1. Basic Usage

The first step in embedding a calendar on a web page is to have the right JavaScript and CSS files. Make sure you are including the FullCalendar stylesheet, as well as the FullCalendar and jQuery JS files, in the `<head>` of your page:

```
<link rel='stylesheet' type='text/css' href='fullcalendar.css' />

<script type='text/javascript' src='jquery.js'></script>

<script type='text/javascript' src='fullcalendar.js'></script>
```

If you plan on doing dragging or resizing, you need some additional jQuery UI files (more information here).

Once you have your dependencies, you need to write the JavaScript code that initializes the calendar. This code must be executed *after* the page has initialized. The best way to do this is with jQuery's `$(document).ready` like so:

```
$(document).ready(function() {



    // page is now ready, initialize the calendar...



    $('#calendar').fullCalendar({

        // put your options and callbacks here

    })



});
```

The above code should be in a `<script>` tag in the head of your page. The code relies on there being an element with an id of "calendar" in the body of your page. The calendar will be placed *inside* this div:

```
<div id='calendar'></div>
```

An that's it, you should see a month-based calendar that has no events on it. If you want to learn how to display events, visit the Google Calendar or Event Datasections.

Options

Most of FullCalendar's documentation describes options that affect the look or behavior of the calendar. Options are usually set when the calendar is initialized, like so:

```
$('#calendar').fullCalendar({

    weekends: false // will hide Saturdays and Sundays

});
```

Callbacks

Callbacks are sort of like options, but they are *functions* that get *called* whenever something special happens. In the following example, an alert box will appear whenever the user clicks on a day:

```
$('#calendar').fullCalendar({

    dayClick: function() {

        alert('a day has been clicked!');

    }

});
```

Methods

Methods provide ways to manipulate the calendar from JavaScript code. A method operates on the jQuery object of a calendar that has already been initialized, using the familiar `fullCalendar` command, but in a completely different way:

```
$('#calendar').fullCalendar('next');
```

This will call the next method and will force to the calendar to move to the next month/week/day.

2. General Display

header **1.3**

Defines the buttons and title at the top of the calendar.

`Object/false, default:`

```
{
    left:   'title',
    center: '',
    right:  'today prev,next'
}
```

Setting the header options to `false` will display no header. An object can be supplied with properties `left`, `center`, and `right`. These properties contain strings with comma/space separated values. Values separated by a comma will be displayed adjacently. Values separated by a space will be displayed with a small gap in between. Strings can contain any of the following values:

title

text containing the current month/week/day

prev

button for moving the calendar back one month/week/day

next

button for moving the calendar forward one month/week/day

prevYear

button for moving the calendar back on year

nextYear

button for moving the calendar forward one year

today

button for moving the calendar to the current month/week/day

a view name

button that will switch the calendar to any of the Available Views

Specifying an empty string for a property will cause it display no text/buttons.

theme 1.3

Enables/disables use of jQuery UI theming.

`Boolean, default: false`

Once you enable theming with `true`, you still need to include the CSS file for the theme you want. For example, if you just downloaded a theme from the jQuery UI Themeroller, you need to put a `<link>` tag in your page's `<head>`.

buttonIcons 1.3

Determines which theme icons appear on the header buttons.

`Object, default:`

```
{
    prev: 'circle-triangle-w',
    next: 'circle-triangle-e'
}
```

This option only applies to calendars that have jQuery UI theming enabled with thetheme option.

A hash must be supplied that maps button names (from the header) to icon strings. The icon strings determine the CSS class that will be used on the button. For example, the string `'circle-triangle-w'` will result in the class `'ui-icon-triangle-w'`.

If a button does not have an entry, it falls back to using buttonText.

If you are using a jQuery UI theme and would prefer not to display any icons and would rather use `buttonText` instead, you can set the `buttonIcons` option to `false`.

firstDay 1.3

The day that each week begins.

```
Integer, default: 0 (Sunday)
```

The value must be a number that represents the day of the week.

Sunday=`0`, Monday=`1`, Tuesday=`2`, etc.

This option is useful for UK users who need the week to start on Monday (`1`).

In versions 1.1 through 1.2.1, this option was known as *weekStart*.

isRTL 1.3

Displays the calendar in right-to-left mode.

```
Boolean, default: false
```

This option is useful for right-to-left languages such as Arabic and Hebrew.

In versions 1.1 through 1.2.1, this option was known as *rightToLeft*.

weekends 1.4.1

Whether to include Saturday/Sunday columns in any of the calendar views.

```
Boolean, default: true
```

weekMode 1.3

Determines the number of weeks displayed in a month view. Also determines each week's height.

```
String, default: 'fixed'
```

Available options:

'fixed'

> The calendar will always be 6 weeks tall. The height will always be the same, as determined by height, contentHeight, or aspectRatio.

'liquid'

> The calendar will have either 4, 5, or 6 weeks, depending on the month. The height of the weeks will stretch to fill the available height, as determined byheight, contentHeight, or aspectRatio.

'variable'

> The calendar will have either 4, 5, or 6 weeks, depending on the month. Each week will have the same constant height, meaning the calendar's height will change month-to-month.

In versions 1.0 through 1.2.1, this option was known as *fixedWeeks*.

weekNumbers 1.6

Determines if week numbers should be displayed on the calendar.

```
Boolean, default: false
```

If set to `true`, week numbers will be displayed in a separate left column in the month/basic views as well as at the top-left corner of the agenda views. See Available Views.

By defaut, FullCalendar will display the ISO8601 week number. To display other types of week numbers, see weekNumberCalculation.

weekNumberCalculation 1.6

The method for calculating week numbers that are displayed with the weekNumberssetting.

```
String/Function, default:"iso"
```

The default (`"iso"`) causes ISO8601 week numbers.

You may also specify a function, which accepts a single native Date object and returns an integer week number. Since FullCalendar currently lacks a built-in internationlization system, this is only way to implement localized week numbers.

height 1.4.2

Will make the entire calendar (including header) a pixel height.

```
Integer
```

By default, this option is unset and the calendar's height is calculated by aspectRatio.

