Calendar extension "cz_simple_cal"

Christian Zenker

Calendar extension "cz_simple_cal"

Christian Zenker

Abstract

cz_simple_cal is a simple calendar written on top of extbase

Table of Contents

List of Tables

List of Examples

Chapter 1. Introduction

Caution

This extension in alpha state.

The current version is only compatible with TYPO3 4.5 LTS and the corresponding version of extbase (1.3).
There is a version for TYPO3 4.4 and extbase 1.2 on github in the `extbase-1.2` branch. But this is deprecated, so don't expect any bugfixes or even new features.

What does it do?

This extension provides a simple calendar.

Feature List

- Allday Events
- Recurring Events
- Exceptions, Exception Groups
- Categories
- adding actions via Typoscript
- templates using the hCalendar and hCard microformats
- ...and all the goodness that is Extbase and Fluid.

Sources

Home in forge	http://forge.typo3.org/projects/extension-cz_simple_cal
Distribution in TER	http://typo3.org/extensions/repository/view/cz_simple_cal/current/
Bugtracker	http://forge.typo3.org/projects/extension-cz_simple_cal/issues
Official Git Repository	http://github.com/czenker/cz_simple_cal

Chapter 2. Administration

Concepts

This section tries to explain some of the basic concepts of the calendar.

Event Index

Calendar Base introduced something called *New Recurring Event Model*. This concept was borrowed and applied to all of cz_simple_cal events by default. The index is automatically updated if you modify an event. So depending on how many recurrances and exceptions you've set up, storing might take a while longer.

Note

The extension is smart enough to notify if you changed some values that actually require indexing to run again. So if you only change the title or a description, no indexing is done.

Note

You can use the scheduler extension to re-index all your events. This should be done if you updated Exceptions that are applied to multiple Events, moved records around or if you changed the `recurrenceEnd` setting. The scheduler is a core extension and shipped with TYPO3, but you might have to install it in the extension manager.

Due to the indexing of events you usually deal with EventIndices in your templates. But the objects are smart enough to tunnel unknown methods to the Event they belong to. So you can work with `EventIndices` as if they were Events.

Fake Actions

To make the extension as flexible as possible you can add fake actions to the controllers in your TypoScript.

Fake Actions are actions you can add to a Controller solely through TypoScript.

See HowTo: Add a fake action to learn - guess what - how to add a fake action.

The type `date`

This type is quite heavily used in the extension. It allowes for a very flexible and simple calculation of dates and times. It is based on the english language and international date and time formats.

The recommended way of setting a fixed day and time is the `YYYY-MM-DD (HH:MM:SS[T]?)?` syntax, but there are also different valid syntaxes:

Example 2.1. Examples of valid dates and times

- **`2009-02-13`**
- **`2009-02-13 23:31:30`**
- **`2009-02-13 23:31:30UTC`**
- **`13.2.09 23.31.30+00:00`**
- **`February 13th, 2009 11 pm`**

Note

Note that you can't use localized month names here.

Additionally you can use relative dates and chaining of different relative dates.

Example 2.2. Examples of valid relative dates

- **yesterday**
- **last monday**
- **+1 month -1 day**
- **first day this month|monday this week**

See the appendix for the complete syntax.

Learning by example

We will go through the default template and comment on how and why things work. You'll find the default template at `EXT:cz_simple_cal/Configuration/TypoScript/main/setup.txt`.

```
01 plugin.tx_czsimplecal {
02   #@description: the pid the records are stored in
03   persistence.storagePid = {$plugin.tx_czsimplecal.pidList}
04   # [...]
05   settings {
06     EventIndex {
07       # first one is the default action
08       allowedActions = list,minimonth,day,week,show,next
09       actions {
10         list {
11           defaultPid = {$plugin.tx_czsimplecal.pids.default}
12           startDate  = today
13           endDate    = today +1 month
14           maxEvents = 9999
15           orderBy   = start
16           order     = ASC
17           # filter.categories.uid = 42
18           # excludeOverlongEvents = 0
19           # includeStartedEvents = 0
20         }
21         show {
22           defaultPid = {$plugin.tx_czsimplecal.pids.default}
23         }
24         next {
25           defaultPid = {$plugin.tx_czsimplecal.pids.default}
26           useAction = list
27           startDate  = now
28           endDate    = now +1 month
29           maxEvents = 1
30           orderBy   = start
31           order     = ASC
```

```
32          }
33          week {
34            defaultPid = {$plugin.tx_czsimplecal.pids.default}
35            useAction = list
36            startDate = monday this week
37            endDate   = sunday this week 23:59:59
38            getPostAllowed = getDate
39            maxEvents = 999
40            orderBy = start
41            order   = ASC
42          }
43          minimonth {
44            defaultPid = {$plugin.tx_czsimplecal.pids.default}
45            useAction = countEvents
46            getPostAllowed = getDate
47            startDate  = first day of this month|monday this week
48            endDate    = last day of this month|monday next week -1 second
49            groupBy    = day
50          }
51          day {
52            defaultPid = {$plugin.tx_czsimplecal.pids.default}
53            useAction = list
54            startDate = today
55            endDate   = today 23:59:59
56            getPostAllowed = getDate
57            maxEvents = 999
58            orderBy = start
59            order   = ASC
60          }
61        }
62      }
63      Event {
64        allowedActions = show
65        actions {
66          show {
67            defaultPid = {$plugin.tx_czsimplecal.pids.default}
68          }
69        }
70      }
71      # this is where you can put your customized options
72      custom {
73      }
74    }
75 }
```

Line `03` holds the comma-seperated list of uids of the pages all relevant records are stored. Don't forget to give the pages of your tt_address records if you plan on using them for organizers and locations.

Lines `06` to `62` hold configuration for the `EventIndex`-Controller.

Line `08` lists all the actions of the `EventIndex`-Controller that might be called. You can use this field to disable actions without the need to delete their configuration.
The first named action is automatically the default action. So by setting `allowedAction = minimonth` this page was only able to show the minimonth-view.

Lines 09 to 61 holds the configuration of the different available Views. Each entry defines a View. Its key is the name of the view. In the example we define the list, show, next, minimonth, week and day-Views.

