },
          “slug”: “predigerkirche”,
          “title”: “Predigerkirche”,
          “content”: “<p>Die Predigerkirche ist am Zähringerplatz in Zürich</p><p>Das Turmzimmer befindet sich im 2. Stock im Turm – Eingang
durch die Seitentür oder neben dem Präsenzdienst. Ein Lift ist vorhanden</p>”,
          “terms”: [
            91,
            90
          ],
          “coordinate”: {
            “address”: “Predigerkirche, Zähringerplatz, Zurich, Switzerland”,
            “lat”: “47.37388”,
            “lng”: “8.545094000000063”
          }
        },
        “contributors”: [
          {
            “name”: “Michelle Defalque und Martin Mäder”,
            “role”: “Musik”
          },
          {
            “name”: “Roland Brendle”,
            “role”: “Liturg”
          }
        ],
        “categories”: [
          {
            “ID”: 4,
            “type”: “category”,
            “title”: “Freitagsvesper”,
            “slug”: “freitagsvesper”
          }
        ],
        “image”: {
          “ID”: “5353”,
          “type”: “attachment”,
          “mimeType”: “image/jpeg”,
          “url”: “http://dev.predigern.ch/wordpress/wp-content/uploads/2016/06/weltmusik-e1464874965253.jpg?1469370409”,
          “thumbnailUrl”: “http://dev.predigern.ch/wordpress/wp-content/uploads/2016/06/weltmusik-e1464874965253-150x150.jpg”
        },
        “excerpt”: “Weltlieder, farbenfroh interpretiert mit Stimme, Gitarre und Kontrabass”,
        “index”: 0
      },
      {
        “route”: {
          “intRoute”: “post/5454”,
          “extRoute”: “/beitrag/5454”
        },
        “prefix”: “post”,
        “ID”: 5454,
...

        “type”: “post”,
        “title”: “Neue Klänge zum neuen Jahr”,
        “post_date”: “2017-01-01 17:00:00”,
        “status”: “future”,
        “slug”: “zwischen-zeit-und-ewigkeit-duplikat”,
        “fullSlug”: “beitrag/4896”,
        “subtitle”: “Gottesdienst”,
        “dateTimeType”: “dateTime”,
        “startDateTime”: “1483286400”,
        “startDT”: “2017-01-01 17:00”,
        “location”: {
          “ID”: 1246,
          “type”: “location”,
          “route”: {
            “intRoute”: “location/1246”,
```

```
            "extRoute": "/standort/1246"
          },
          "slug": "predigerkirche",
          "title": "Predigerkirche",
          "content": "<p>Die Predigerkirche ist am Zähringerplatz in Zürich</p><p>Das Turmzimmer befindet sich im 2. Stock im Turm – Eingang
durch die Seitentür oder neben dem Präsenzdienst. Ein Lift ist vorhanden</p>",
          "terms": [
            91,
            90
          ],
          "coordinate": {
            "address": "Predigerkirche, Zähringerplatz, Zurich, Switzerland",
            "lat": "47.37388",
            "lng": "8.545094000000063"
          }
        },
        "contributors": [
          {
            "name": "Ruth Bischofberger",
            "role": "Flöten"
          },
          {
            "ID": 1029,
            "type": "person",
            "route": {
              "intRoute": "person/1029",
              "extRoute": "/person/1029"
            },
            "slug": "christian-doehring",
            "role": "Klavier und Orgel",
            "name": "Christian Döhring"
          },
          {
            "ID": 2265,
            "type": "person",
            "route": {
              "intRoute": "person/2265",
              "extRoute": "/person/2265"
            },
            "slug": "renate-von-ballmoos-2",
            "role": "Liturgie",
            "name": "Renate von Ballmoos"
          }
        ],
        "categories": [
          {
            "ID": 74,
            "type": "category",
            "title": "Gottesdienst",
            "slug": "gottesdienst"
          },
          {
            "ID": 114,
            "type": "category",
            "title": "Predigerkirche (Musik mittendrin)",
            "slug": "predigerkirche-musik-mittendrin"
          }
        ],
        "image": {
          "ID": "4187",
          "type": "attachment",
          "mimeType": "image/jpeg",
          "url": "http://dev.predigern.ch/wordpress/wp-content/uploads/2015/11/schneekristalle-e1447965707151.jpg?1469370784",
          "thumbnailUrl": "http://dev.predigern.ch/wordpress/wp-content/uploads/2015/11/schneekristalle-e1447965707151-150x150.jpg"
        },
        "index": 0
      }
    ],
    "details": [
      {
        "_compability": "needed for V1.0"
      }
    ]
  },
  "param": {
    "terms": [
      2
    ]
  }
}
```