Example usage of `height`:

```
$('#calendar').fullCalendar({

    height: 650

});
```

Setter

You can dynamically set a calendar's height after initialization:

```
$('#calendar').fullCalendar('option', 'height', 700);
```

contentHeight 1.4.2

Will make the calendar's content area a pixel height.

```
Integer
```

By default, this option is unset and the calendar's height is calculated by aspectRatio.

Example usage of `contentHeight`:

```
$('#calendar').fullCalendar({

    contentHeight: 600
```

```
});
```

Setter

You can dynamically set a calendar's contentHeight after initialization:

```
$('#calendar').fullCalendar('option', 'contentHeight', 650);
```

aspectRatio 1.3

Determines the width-to-height aspect ratio of the calendar.

```
Float, default: 1.35
```

A calendar is a block-level element that fills its entire avaiable width. The calendar's height, however, is determined by this ratio of width-to-height. (Hint: larger numbers make smaller heights).

The following example will initialize a calendar who's width is twice its height:

```
$('#calendar').fullCalendar({

    aspectRatio: 2

});
```

Setter

You can dynamically set a calendar's aspectRatio after initialization:

```
$('#calendar').fullCalendar('option', 'aspectRatio', 1.8);
```

The *setter* only works in version 1.4.2 and later.

viewDisplay *(callback)* 1.3

Triggered when the calendar loads and every time a different date-range is displayed.

```
function( view ) { }
```

The calendar's date-range changes whenever the user switches to a new view (for example, if they switch from "month" to "agendaWeek") or when they click the prev/next buttons.

`view` is the current View Object.

Within the callback function, `this` will be set to the calendar's main element.

Example usage of viewDisplay:

```
$('#calendar').fullCalendar({

    viewDisplay: function(view) {

        alert('The new title of the view is ' + view.title);

    }

});
```

In versions 1.0 through 1.2.1, this option was known as *monthDisplay*.

windowResize *(callback)* 1.3

Triggered after the calendar's dimensions have been changed due to the browser window being resized.

```
function( view ) { }
```

The calendar has automatically adapted to the new size when windowResize is triggered.

`view` is the current View Object.

Within the callback function, `this` will be set to the calendar's main element.

Example usage of windowResize:

```
$('#calendar').fullCalendar({

    windowResize: function(view) {

        alert('The calendar has adjusted to a window resize');

    }
```

```
});
```

In version 1.1 through 1.2.1, this option was known as *resize*.

dayRender *(callback)* 1.6

A hook for modifying a day cell.

```
function( date, cell ) { }
```

This callback lets you modify day cells that are part of the *month*, *basicWeek*, and*basicDay* views. See Available Views.

`date` is the native Date object for the given day.

You cannot return a new element. You must only modify the `cell` (a `<td>` element) that is provided.

This callback is called each time a cell needs to be freshly rendered. However, a cell does not need to be rerendered when switching views. Rendering only occurs when the calendar navigates to a different date/time.

Render (method) 1.3.1

Immediately forces the calendar to render and/or readjusts its size.

```
.fullCalendar( 'render' )
```

This method is useful in the scenario where a tab setup might hide/show a calendar. Call this method whenever the tabs are shown. Here is an example with the jQuery UI tab plugin:

```
$('#my-tabs').tabs({

    show: function(event, ui) {

        $('#calendar').fullCalendar('render');

    }

});
```

Notice that this example calls `render` whenever *any* tab is shown, not just the tab that the calendar is within. This is okay, because FullCalendar is smart enough to only render calendars that are visible to the user.

Prior to version 1.4.2, the *render* method did not readjust the size of the calendar. It only rendered the calendar if it was not already rendered.

Destroy (method) 1.4.3

Restores the element to the state before FullCalendar was initialized.

```
.fullCalendar( 'destroy' )
```

Removes elements, events handlers, and internal data.