Lines 10 to 20 hold the configuration for the list Action.
The list action is used to show events from a certain timespan, like all events from one day or from one week.

In line 11 the default pid for this action is set. This is only used by the Tx_CzSimpleCal_ViewHelpers_Link_ActionViewHelper to allow easy linking of your whole calender is not shown on only one pid, but split over multiple pages.
If you link to the list View from another view, you don't have to set the pageUid property of the Tx_Fluid_ViewHelpers_Link_ActionViewHelper.

In lines 12 and 13 the start and end dates for the list are defined. In this example the list view shows all events from (midnight) today up to one month from today.

Note

Only the start date of an event is taken into account, when deciding if an event is in that interval by default. So an event, that started yesterday but is still running today, won't be shown in that list. But an event that starts today and runs for the next three months would be displayed.

You can change that behaviour by using excludeOverlongEvents and includeStartedEvents as seen in lines 18 and 19.

Line 14 defines how many events are selected at max.

Lines 15 and 16 define how the selected EventIndices are returned. In this example they are ordered ascending by their start date.

Line 17 shows how you could filter the selected Events by their category.
If you would uncomment that line, you would only select events in the category with the uid 42.

Note

The filter array is quite powerfull. You can also use it to filter by organizer, location or even an events name. See the configuration documentation for more details.

Lines 18 and 19 allow you to also include already started but not yet finished events to your results (includeStartedEvents) as well as exclude events that aren't yet finished on endDate (excludeOverlongEvents).

Lines 21 to 23 hold the configuration for another action - the show action. It is used to show a single *EventIndex*! So if you had a dance course that meets once a week, this action would show the dance cours on Thursday, 13th January at 18:00, for instance. The dance course as a whole could be represented by the Event-Controllers show action.

Lines 24 to 32 define an other action. It is thought to show the next event in the calendar. This is done by utilizing the already known settings startDate, endDate and maxEvents.

But there is a twist to that action: It does not really exist in the EventIndex Cotroller. It's something we call a Fake Action. We define which Real Action this action should inherit from in line 26 with the useAction setting. So this action just behaves as if it was a list action, but with different settings and it uses the next template for its view.

Lines 33 to 42 define another fake action for a week view. The first notable thing to mark, are the startDate and the endDate settings here: You can use some phrases like monday this week as part of the type date. You find a full reference of the date type in the appendix.

Line 38 defines which GET-parameters might override some of the settings of this action. In this case we allow the user to override the getDate setting. getDate is the date we asume to be "now" before calculating the startDate and endDate. So with this setting enabled, we create pages for each week.

In lines 43 to 50 we define an other Fake Action. But this time it inherits from the countEvents action. This action counts all events instead of listing them and groups them by their day (line 49).

Line 48 shows another (rather ugly) statement for the type date. It uses a technique, we call *chaining* (through the pipe character |)to evaluate complex date calculations. The end date is calculated by first finding the first day of this month, then looking for the first monday next week (that is the first monday in the next month) and then subtracting one second.

Lines 63 to 70 hold configuration for the Event-Controller. The configuration is similar to that of the EventIndex controller.

Lines 66 to 68 configurate the only Real action of the Event controller - the show action.

Lines 72 to 73 is the custom-array where you can store whichever values you like. There are no conventions whatsoever.

Templating

As an Extbase extension, cz_simple_cal makes use of the Fluid templating engine.

Here are some resources to get you started with Fluid:

- http://flow3.typo3.org/documentation/manuals/fluid/
- http://www.typovision.de/cheatsheet/
- http://forge.typo3.org/projects/typo3v4-mvc/wiki/Collection_of_Documentation

You find the default templates and partials in typo3conf/ext/cz_simple_cal/Resources/Private/Templates or Resources/Private/Partials respectively.

To customize these templates, first copy the two folders Templates and Partials to a different location, like fileadmin/template/cz_simple_cal/. After that you just have to point Fluid to the new template files by setting the following in your TypoScript:

```
plugin.tx_czsimplecal {
  view.templateRootPath = fileadmin/template/cz_simple_cal/Templates/
  view.partialRootPath = fileadmin/template/cz_simple_cal/Partials/
```

After doing that you are free to modify the templates as desired. The viewHelpers shipped with cz_simple_cal are described in an appendix.

Chapter 3. Configuration

extConf

When installing the extension the Extension Manager will let you set some values.

recurrenceEnd — cz_simple_cal indexes recurring events. Here you can set until what date the indexing should be done. Use any valid format for the `date` type here - `2020-12-31` or `+2 years` could be some.

> **Note**
>
> Don't forget to reindex your events when using a relative date. The index could be outdated else.

TypoScript

You can also configure the extension on page-level using TypoScript. Everything you configure goes into `plugin.tx_czsimplecal`

The following settings are Extbase-specific. If you have worked with Extbase before, you should be quite familiar with them.

persistence.storagePid (type: `string`) — The page id where your records are stored. You can set multiple pages seperated by comma (`,`).

> **Note**
>
> If you use tt_address for location and/or organizer, you also have to give the pids of these records.

view.templateRootPath
view.partialRootPath
(type: `string`) — The path to the folder your customized templates or partials are stored in.

> **Note**
>
> You find the default templates and partials in `typo3conf/ext/cz_simple_cal/Resources/Private/Templates` or `Resources/Private/Partials`.
> If you copied those folders to `fileadmin/template/cz_simple_cal/` the paths to set were `fileadmin/template/cz_simple_cal/Templates/` or `fileadmin/template/cz_simple_cal/Partials/` respectively.

_LOCAL_LANG.[lang].[key]
(type: `string`) — An array were you can override localized strings with your custom substitudes.
`[lang]` is the 2-sign-language code used by TYPO3, like `en` or `de`.
`[key]` is the identifier of the string. For a list of identifiers, have a look at `EXT:cz_simple_cal/Resources/Private/Language/locallang.xml`.

settings
(type: `settings`)

See `settings` for all options.

type `settings`

All cz_simple_cal-configuration goes into `plugin.tx_czsimplecal.settings`.

Note

Thanks to fluid, you'll have every setting available in each fluid-template through the property `{settings}`.

custom
(type: `array`)

This space is reserved for whatever values you'd like to have available in your templates.