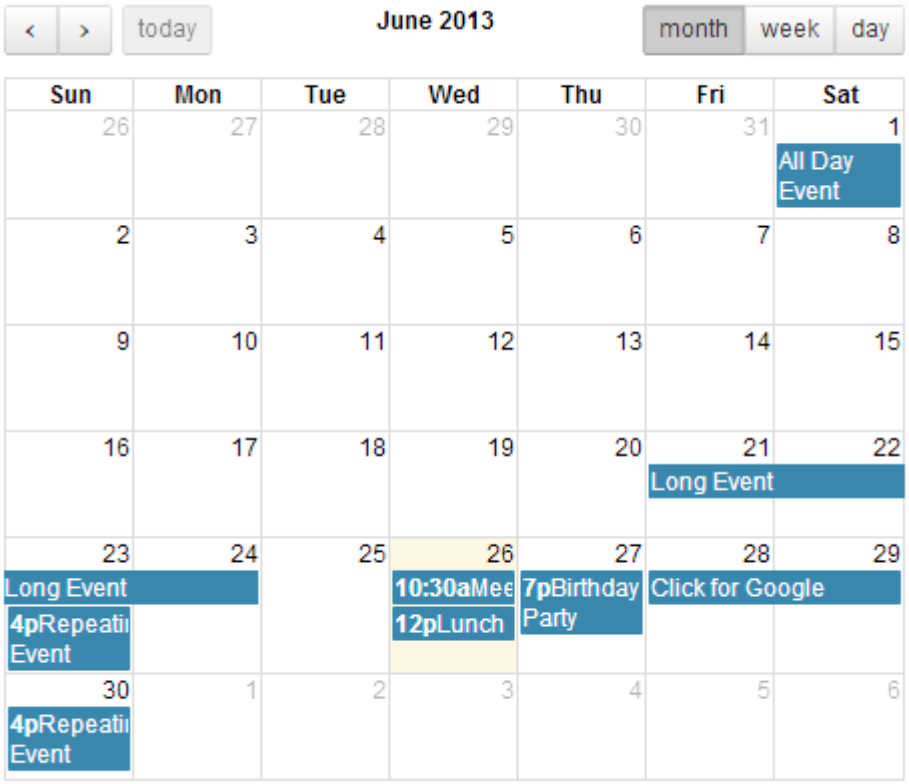
See also: buttonText

3. Views

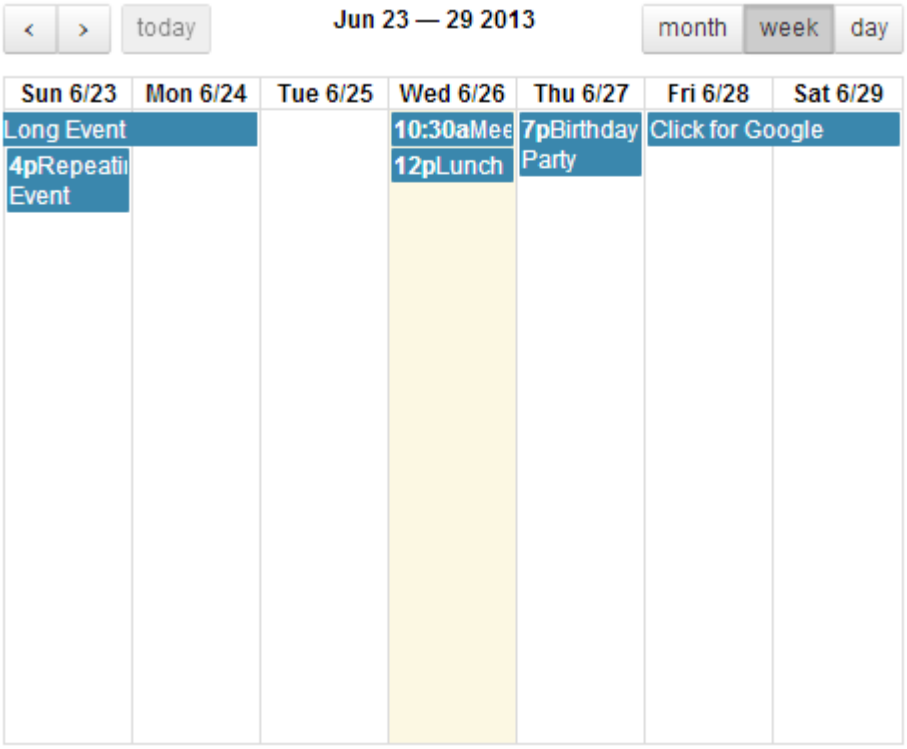
Available Views

FullCalendar has a number of different "views", or ways of displaying days and events. The following 5 views are all built in to FullCalendar:

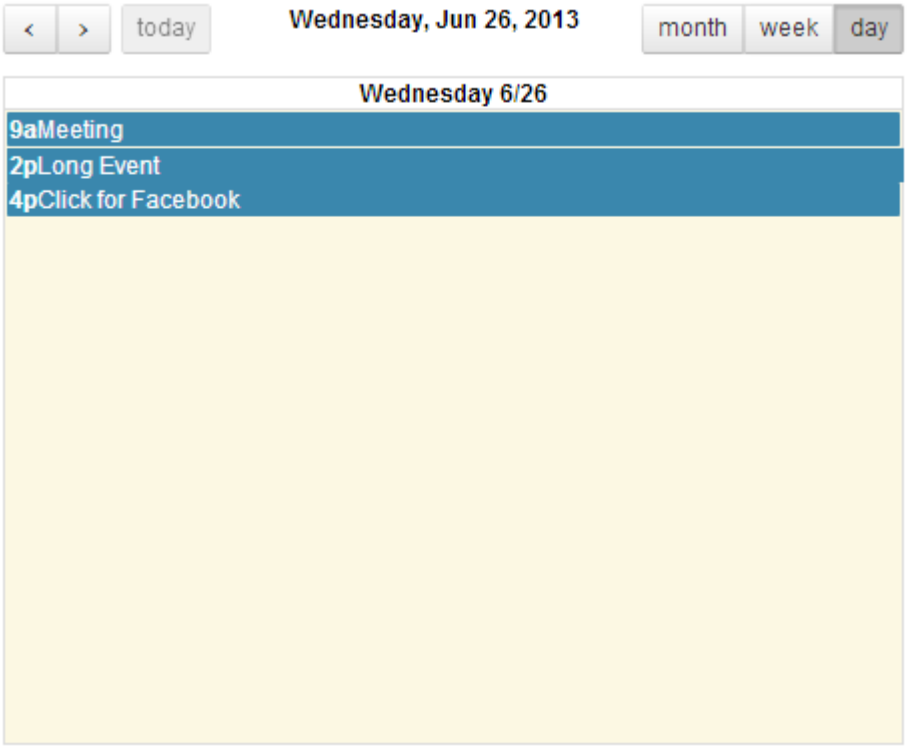
- **month –**


today
June 2013
month
week
day
Sun
Mon
Tue
Wed
Thu
Fri
Sat
All Day Event
Long Event
Long Event
10:30aMee
12pLunch
7pBirthday Party
Click for Google
4pRepeati Event
4pRepeati Event

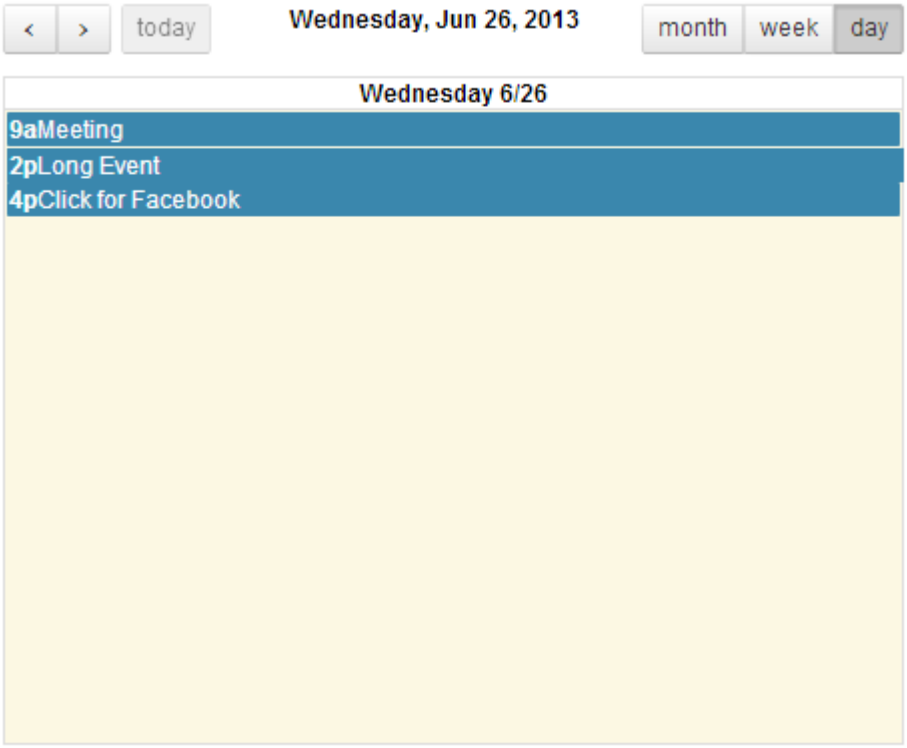
- **basicWeek** - (available since version 1.3)



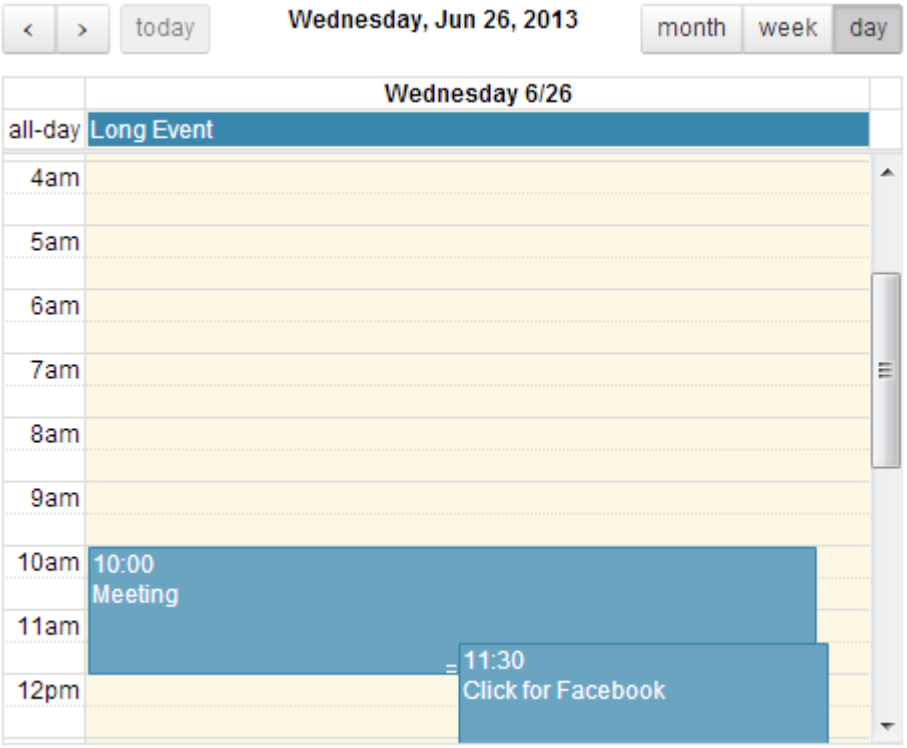
- **basicDay** - (available since version 1.3)



- **agendaWeek** - (available since version 1.4)



- **agendaDay** - (available since version 1.4)



You can define header buttons to allow the user to switch between them. You can set the initial view of the calendar with the defaultView option.

View Object **1.3**

An object that is passed to every callback, containing info about the current view.

It will be populated with the following properties:

name

Name of one of the Available Views (a string).

title

Title text that is displayed at the top of the header (such as "September 2009" or "Sep 7 - 13 2009").

start

A Date object of the first day of the month/week. If in day-view, the date of the single day.

end

A Date object of the day after the last day of the month/week. If in day-view, the Date after the single day.

Because this is an exclusive value, if the calendar has a month-view on October 2009, `end` will be November 1st.

visStart

A Date object of the first visible day of the view. In month-view, this value is often before the 1st day of the month, because most months do not begin on a Monday.

In week and day views, this value will always be the same as `start`.

visEnd

A Date object of the day after the last visible day (because it is exclusive like `end`).

View Option Hash 1.4

A way to specify options on a per-view basis.

This is specified as an object with key/value pairings. The key tells FullCalendar that the specified option value should only be applied to certain views.

Currently, View Option Hashes only work for dragOpacity, titleFormat,columnFormat, and timeFormat.

Here is how you target certain views:

```
{

    month:      // month view

    week:       // basicWeek & agendaWeek views
```

```
    day:        // basicDay & agendaDay views



    agenda:     // agendaDay & agendaWeek views

    agendaDay:  // agendaDay view

    agendaWeek: // agendaWeek view



    basic:      // basicWeek & basicDay views

    basicWeek:  // basicWeek view

    basicDay:   // basicDay view



    '':         // (an empty string) when no other properties match

}
```

So for example, if you wanted to `dragOpacity` to be `.2` for month view, but `.5` for all other views, here is what your FullCalendar initialization code would look like:

```
$('#calendar').fullCalendar({

    dragOpacity: {

        month: .2,

        ''   : .5

    }

});
```

defaultView 1.3

The initial view when the calendar loads.

```
String, default: 'month'
```

Any of the Available Views.

getView 1.4.2

Returns the View Object for the current view.

```
.fullCalendar( 'getView' ) -> View Object
```

This is useful if you want to get information about the calendar's title or start/end dates.

Example Usage:

```
var view = $('#calendar').fullCalendar('getView');

alert("The view's title is " + view.title);
```

changeView 1.3.1

Immediately switches to a different view.

```
.fullCalendar( 'changeView', viewName )
```

`viewName` must be a string value containing one of the Available Views.