Example 3.1. Using the `custom` array

When setting

```
plugin.tx_czsimplecal.settings.custom.foo = bar
```

in TypoScript, the Fluid-template

```
<p>{settings.custom.foo}</p>
```

would print

```
<p>bar</p>
```

in the frontend.

Note

Please note, that this is just a dumb plain array! So it does know nothing of stdWraps and cObjects. If you want to create content using TypoScript objects, use the `Tx_Fluid_ViewHelpers_CObjectViewHelper` instead.

EventIndex
(type: `EventIndex configuration`)

See `EventIndex configuration`.

Event
(type: `Event configuration`)

See `Event configuration`.

`EventIndex configuration`

allowedActions
(type: `comma seperated values`)

A comma seperated list of the actions that are enabled.

Note

The TypoScript functions `addToList` and `removeFromList` could come in handy here. They

allow to add and remove values from a csv-list when not knowing which values are already present.

Example 3.2. Using TypoScript functions

```
plugin.tx_czsimplecal.settings.EventIndex {
  allowedActions = foo,bar
  allowedActions := addToList(baz)
  allowedActions := removeFromList(foo)
}
```

will leave `bar,baz` as value of `allowedActions`.

actions.[action]

Configure your Fake Actions here. There are two properties each configuration understands:

`useAction` is mandatory and names the Real Action to use. You'll find a list of all allowed Real Actions right below.

`defaultPid` is the pid where this action is typically shown. This property is used by a viewHelper to ease linking.

Additional configuration options are available depending on the selected Real Action.

`list` action

This action is used to display a list of timely connected events.

Examples would be to show upcoming events, the last events or all events on a certain month.

startDate
(type: `date`)

The start of the event list.
By default this list contains only events that start *after* this point.

endDate
(type: `date`)

The end of the event list.
By default this list contains only events that end *before* this point.

includeStartedEvents
(type: `boolean`)

When enabled, events that were already started on `startDate` (but not yet finished) are also shown.

excludeOverlongEvents
(type: `boolean`)

When enabled, events that are not yet finished on `endDate` (but already started) are also shown.

order
(type: "`asc`" or "`desc`")

Sort the resulting EventIndexes ascending (`asc`) or descending (`desc`).

orderBy
(type: "`start`" or "`end`")

If the resulting events should be sorted by their `start` or `end` date.

maxEvents
(type: `positive integer`)

The maximum of events to return.

filter.[field]
(type: `array`)

Filter the resulting events by a value from the database. You can assign multiple values seperated by comma.

If you give multiple fields, all conditions must be fullfilled for a record to show up ("`AND`" when speaking in terms of SQL).

Example 3.3. Usage of filters

`filter.categories.uid = 42`

selects only events of the category with uid `42`.

`filter.organizer.uid = 4,2`

selects only events of the organizers with uids `4` and `2`.

```
filter.categories.uid = 4
filter.location.uid = 2
```

selects only events of the organizer with uid `4` at the location with uid `2`.

getDate
(type: `date`)

The date given here will be asumed to be `now` for relative dates in `startDate` and `endDate`.

getPostAllowed
(type: `comma seperated values`)

Comma seperated names of the above settings that are allowed to be overriden by GET and POST variables.

`countEvents` action

This action is used to display a number of available request without giving any information on the events.

An example would be the usage for a minicalender where you show the numbers of events of a day.

startDate
(type: `date`)

The start of the event list.
By default this list contains only events that start *after* this point.

endDate
(type: `date`)

The end of the event list.
By default this list contains only events that end *before* this point.

includeStartedEvents
(type: `boolean`)

When enabled, events that were already started on `startDate` (but not yet finished) are also shown.

excludeOverlongEvents
(type: `boolean`)

When enabled, events that are not yet finished on `endDate` (but already started) are also shown.

order
(type: "`asc`" or "`desc`")

Sort the resulting EventIndexes ascending (`asc`) or descending (`desc`).

orderBy
(type: "`start`" or "`end`")

If the resulting events should be sorted by their `start` or `end` date.

maxEvents
(type: `positive integer`)

The maximum of events to return.

filter.[field]
(type: `array`)

Filter the resulting events by a value from the database. You can assign multiple values seperated by comma.

If you give multiple fields, all conditions must be fullfilled for a record to show up ("`AND`" when speaking in terms of SQL).

Example 3.4. Usage of filters

```
filter.categories.uid = 42
```

selects only events of the category with uid 42.

```
filter.organizer.uid = 4,2
```

selects only events of the organizers with uids 4 and 2.

```
filter.categories.uid = 4
filter.location.uid = 2
```

selects only events of the organizer with uid 4 at the location with uid 2.

groupBy	The timespan for which to group the events. Allowed values are `day`, `week`, `month` and `year`.
getDate (type: `date`)	The date given here will be asumed to be `now` for relative dates in `startDate` and `endDate`.
getPostAllowed (type: `comma seperated values`)	Comma seperated names of the above settings that are allowed to be overriden by GET and POST variables.

`show` action

This action is just showing an EventIndex.

To explain the difference between the `EventIndex` and the `Event` action `show`:

If you'd try to model a dancing class that meets once a week for three months, the `Event` action `show` would represent the whole dancing class, an `EventIndex` action would, for instance, represent the dancing hour on Thursday, 13th of January at 18:00.

No further configuration available.

`Event configuration`

allowedActions (type: `comma seperated values`)	A comma seperated list of the actions that are enabled.

Note

The TypoScript functions `addToList` and `removeFromList` could come in handy here. They allow to add and remove values from a csv-list when not knowing which values are already present.

Example 3.5. Using TypoScript functions

```
plugin.tx_czsimplecal.settings.EventIndex {
  allowedActions = foo,bar
  allowedActions := addToList(baz)
  allowedActions := removeFromList(foo)
}
```

will leave `bar,baz` as value of `allowedActions`.

actions.[action]

Configure your Fake Actions here. There are two properties each configuration understands:

`useAction` is mandatory and names the Real Action to use. You'll find a list of all allowed Real Actions right below.