See also: viewDisplay

4. Agenda Options

The following options only apply to the `agendaWeek` and `agendaDay` views:

allDaySlot 1.4

Determines if the "all-day" slot is displayed at the top of the calendar.

```
Boolean, default: true
```

When hidden with `false`, all-day events will not be displayed in agenda views.

allDayText 1.4

The text titling the "all-day" slot at the top of the calendar.

```
String, default: 'all-day'
```

The text titling the "all-day" slot at the top of the calendar.

axisFormat 1.4

Determines the time-text that will be displayed on the vertical axis of the agenda views.

```
String, default: 'h(:mm)tt'
```

The value is a format-string that will be processed by formatDate.

The default value will produce times that look like "5pm" and "5:30pm".

slotMinutes 1.4

The frequency for displaying time slots, in minutes.

```
Integer, default: 30
```

The default will make a slot every half hour.

snapMinutes 1.6

The time interval at which a dragged event will snap to the agenda view time grid. Also affects the granularity at which selections can be made. Specified in number of minutes.

```
Integer
```

The default value will be whatever slotMinutes is, which defaults to `30` (half an hour).

defaultEventMinutes 1.4

Determines the length (in minutes) an event appears to be when it has an unspecified end date.

```
Integer, default: 120
```

By default, if an Event Object as no `end`, it will appear to be 2 hours.

This option only affects events that appear in the agenda slots, meaning they have`allDay` set to `true`.

firstHour 1.4

Determines the first hour that will be visible in the scroll pane.

```
Integer, default: 6
```

Values must be from 0-23, where 0=midnight, 1=1am, etc.

The user will be able to scroll upwards to see events before this time. If you want to prevent users from doing this, use the minTime option instead.

minTime 1.4.2

Determines the first hour/time that will be displayed, even when the scrollbars have been scrolled all the way up.

```
Integer/String, default: 0
```

This can be a number like `5` (which means 5am), a string like `'5:30'` (which means 5:30am) or a string like `'5:30am'`.

maxTime 1.4.2

Determines the last hour/time (exclusively) that will be displayed, even when the scrollbars have been scrolled all the way down.

```
Integer/String, default: 24
```

This can be a number like `22` (which means 10pm), a string like `'22:30'` (which means 10:30pm) or a string like `'10:30pm'`.

5. Current Date

year

The initial year when the calendar loads.

```
Integer
```

Must be a 4-digit year like `2009`.

If `year` is unspecified, the calendar will begin at the current year.

month

The initial month when the calendar loads.

```
Integer
```

IMPORTANT: The value is 0-based, meaning January=`0`, February=`1`, etc.

If `month` is unspecified and `year` is set to the current year, the calendar will start on the current month.
If `month` is unspecified and `year` is *not* set to the current year, the calendar will start on January.

date 1.3

The initial date-of-month when the calendar loads.

```
Integer
```

This option only matters for the week & day views. Month view does not need this option, because month view always displays the entire month from start to finish.

If `date` is unspecified, and `year`/`month` are set to the current year/month, then the calendar will start on the current date. If `date` is unspecified, and `year`/`month` are *not*set to the current year/month, then the calendar will start on the first of the month.

prev (menthod)1.3

Moves the calendar one step back (either by a month, week, or day).

```
.fullCalendar( 'prev' )
```

If the calendar is in `month` view, will move the calendar back one month.

If the calendar is in `basicWeek` or `agendaWeek`, will move the calendar back one week.

If the calendar is in `basicDay` or `agendaDay`, will move the calendar back one day.

Example using `prev` with an external button:

```
$('#my-prev-button').click(function() {

    $('#calendar').fullCalendar('prev');

});
```

In versions 1.0 through 1.2.1, this option was known as *prevMonth*.

next (menthod) 1.3

Moves the calendar one step forward (either by a month, week, or day).

```
.fullCalendar( 'next' )
```

If the calendar is in `month` view, will move the calendar forward one month.

If the calendar is in `basicWeek` or `agendaWeek`, will move the calendar forward one week.

If the calendar is in `basicDay` or `agendaDay`, will move the calendar forward one day.

Example using `next` with an external button:

```
$('#my-next-button').click(function() {

    $('#calendar').fullCalendar('next');

});
```

In versions 1.0 through 1.2.1, this option was known as *prevMonth*.

prevYear (menthod) 1.2.x and 1.4 onward

Moves the calendar back one year.

```
.fullCalendar( 'prevYear' )
```

Version 1.3.x did not include this method, but you can use `.fullCalendar('incrementDate', -1)` instead.

nextYear (menthod) 1.2.x and 1.4 onward

Moves the calendar forward one year.

```
.fullCalendar( 'nextYear' )
```

Version 1.3.x did not include this method, but you can use `.fullCalendar('incrementDate', 1)` instead.

today (menthod)

Moves the calendar to the current date.

```
.fullCalendar( 'today' )
```

Example using `today` with an external button:

```
$('#my-today-button').click(function() {

    $('#calendar').fullCalendar('today');

});
```

gotoDate (menthod)

Moves the calendar to an arbitrary year/month/date.

```
.fullCalendar( 'gotoDate', year [, month, [ date ]] )
```

IMPORTANT: `month` is 0-based, meaning January=`0`, February=`1`, etc.

This method can also be called with a single argument, a Date object (only works in versions >= 1.3.2).

incrementDate (menthod) 1.3

Moves the calendar forward/backward an arbitrary amount of time.

```
.fullCalendar( 'incrementDate', years [, months, [ days ]] )
```

getDate (menthod) 1.4.2

Returns a Date object for the current date of the calendar.

```
.fullCalendar( 'getDate' ) -> Date
```

For month view, it will always be sometime between the first and last day of the month. For week views, it will always be sometime between the first and last day of the week.

Example of displaying a calendar's current date when a button is clicked:

```
$('#my-button').click(function() {

    var d = $('#calendar').fullCalendar('getDate');

    alert("The current date of the calendar is " + d);

});
```

6. Text/Time Customization

timeFormat 1.1

Determines the time-text that will be displayed on each event.

```
String/View Option Hash, default:
```

```
{
    // for agendaWeek and agendaDay
    agenda: 'h:mm{ - h:mm}', // 5:00 - 6:30

    // for all other views
    '': 'h(:mm)t'            // 7p
}
```

A single format string will change the time-text for events in all views. A View Option Hash may be provided to target specific views (this is what the default does).

Uses formatDate/formatDates formatting rules. (The formatting rules were much different in versions before 1.3. See here)

Time-text will only be displayed for Event Objects that have `allDay` equal to `false`.

Here is an example of displaying all events in a 24-hour format:

```
$('#calendar').fullCalendar({
    events: [
        {
            title:  'My Event',
            start:   '2010-01-01T14:30:00',
            allDay: false
        }
        // other events here...
    ],
    timeFormat: 'H(:mm)' // uppercase H for 24-hour clock
```

```
});
```

columnFormat 1.3

Determines the text that will be displayed on the calendar's column headings.

```
String/View Option Hash, default:
```

```
{
    month: 'ddd',    // Mon
    week: 'ddd M/d', // Mon 9/7
    day: 'dddd M/d'  // Monday 9/7
}
```

A single string will set the title format for all views. A View Option Hash may be provided to target specific views (this is what the default does).

Uses formatDate/formatDates formatting rules. (The formatting rules were much different in versions before 1.3. See here)

A *View Option Hash* can only be provided in version 1.4 and later.

titleFormat 1.1

Determines the text that will be displayed in the header's title.

```
String/View Option Hash, default:
```

```
{
    month: 'MMMM yyyy',                              // September 2009
    week: "MMM d[ yyyy]{ '&#8212;'[ MMM] d yyyy}", // Sep 7 - 13 2009
    day: 'dddd, MMM d, yyyy'                    // Tuesday, Sep 8, 2009
}
```

A single string will set the title format for all views. A View Option Hash may be provided to target specific views (this is what the default does).

Uses formatDate/formatDates formatting rules. (The formatting rules were much different in versions before 1.3. See here)

A *View Option Hash* can only be provided in version 1.4 and later.

buttonText 1.3

Text that will be displayed on buttons of the header.

```
Object, default:
```

```
{
    prev:     '&lsaquo;', // <
    next:     '&rsaquo;', // >
    prevYear: '&laquo;',  // <<
    nextYear: '&raquo;',  // >>
    today:    'today',
    month:    'month',
    week:     'week',
    day:      'day'
}
```

If you wanted to change the prev/next buttons to use < and > characters, here is what you would do:

```
$('#calendar').fullCalendar({
    buttonText: {
        prev: '&lt;',
        next: '&gt;'
```

```
    }

});
```

monthNames 1.3

Full names of months.

```
Array, default:
```

```
['January', 'February', 'March', 'April', 'May', 'June', 'July',
 'August', 'September', 'October', 'November', 'December']
```

Prior to version 1.3, this was possible by setting *$.fullCalendar.monthNames*

monthNamesShort 1.3

Abbreviated names of months.

```
Array, default:
```

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun'
 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
```

Prior to version 1.3, this was possible by setting *$.fullCalendar.monthAbbrevs*

dayNames 1.3

Full names of days-of-week.

```
Array, default:
```

```
['Sunday', 'Monday', 'Tuesday', 'Wednesday',
 'Thursday', 'Friday', 'Saturday']
```

Prior to version 1.3, this was possible by setting *$.fullCalendar.dayNames*

dayNamesShort 1.3

Abbreviated names of days-of-week.

```
Array, default:
```

```
['Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat']
```

Prior to version 1.3, this was possible by setting *$.fullCalendar.dayAbbrevs*

weekNumberTitle 1.6

The heading text for week numbers.

```
String, default: "W"
```

This text will go above the week number column in the month/basic views. It will go alongside the week number text in the top-left cell for agenda views.

Related agenda options: allDayText, axisFormat

Essential for date/time formatting: formatDate commands.

7. Clicking & Hovering

dayClick (callback)

Triggered when the user clicks on a day.

```
function( date, allDay, jsEvent, view ) { }
```

`date` holds a Date object for the current day. If user has clicked on a slot in the agendaWeek or agendaDay views, `date` will also have its time set.

`allDay` will be set to true `false` if the user has clicked on a slot in the agendaWeek or agendaDay views. Otherwise, it will be `true`.

`jsEvent` holds the native JavaScript event with low-level information such as click coordinates.

`view` is set to the current View Object.

Within the callback function, `this` is set to the `<td>` of the clicked day.

Here is an example that demonstrates all of these variables:

```
$('#calendar').fullCalendar({

    dayClick: function(date, allDay, jsEvent, view) {



        if (allDay) {

            alert('Clicked on the entire day: ' + date);

        }else{

            alert('Clicked on the slot: ' + date);

        }



        alert('Coordinates: ' + jsEvent.pageX + ',' + jsEvent.pageY);



        alert('Current view: ' + view.name);



        // change the day's background color just for fun

        $(this).css('background-color', 'red');
```

```
    }

});
```

The *allDay* parameter has only been available since version 1.4.

eventClick (callback)

Triggered when the user clicks an event.

```
function( event, jsEvent, view ) { }
```

`event` is an Event Object that holds the event's information (date, title, etc).

`jsEvent` holds the native JavaScript event with low-level information such as click coordinates.

`view` holds the current View Object.

Within the callback function, `this` is set to the event's `<div>` element.

Here is an example demonstrating all these variables:

```
$('#calendar').fullCalendar({

    eventClick: function(calEvent, jsEvent, view) {



        alert('Event: ' + calEvent.title);

        alert('Coordinates: ' + jsEvent.pageX + ',' + jsEvent.pageY);

        alert('View: ' + view.name);



        // change the border color just for fun

        $(this).css('border-color', 'red');
```

```
    }

});
```

Return Value

Normally, if the Event Object has its `url` property set, a click on the event will cause the browser to visit the event's url (in the same window/tab). Returning `false` from within your function will prevent this from happening.

Often, developers want an event's `url` to open in a different tab or a popup window. The following example shows how to do this:

```
$('#calendar').fullCalendar({

    events: [

        {

            title: 'My Event',

            start: '2010-01-01',

            url: 'http://google.com/'

        }

        // other events here

    ],

    eventClick: function(event) {

        if (event.url) {

            window.open(event.url);

            return false;
```

```
        }

    }

});
```

The `window.open` function can take many other options.

eventMouseover (callback)

Triggered when the user mouses over an event.