`defaultPid` is the pid where this action is typically shown. This property is used by a viewHelper to ease linking.

Additional configuration options are available depending on the selected Real Action.

show action

This action is just showing a single event.

getDate
(type: `date`)

The date given here will be asumed to be `now` for relative dates in `startDate` and `endDate`.

getPostAllowed
(type: `comma seperated values`)

Comma seperated names of the above settings that are allowed to be overriden by GET and POST variables.

Chapter 4. Scheduler

Understanding the index task

The purpose of the index task is to create an index of all events as described in the basic concepts.

But you usually don't have to run the scheduler permanently to keep the index up to date. The extension makes sure, that the index is up to date, when you create, edit, move or delete events.

Some of the cases where you actually need to update the index include:

- Events where not added through TYPO3s native forms.

  This might happen if you use any weired extension, that manipulates records on a database level or if you even manipulate the records on the database yourself.

- You have recurring events and use `recurranceEnd` with a relative date format.

  I would recommend this way as it keeps your database tables as small as possible and you probably don't want recurring events to repeat until ten years from now. If you had limited `recurranceEnd` to **`+1 year`** you should update your index every once in a while as the event index will only have been created for one year from your last editing of the event. If you would not run the scheduler, your event would "disappear" one year after.

So, to comprehend: You don't need to set up a scheduler task if you do not use recurring events or only use them with `recurrance until` for a short time.

index task configuration

minimum age for reindexing

An event will only be reindexed if it was indexed prior to the date given here.

This field is of the `date` type. So if you enter **`-1 month`** all the events will be re-indexed after one month.

Note

Depending on how many events you have, the indexer might be to slow to index all events in one run. So your scheduler should run more often than the value in this property, if you can't ensure that your machine is capable of indexing all events at once.

If you leave this field empty the events that where indexed the longest time ago will be indexed. The process will run until all events where indexed or the script stops due to missing memory or time.

Note

Needless to say it is not recommended to run this task every 5 minutes. But this comes in handy if you need to reindex your events manually.

Chapter 5. HowTo's

Add a fake action

Adding a fake action is pretty simple and can be done only using TypoScript. Let's say we want to add a view that displays the event that has recently finished. We'll call this action `recent`.

1. **Choose a fitting real action**

   In our example case this would be the `listAction`. `ShowAction` won't fit as you don't know the id of the event to display in advance. Instead, we'll limit the list to display only one event.

2. **Extend the TypoScript configuration**

   > ### Note
   >
   > All given TypoScript paths are relative to `plugin.tx_czsimplecal`. I am to lazy to add this to the path each time, and so should you. Use the curly brackets: `plugin.tx_czsimplecal{ //... }`

   The TypoScript settings already hold some action configurations. So we'll just copy the configuration for the `listAction` in `settings.EventIndex.actions.list` to `settings.EventIndex.actions.recent`.

   The configureable options should be pretty obvious. But first we'll add `useAction = list` to the actions configuration. This way the fake `recentAction` knows which real action to call. Now we can change the other configuration like this:

   ```
   recent {
       useAction  = list
       startDate  = now -1 month
       endDate    = now

       maxEvents = 1
       orderBy   = end
       order     = DESC
   }
   ```

   Guess what each of the configuration values is doing. ;) We select all events between now and one month ago, order them by their endtime and just pick the first one.

3. **Add the action to the allowed actions**

   Just add the name of the action to the `allowedActions` in `settins.EventIndex`.

4. **Create a view**

   Well, thats even simpler: Just copy the `list.html` template and rename it to `recent.html`. Do changes on the file if you like - removing the dayWrapper could be a good idea.

Chapter 6. Known problems

- Only compatible with Extbase 1.3 (shipped with TYPO3 4.5 LTS).

  ### Note

  There is a version for TYPO3 4.4 and Extbase 1.2 in the github repository on the `extbase-1.2` branch. But this branch is deprecated. There might be some bugfixes to that branch, but don't expect them or any new features.

- Updating problems due to the development in progress of extbase might occur.

  As extbase is still under heavy development and this extension uses some methods that are not marked as public API, there might be problems when updating your Extbase installation. We hope to fix them as soon as possible, but you should be aware of that issue.

Any known bugs can be found in the official bugtracker for cz_simple_cal [http://forge.typo3.org/projects/extension-cz_simple_cal/issues].

Chapter 7. FAQs

7.1. Why isn't [name any feature here] implemented?

Most likely because no one has needed it yet.

Following the *YAGNI* (= "You Ain't Gonna Need It") principle of programming, features are implemented as soon as they are needed. If you miss a feature, it might be because no one has programmed or at least requested it.

The best way to get your feature added is by opening a ticket, doing the coding and submitting your work.
If you are not a coder please submit a feature request to the bugtracker. If you find, someone else has already requested that feature, please comment on it and state that you'd like to have that feature, too. This way we can easily decide which are the most desired features.
And last but not least you could consider sponsoring the development of a feature. Get in touch with us if you wish to do so.

7.2. I'm a programmer, how can I dive into the code?

You should be familiar with Extbase and Fluid. The source code itself is documented inline, so there should be a brief explanation on what each class and method is used for. Read the according section to understand the basic concepts and philosophies of the extension.

If you have any difficulties on understanding the code, you can contact the developer through the email-adressen given in TYPO3 Forge.

Chapter 8. Copyright Notice

This project uses some of the Fugue Icons [http://p.yusukekamiyamane.com/]. They are published under a Creative Commons [http://creativecommons.org/licenses/by/3.0] license:

The documentation on the date type in the appendix is a slight modification of the official PHP documentation which is licensed under a Creative Commons [http://creativecommons.org/licenses/by/3.0] license.

Glossary

Date	A date usually means the combination of day and time. See Also Day, Time.
Day	When speaking of a *day* usually no time is meant. For example 1st January 2010 would be a day. See Also Date, Time.
Time	When speaking of a *time* usually no day is meant. For example 12:34:56 would be a time. See Also Date, Day.
Event (Domain Object)	The Domain Object `Event` represents a series of events that share some common information like the name or a description. Events might be recurrant or have exceptions in this recurrances. See Also EventIndex (Domain Object).
Event (Controller)	The most important controller for the Events. Technically it is no controller for the `Event` but for the `EventIndex`
EventIndex (Domain Object)	In contrast to the Event an `EventIndex` is a representation of a concrete occurance of the event. So an `Event` the recurrs every week will have a `EventIndex` representation fore every week. Even not recurring Events have an `EventIndex` representation. Queries on several events are almost exclusivly done on these domain objects. See Also Event.
Exception (Domain Object)	An `Exception` is an "`Event`" that symbolizes that an Event is not taking place when the exception is active. It might be recurring, but Exceptions is not stored are not stored as Indices in the database as it is done with Events.
ExceptionGroup (Domain Object)	A collection of `Exceptions` that belong together somehow.
GetDate	GetDate is a concept taken from the TYPO3 extension cal. GetDate makes some actions configurable using GET-parameters. All relative dates of the action are calculated based on that date.
Timespan	A timespan has a start and an end date and covers everything in between. There are no gaps in a timespan.
Timeline	A timeline is a collection of timespans. The contained timespans might overlap or build gaps. See Also Timespan.
Fake Action	One of the concepts of this calendar is to generate actions dynamically based on TypoScript configuration. Actions that have no method in the corresponding controller are called "fake actions". See Also Real Action.
Real Action	In comparisson to fake actions the real actions have a method in the corresponding controller. These are the actions as they are conceptually intended by extbase. See Also Fake Action.

Appendix A. The type `date`

The first thing to mention is that you can *only* use english phrases and month names. Numeric formats are usually standartized formats.

Note

The type `date` is based on PHP's date and time formats [http://php.net/manual/en/datetime.formats.php]. With some exceptions all of the formats there can be used. Some of the relative formats require PHP 5.3, so you should avoid them if possible to keep compatibility.

Date Formats

Table A.1. Used Symbols

Description	Format	Examples
`daysuf`	"st" | "nd" | "rd" | "th"	
`dd`	([0-2]?[0-9] | "3"[01]) `daysuf`?	"7th", "22nd", "31"
`DD`	"0" [0-9] | [1-2][0-9] | "3" [01]	"07", "31"
`m`	'january' | 'february' | 'march' | 'april' | 'may' | 'june' | 'july' | 'august' | 'september' | 'october' | 'november' | 'december' | 'jan' | 'feb' | 'mar' | 'apr' | 'may' | 'jun' | 'jul' | 'aug' | 'sep' | 'sept' | 'oct' | 'nov' | 'dec' | "I" | "II" | "III" | "IV" | "V" | "VI" | "VII" | "VIII" | "IX" | "X" | "XI" | "XII"	
`M`	'jan' | 'feb' | 'mar' | 'apr' | 'may' | 'jun' | 'jul' | 'aug' | 'sep' | 'sept' | 'oct' | 'nov' | 'dec'	
`mm`	"0"? [0-9] | "1"[0-2]	"0", "04", "7", "12"
`MM`	"0" [0-9] | "1"[0-2]	"00", "04", "07", "12"
`y`	[0-9]{1,4}	"00", "78", "08", "8", "2008"
`yy`	[0-9]{2}	"00", "08", "78"
`YY`	[0-9]{4}	"2000", "2008", "1978"

Table A.2. Localized Notations

Description	Format	Examples
American month and day	`mm` "/" `dd`	"5/12", "10/27"
American month, day and year	`mm` "/" `dd` "/" `y`	"12/22/78", "1/17/2006", "1/17/6"
Four digit year, month and day with slashes	`YY` "/" `mm` "/" `dd`	"2008/6/30", "1978/12/22"
Four digit year and month (GNU)	`YY` "-" `mm`	"2008-6", "2008-06", "1978-12"

Description	Format	Examples
Year, month and day with dashes	y "-" mm "-" dd	"2008-6-30", "78-12-22", "8-6-21"
Day, month and four digit year, with dots, tabs or dashes	dd [.\t-] mm [.-] YY	"30-6-2008", "22.12\t1978"
Day, month and two digit year, with dots or tabs	dd [.\t] mm "." yy	"30.6.08", "22\t12\t78"
Day, textual month and year	dd ([\t.-])* m ([\t.-])* y	"30-June 2008", "22DEC78", "14 III 1879"
Textual month and four digit year (Day reset to 1)	m ([\t.-])* YY	"June 2008", "DEC1978", "March 1879"
Four digit year and textual month (Day reset to 1)	YY ([\t.-])* m	"2008 June", "1978-XII", "1879.MArCH"
Textual month, day and year	m ([.\t-])* dd [,.stndrh\t]+ y	"July 1st, 2008", "April 17, 1790", "May.9,78"
Textual month and day	m ([.\t-])* dd [,.stndrh\t]*	"July 1st,", "Apr 17", "May.9"
Day and textual month	d ([.\t-])* m	"1 July", "17 Apr", "9.May"
Month abbreviation, day and year	M "-" DD "-" y	"May-09-78", "Apr-17-1790"
Year, month abbreviation and day	y "-" M "-" DD	"78-Dec-22", "1814-MAY-17"
Year (and just the year)	YY	"1978", "2008"
Textual month (and just the month)	m	"March", "jun", "DEC"

Table A.3. ISO8601 Notations

Description	Format	Examples
Eight digit year, month and day	YY MM DD	"15810726", "19780417", "18140517"
Four digit year, month and day with slashes	YY "/" MM "/" DD	"2008/06/30", "1978/12/22"
Two digit year, month and day with dashes	yy "-" MM "-" DD	"08-06-30", "78-12-22"
Four digit year with optional sign, month and day	[+-]? YY "-" MM "-" DD	"-0002-07-26", "+1978-04-17", "1814-05-17"

Note

For the y and yy formats, years below 100 are handled in a special way when the y or yy symbol is used. If the year falls in the range 0 (inclusive) to 69 (inclusive), 2000 is added. If the year falls in the range 70 (inclusive) to 99 (inclusive) then 1900 is added. This means that "00-01-01" is interpreted as "2000-01-01".

Note

The "Day, month and two digit year, with dots or tabs" format (dd [.\t] mm "." yy) only works for the year values 61 (inclusive) to 99 (inclusive) - outside those years the *time format* "HH [.:] MM [.:] SS" has precedence.

Note

The "Year (and just the year)" format only works if a time string has already been found -- otherwise this format is recognised as HH MM.

Note

It is possible to over- and underflow the dd and DD format. Day 0 means the last day of previous month, whereas overflows count into the next month. This makes "2008-08-00" equivalent to "2008-07-31" and "2008-06-31" equivalent to "2008-07-01" (June only has 30 days).

It is also possible to underflow the mm and MM formats with the value 0. A month value of 0 means December of the previous year. As example "2008-00-22" is equivalent to "2007-12-22".

If you combine the previous two facts and underflow both the day and the month, the following happens: "2008-00-00" first gets converted to "2007-12-00" which then gets converted to "2007-11-30". This also happens with the string "0000-00-00", which gets transformed into "-0001-11-30" (the year -1 in the ISO 8601 calendar, which is 2 BC in the proleptic Gregorian calendar).

Time Formats

Table A.4. Used Symbols

Description	Formats	Examples
frac	. [0-9]+	".21342", ".85"
hh	"0"?[1-9] | "1"[0-2]	"04", "7", "12"
HH	[01][0-9] | "2"[0-4]	"04", "7", "19"
meridian	[AaPp] .? [Mm] .? [\0\t]	"A.m.", "pM", "am."
MM	[0-5][0-9]	"00", "12", "59"
II	[0-5][0-9]	"00", "12", "59"
space	[\t]	
tz	("? [A-Za-z]{1,6} ")"? | [A-Z][a-z]+([_/][A-Z][a-z]+)+	"CEST", "Europe/Amsterdam", "America/Indiana/Knox"
tzcorrection	"GMT"? [+-] hh ":"? MM?	"+0400", "GMT-07:00", "-07:00"

Table A.5. 12 Hour Notation

Description	Format	Examples
Hour only, with meridian	hh space? meridian	"4 am", "5PM"
Hour and minutes, with meridian	hh [.:] MM space? meridian	"4:08 am", "7:19P.M."
Hour, minutes and seconds, with meridian	hh [.:] MM [.:] II space? meridian	"4:08:37 am", "7:19:19P.M."
MS SQL (Hour, minutes, seconds and fraction with meridian), PHP 5.3 and later only	hh ":" MM ":" II [.:] [0-9]+ meridian	"4:08:39:12313am"

Table A.6. 24 Hour Notation

Description	Format	Examples
Hour and minutes	't'? HH [.:] MM	"04:08", "19.19", "T23:43"
Hour and minutes, no colon	't'? HH MM	"0408", "t1919", "T2343"
Hour, minutes and seconds	't'? HH [.:] MM [.:] II	"04.08.37", "t19:19:19"
Hour, minutes and seconds, no colon	't'? HH MM II	"040837", "T191919"
Hour, minutes, seconds and timezone	't'? HH [.:] MM [.:] II space? (tzcorrection | tz)	"040837CEST", "T191919-0700"
Hour, minutes, seconds and fraction	't'? HH [.:] MM [.:] II frac	"04.08.37.81412", "19:19:19.532453"
Time zone information	tz | tzcorrection	"CEST", "Europe/Amsterdam", "+0430", "GMT-06:00"

Compound Formats

Table A.7. Used Symbols

Description	Formats	Examples
DD	"0" [0-9] | [1-2][0-9] | "3" [01]	"02", "12", "31"
doy	"00"[1-9] | "0"[1-9][0-9] | [1-2][0-9][0-9] | "3"[0-5][0-9] | "36"[0-6]	"36"[0-6] "000", "012", "366"
frac	. [0-9]+	".21342", ".85"
hh	"0"?[1-9] | "1"[0-2]	"04", "7", "12"
HH	[01][0-9] | "2"[0-4]	"04", "7", "19"
meridian	[AaPp] .? [Mm] .? [\0\t]	"A.m.", "pM", "am."
ii	[0-5][0-9]	"04", "8", "59"
II	[0-5][0-9]	"04", "08", "59"
M	'jan' | 'feb' | 'mar' | 'apr' | 'may' | 'jun' | 'jul' | 'aug' | 'sep' | 'sept' | 'oct' | 'nov' | 'dec'	
MM	[0-5][0-9]	"00", "12", "59"
space	[\t]	
ss	[0-5][0-9]	"04", "8", "59"
SS	[0-5][0-9]	"04", "08", "59"
W	"0"[1-9] | [1-4][0-9] | "5"[0-3]	"05", "17", "53"
tzcorrection	"GMT"? [+-] hh ":"? MM?	"+0400", "GMT-07:00", "-07:00"
YY	[0-9]{4}	"2000", "2008", "1978"

Table A.8. Localized Notations

Description	Format	Examples
Common Log Format	dd "/" M "/" YY : HH ":" II ":" SS space tzcorrection	"10/Oct/2000:13:55:36 -0700"
EXIF	YY ":" MM ":" DD " " HH ":" II ":" SS	"2008:08:07 18:11:31"
ISO year with ISO week	YY "-"? "W" W	"2008W27", "2008-W28"
ISO year with ISO week and day	YY "-"? "W" W "-"? [0-7]	"2008W273", "2008-W28-3"
PostgreSQL: Year with day-of-year	YY "."? doy	"2008.197", "2008197"
SOAP	YY "-" MM "-" DD "T" HH ":" II ":" SS frac tzcorrection?	"2008-07-01T22:35:17.02", "2008-07-01T22:35:17.03+08:00"
Unix Timestamp	"@" "-"? [0-9]+	"@1215282385"
XMLRPC	YY MM DD "T" hh ":" II ":" SS	"20080701T22:38:07", "20080701T9:38:07"
XMLRPC (Compact)	YY MM DD 't' hh II SS	"20080701t223807", "20080701T093807"
WDDX	YY "-" mm "-" dd "T" hh ":" ii ":" ss	"2008-7-1T9:3:37"

Note

The "W" in the "ISO year with ISO week" and "ISO year with ISO week day day" formats is case-sensitive, you can only use the upper case "W".

The "T" in the SOAP, XMRPC and WDDX formats is case-sensitive, you can only use the upper case "T".

Relative Formats

Table A.9. Used Symbols

Description	Format
dayname	'sunday' | 'monday' | 'tuesday' | 'wednesday' | 'thursday' | 'friday' | 'saturday' | 'sun' | 'mon' | 'tue' | 'wed' | 'thu' | 'fri' | 'sat' | 'sun'
number	[+-]?[0-9]+
reltext	'next' | 'last' | 'previous' | 'this'
space	[\t]+
unit	(('sec' | 'second' | 'min' | 'minute' | 'hour' | 'day' | 'month' | 'year') 's'?) | 'weeks' | daytext

Table A.10. Day-based Notations

Format	Description	Examples
'yesterday'	Midnight of yesterday	"yesterday 14:00"

Format	Description	Examples
'midnight'	The time is set to 00:00:00	
'today'	The time is set to 00:00:00	
'now'	Now - this is simply ignored	
'noon'	The time is set to 12:00:00	"yesterday noon"
'tomorrow'	Midnight of tomorrow	
'first day' ' of'?	Sets the day of the first of the current month. This phrase is best used together with a month name following it.	"first day of January 2008"
'last day' ' of'?	Sets the day to the last day of the current month. This phrase is best used together with a month name following it.	"last day of next month"
'last' `space dayname space` 'of'	Calculates the *last* week day of the current month.	"last sat of July 2008"
`number space ? ( unit |` 'week')	Handles relative time items where the value is a number.	"+5 weeks", "12 day", "-7 weekdays"
`reltext space` 'week'	Handles the special format "weekday + last/this/next week".	"Monday next week"

Caution

Weeks always start with mondays. That's a difference to the original DateTime starts with sundays.

Chaining

All afore mentioned formats can be chained by using the pipe character (`|`). All rules will be applied one after another.

Example A.1. Chaining of DateTime formats

- **`first day this month|monday this week`**
- **`2009-02-13 00:00:00|sunday next week`**

Appendix B. ViewHelpers

Tx_CzSimpleCal_ViewHelpers_Array_JoinViewHelper

Join multiple values from an array into a string (kind of PHP's `implode()`).

You might use the item property as well as the `Tx_CzSimpleCal_ViewHelpers_Array_JoinItemViewHelper` to give the items to be joined.

items (type: `array`)	an array of strings that need to be joined
by (type: `string`)	the string used to glue the items together
removeEmpty (type: `boolean`)	if true, empty items will be removed

Example B.1. Using `Tx_CzSimpleCal_ViewHelpers_Array_JoinViewHelper` and `Tx_CzSimpleCal_ViewHelpers_Array_JoinItemViewHelper`

```
<cal:array.join>
  <cal.array.joinItem>foo</cal.array.joinItem>
  <cal.array.joinItem>bar</cal.array.joinItem>
  <cal.array.joinItem>baz</cal.array.joinItem>
</cal:array.join>
```

renders as `foo, bar, baz`.

Example B.2. Using `Tx_CzSimpleCal_ViewHelpers_Array_JoinViewHelper` shorthand-syntax

```
<cal:array.join items="{0:'foo', 1:'bar', 2:'baz'}" by=", " />
```

renders as `foo, bar, baz`.

Tx_CzSimpleCal_ViewHelpers_Calendar_CreateDateTimeViewHelper

create a `Tx_CzSimpleCal_Utility_DateTime` object.

dateTime (type: `date`)	some string of the type `date`

Example B.3. Using `Tx_CzSimpleCal_ViewHelpers_Array_JoinViewHelper` and `Tx_CzSimpleCal_ViewHelpers_Array_JoinItemViewHelper`

```
<f:map alias="foo:{cal:calendar.dateTime(dateTime:'now')}">
  <f:debug>{foo}</f:debug>
</f:map>
```

`Tx_CzSimpleCal_ViewHelpers_Calendar_OnNewDayViewHelper`

renders its content if the submitted event is on a different date then the previous one

event (type: `Tx_CzSimpleCal_Domain_Model_EventIndexer`)	the event to compare to the previously submitted one
label (type: `string`)	if you need multiple irrelated instances set this to something unique

Example B.4. Using `Tx_CzSimpleCal_ViewHelpers_Calendar_OnNewDayViewHelper`

```
<f:for each="{events}" as="event">
  <cal:calendar.onNewDay event="{event}">
    Good morning. This is a new day.
  </cal:calendar.onNewDay>
  {event.title}
</f:for>
```

`Tx_CzSimpleCal_ViewHelpers_Condition_CompareViewHelper`

Compare two values. Best used in conjunction with `Tx_Fluid_ViewHelpers_IfViewHelper`.

value1 (type: `mixed`)	first value
value2 (type: `mixed`)	second value
operation (type: `string`)	The following operations are supported: `=`, `==` (default) — check if both values are equal (integer 10 would be equal to string "10"). `===` — check if both values are identical (integer 10 would *not* be equal to string "10") `!=`, `<>` — check if both values are not equal

`!==`	check if both values are not equal and do an additional type check
`>`	check if the first value is larger than the second one
`>=`, `=>`	check if the first value is larger or equal than the second one
`<`	check if the first value is smaller than the second one
`<=`, `=<`	check if the first value is smaller or equal than the second one

Example B.5. Usage of `Tx_CzSimpleCal_ViewHelpers_Condition_CompareViewHelper`

```
<f:if
  condition="{x:condition.compare(value1: 10, value2: 10)}">
  Both values are equal</f:if>
<f:if
  condition="{x:condition.compare(value1: 10, value2: '10', operation:'=')}">
  Both values are equal</f:if>
<f:if
  condition="{x:condition.compare(value1: 10, value2: '10', operation:'===')}">
  Both values are equal</f:if>
<f:if
  condition="{x:condition.compare(value1: person.age, value2: 18, operation='&lt;'
  You are too young</f:if>
```

The first expression would be true, as `=` is the default comparison.

The second expression would be true as the integer `10` is equal to the string "`10`" in PHP.

The third expression would be false as the types (integer and string) won't match.

The fourth expression would evaluate depending on the value of the `age` property of the `person` object.

`Tx_CzSimpleCal_ViewHelpers_Condition_OneNotEmptyViewHelper`

A view helper to return true if one of the values is not empty.

values (type: `array`)	the values

Example B.6. Usage of `Tx_CzSimpleCal_ViewHelpers_Condition_OneNotEmptyViewHelper`

```
<f:if condition="{x:condition.oneNotEmpty(0:'', 1:0, 2:{})}">
  Hello World</f:if>
<f:if condition="{x:condition.oneNotEmpty(0:foo.bar, 1:foo.baz)}">
  {foo.bar} {foo.baz}</f:if>
```

The first expression would evaluate to false as all these given values are considered empty.

The second expression evaluates to true if eiter `foo.bar` or `foo.baz` returns a non-empty value.

`Tx_CzSimpleCal_ViewHelpers_Format_DateTimeViewHelper`

Formats a unix timestamp or `DateTime` object to a human-readable, localized string.

timestamp
(type: `integer` | `date` | `DateTime`)

This might be either

- a unix timestamp
- a `DateTime` object (this includes `Tx_CzSimpleCal_Utility_DateTime`)
- some string of the type `date`

format
(type: `string`

Formatting string to be parsed by `strftime()`. See the PHP documentation on `strftime()` [http://www.php.net/manual/en/function.strftime.php] for details.

> **Warning**
>
> The Fluid `Tx_Fluid_ViewHelpers_Format_DateViewHelper` uses a different syntax for the `format` property.
>
> That is because we use a different PHP function to format the date and time as this function has the advantage of using localized names for months and weekdays.

get
(type: `date`

Get some related date.

You should usually use a relative format here. This is applied to the date given in `timestamp`.

Example B.7. Using `Tx_CzSimpleCal_ViewHelpers_Format_DateTimeViewHelper`

```
<cal:format.dateTime timestamp="1234567890" /></programlisting>
<cal:format.dateTime timestamp="2009-02-13 20:31:30GMT" />
<cal:format.dateTime timestamp="dateTimeObject" />
<cal:format.dateTime format="%a, %e. %B %G" timestamp="1234567890" />
<cal:format.dateTime timestamp="1234567890" get="+1 day"/>
<cal:format.dateTime timestamp="1234567890" get="first of this month"/>
```

The first two examples would output `2009-02-13` as this is the default formatting option.

The third example would output the date in the `YYYY-MM-DD` format of the given `DateTime` object.

The fourth example would output `Fre, 13. Februar 2009` for german localization.

The fifth example outputs `2009-02-14`.

The sixth example outputs `2009-02-01`.

`Tx_CzSimpleCal_ViewHelpers_Format_NumberChoiceViewHelper`

Renders a string based on a given number.

Usefull for localization of singular and plural and many other things.

number (type: `integer`)	The number that determines which text to use.
format (type: `numberChoice format`)	The `numberChoice format` will be explained in more detail a few lines below.
arguments (type: `array`)	Values for the markerst. The array key is the name of the marker, its value the value to substitude.

The `numberChoice format`

Note

The syntax is identical to the `format_number_choice()` helper of symfony - and they took the class from the PRADO project. If you are familiar with either, there is nothing new to learn for you here.

The basic idea is to define intervals of numbers and a corresponding string to use. Intervals are closed when using square brackets (`[` and `]`) meaning, they include the given number or open when using round brackets (`(` and `)`) meaning, they exclude the given number.

Example B.8. Examples for intervals

- `[0]` would match 0.
- `[0,1]` would match 0 and 1.
- `[0,2)` would match 0 and 1, not 2.
- `[0,+Inf]` would match every non-negative number.
- `(0,+Inf]` would match every positive number.
- `[-Inf,+Inf]` would match any number

You can combine different conditions using the pipe (`|`. The conditions are parsed from left to right using the first matching, so your conditions won't have to be distinct, although it is considered a good practice.

Example B.9. Examples for `numberChoice format`

- `[0] no eggs|[1,+Inf] there are eggs`
- `[0] no eggs|[1,12) some eggs|[12] one dozen eggs|(12,+Inf]lots of eggs`

And of course you can use placeholders using the `###foobar###` syntax. Those placeholders are substituted with the settings used in `arguments`.

Example B.10. Example for `numberChoice format` with placeholder

```
[0] no eggs|[1] 1 egg|[2,+Inf] ###number### eggs
```

Example B.11. Basic usage of `Tx_CzSimpleCal_ViewHelpers_Format_NumberChoiceViewHelper`

```
<f:format.numberChoice text="[0] no eggs|[1,+Inf] eggs" number="1"/>
<f:format.numberChoice number="1">
  [0] no eggs|[1,+Inf] eggs
</f:format.numberChoice>
```

Both of these version are interchangeable and would output `eggs`.

Example B.12. Using `Tx_CzSimpleCal_ViewHelpers_Format_NumberChoiceViewHelper` with placeholders

```
<f:format.numberChoice number="42" arguments="{number:42}">
  [0] no eggs|[1] 1 egg|[2,+Inf] ###number### eggs
</f:format.numberChoice>
```

will output `42 eggs`.

Example B.13. Using `Tx_CzSimpleCal_ViewHelpers_Format_NumberChoiceViewHelper` with localization

```
<f:format.numberChoice number="42" arguments="{number:42}">
  <f:translate key="foobar">
</f:format.numberChoice>
```

Tx_CzSimpleCal_ViewHelpers_Format_TimespanToWordsViewHelper

Renders a readable version for a timespan for days with as little repetition as possible.

start (type: `Tx_CzSimpleCal_Utility_DateTime`)	the start day
end (type: `Tx_CzSimpleCal_Utility_DateTime`)	the end day

Example B.14. Example using `Tx_CzSimpleCal_ViewHelpers_Format_TimespanToWordsViewHelper`

```
<cal:format.timespan start="{christmasEve2010}" end="{newYearsEve2010}" />
```

outputs `Dec 24 to 31, 2010`.

Tx_CzSimpleCal_ViewHelpers_Link_ActionViewHelper

Identical to the `Tx_Fluid_ViewHelpers_Link_ActionViewHelper`, but it determines the correct pageUid to use based on the actions configuration in TypoScript.