```
function( event, jsEvent, view ) { }
```

`event` is an Event Object that holds the event's information (date, title, etc).

`jsEvent` holds the native JavaScript event with low-level information such as mouse coordinates.

`view` holds the current View Object.

Within the callback function, `this` is set to the event's `<div>` element.

eventMouseout (callback)

Triggered when the user mouses out of an event.

```
function( event, jsEvent, view ) { }
```

`event` is an Event Object that holds the event's information (date, title, etc).

`jsEvent` holds the native JavaScript event with low-level information such as mouse coordinates.

`view` holds the current View Object.

Within the callback function, `this` is set to the event's `<div>` element.

8. Selection

selectable 1.4.6

Allows a user to highlight multiple days or timeslots by clicking and dragging.

```
Boolean/View Option Hash, default: false
```

To let the user make selections by clicking and dragging, this option must be set to `true`.

The select and unselect callbacks will be useful for monitoring when selections are made and cleared.

To learn the ways in which selections can be cleared, read the docs for the unselect callback.

To view an example of how to create a new event based on a user's selection see "demos/selectable.html" in the download, or visit this page.

A View Option Hash has been supported since version 1.4.7

selectHelper 1.4.6

Whether to draw a "placeholder" event while the user is dragging.

```
Boolean/Function, default: false
```

This option only applies to the agenda views.

A value of `true` will draw a "placeholder" event while the user is dragging (similar to what Google Calendar does for its week and day views). A value of `false` (the default) will draw the standard highlighting over each cell.

A function can also be specified for drawing custom elements. It will be given 2 arguments: the selection's start date and end date (Date objects). It must return a DOM element that will be used.

unselectAuto 1.4.6

Whether clicking elsewhere on the page will cause the current selection to be cleared.

```
Boolean, default: true
```

This option can only take effect when selectable is set to `true`.

unselectCancel 1.4.6

A way to specify elements that will ignore the unselectAuto option.

```
String, default: ''
```

Clicking on elements that match this jQuery selector will prevent the current selection from being cleared (due to the unselectAuto option).

This option is useful if you have a "Create an event" form that shows up in response to the user making a selection. When the user clicks on this form, you probably don't want to the current selection to go away. Thus, you should add a class to your form such as "my-form", and set the `unselectCancel` option to `".my-form"`.

Select (callback) 1.4.6

A callback that will fire after a selection is made.

```
function( startDate, endDate, allDay, jsEvent, view )
```

`startDate` is a Date object indicating the beginning of the selection.

`endDate` is a Date object indicating the end of the selection. It follows the same rules as the Event Object for inclusivity/exclusivity: when `allDay` is `true`, `endDate` *includes* the last day.

`allDay` is a boolean indicating if entire days were selected (days in month view or the "all-day" slot in the agenda view) or time slots were selected.

`jsEvent` holds the primitive JavaScript event with information such as mouse coordinates. If `select` has been triggered via the select method, `jsEvent` will be `undefined`. (added in version 1.4.7)

unselect (callback) 1.4.6

A callback that will fire when the current selection is cleared.

```
function( view, jsEvent )
```

A selection might be cleared for a number of reasons:

1. The user clicks away from the current selection (doesn't happen when unselectAuto is `false`).
2. The user makes a *new* selection. The `unselect` callback will be fired before the new selection occurs.
3. The user moves forward or backward in the current view, or switches to a new view.
4. The unselect method is called through the API.

`jsEvent` holds the primitive JavaScript event with information such as mouse coordinates. If `unselect` has been triggered via the unselect method, `jsEvent` will be `undefined`. (added in version 1.4.7)

select (callback) 1.4.6

A method for programmatically selecting a period of time.

```
.fullCalendar( 'select', startDate, endDate, allDay )
```

unselect (callback) 1.4.6

A method for programmatically clearing the current selection.

```
.fullCalendar( 'unselect' )
```

9. Event Data

Event Object

A standard JavaScript object that FullCalendar uses to store information about a calendar event.

Here are its properties:

id	String/Integer. Optional Uniquely identifies the given event. Different instances of repeating events should all have the same `id`.
title	String. *Required*. The text on an event's element
allDay	`true` or `false`. Optional. Whether an event occurs at a specific time-of-day. This property affects whether an event's time is shown. Also, in the agenda views, determines if it is displayed in the "all-day" section. **Don't include quotes** around your `true`/`false`. This value is not a string! When specifying Event Objects for `events` or `eventSources`, omitting this property will make it inherit fromallDayDefault, which is normally `true`.
start	Date. *Required*. The date/time an event begins. When specifying Event Objects for `events` or `eventSources`, you may specify a

	string in IETF format (ex: "Wed, 18 Oct 2009 13:00:00 EST"), a string in ISO8601 format (ex: "2009-11-05T13:15:30Z") or a UNIX timestamp.
end	Date. Optional. The date/time an event ends. As with `start`, you may specify it in IETF, ISO8601, or UNIX timestamp format. **If an event is all-day**... the end date is inclusive. This means an event with `start` Nov 10 and`end` Nov 12 will span 3 days on the calendar. **If an event is NOT all-day**... the end date is exclusive. This is only a gotcha when your `end` has time 00:00. It means your event ends on midnight, and it will not span through the next day.
url	String. Optional. A URL that will be visited when this event is clicked by the user. For more information on controlling this behavior, see the eventClick callback.
className	String/Array. Optional. A CSS class (or array of classes) that will be attached to this event's element.
editable	`true` or `false`. Optional. Overrides the master editable option for this single event.
source	Event Source Object. Automatically populated.

	A reference to the event source that this event came from.
New options have been added in version 1.5 to change an event's colors:	
color	Sets an event's background *and* border color just like the calendar-wide eventColor option.
backgroundColor	Sets an event's background color just like the calendar-wide eventBackgroundColor option.
borderColor	Sets an event's border color just like the the calendar-wide eventBorderColor option.
textColor	Sets an event's text color just like the calendar-wide eventTextColor option.

Non-standard Fields

In addition to the fields above, you may also include your own non-standard fields in each Event Object. FullCalendar will not modify or delete these fields. For example, developers often include a `description` field for use in callbacks such as eventRender.

Prior to version 1.3, the *end* date was always exclusive, even when *allDay* was set to *true*.

Prior to version 1.3, the *allDay* property did not exist. *showTime* was used instead. It was set to *true* to show times and *false* to hide them.

The *className* and *source* properties did not exist prior to version 1.2.

Event Source Object

An "event source" is anything that provides FullCalendar with data about events. It can be a simple array, an event-generating function that you define, a URL to a json feed, or a Google Calendar feed.

Since version 1.5, Event Objects can have "options" associated with them. However, before you can start specifying options, you must write an Event Object in its*extended form*. It must be a traditional JavaScript object with properties. Here are the extended forms for each type of Event Source:

Array of events:

```
{
    events: [
        {
            title: 'Event1',
            start: '2011-04-04'
        },
        {
            title: 'Event2',
            start: '2011-05-05'
        }
        // etc...
    ],
    color: 'yellow',   // an option!
    textColor: 'black' // an option!
}
```

Event-generating function:

```
{
    events: function(start, end, callback) {
```

```
        // ...

    },

    color: 'yellow',   // an option!

    textColor: 'black' // an option!

}
```

JSON feed:

```
{

    url: '/myfeed.php',

    color: 'yellow',   // an option!

    textColor: 'black' // an option!

}
```

Google Calendar feed:

```
{

    url: 'http://www.google.com/your_feed_url/',

    color: 'yellow',   // an option!

    textColor: 'black' // an option!

}
```

Event Source Options

color	Sets every Event Object's `color` for this source.
backgroundColor	Sets every Event Object's `backgroundColor` for this source.
borderColor	Sets every Event Object's `borderColor` for this source.

textColor	Sets every Event Object's `textColor` for this source.
className	Sets every Event Object's `className` for this source.
editable	Sets every Event Object's `editable` for this source.
allDayDefault	Sets the allDayDefault option, but only for this source.
ignoreTimezone	Sets the ignoreTimezone option, but only for this source.
eventTransform	Sets the eventTransform callback, but only for this source.

For JSON feeds, there are additional options you can set.

events (as an array)

An array of Event Objects that will be displayed on the calendar.

Here is an example of how to specify an array of events:

```
$('#calendar').fullCalendar({

    events: [

        {

            title  : 'event1',

            start  : '2010-01-01'

        },

        {

            title  : 'event2',

            start  : '2010-01-05',

            end    : '2010-01-07'

        },

        {
```

```
            title  : 'event3',
            start  : '2010-01-09 12:30:00',
            allDay : false // will make the time show
        }
    ]
});
```

Make sure you do not have a comma after the last event in your array! It will make Internet Explorer choke.

Extended Form

Since version 1.5, you are able to specify Event Source options. This often comes in handy when you are using the eventSources option to specify multiple event sources and you want certain options to only apply to certain sources. However, to do this, you must write things a little differently:

```
$('#calendar').fullCalendar({

    eventSources: [

        // your event source
        {
            events: [ // put the array in the `events` property
                {
                    title  : 'event1',
                    start  : '2010-01-01'
```

```
            },

            {

                title  : 'event2',

                start  : '2010-01-05',

                end    : '2010-01-07'

            },

            {

                title  : 'event3',

                start  : '2010-01-09 12:30:00',

            }

        ],

        color: 'black',     // an option!

        textColor: 'yellow' // an option!

    }



    // any other event sources...



]



});
```

A list of available Event Source options can be found here.

events (as a json feed)

A URL of a JSON feed that the calendar will fetch Event Objects from.

FullCalendar will visit the URL whenever it needs new event data. This happens when the user clicks prev/next or changes views. FullCalendar will determine the date-range it needs events for and will pass that information along in GET parameters.

The GET parameter names will be determined by the startParam and endParam options. (`"start"` and `"end"` by default).

The value of the parameters will always be UNIX timestamps (seconds since 1970).

Consider the following script:

```
$('#calendar').fullCalendar({

    events: '/myfeed.php'

});
```

Here is a URL that FullCalendar might visit:

`/myfeed.php?start=1262332800&end=1265011200&_=1263178646`

The `_` parameter is automatically inserted to prevent the browser from caching the result (more below).

If you need to access a feed that is in a different domain, you can use JSONP with a `?` in your URL (see the JSONP discussion in $.ajax).

Extended Form

Since version 1.5, you are able to specify Event Source options. This often comes in handy when you are using the eventSources option to specify multiple event sources and you want certain options to only apply to certain sources. However, to do this, you must write things a little differently:

```
$('#calendar').fullCalendar({

    eventSources: [

        // your event source
        {
            url: '/myfeed.php', // use the `url` property
            color: 'yellow',    // an option!
            textColor: 'black'  // an option!
        }

        // any other sources...

    ]

});
```

A list of general Event Source options can be found here.However, there are additional options that apply specifically to JSON feeds:

startParam	Sets the startParam option, but only for this source.
endParam	Sets the endParam option, but only for this source.

jQuery $.ajax options

You can also specify any of the jQuery $.ajax options within the same object! This allows you to easily pass additional parameters to your feed script, as well as listen to ajax callbacks:

```
$('#calendar').fullCalendar({


    eventSources: [


        // your event source
        {
            url: '/myfeed.php',
            type: 'POST',
            data: {
                custom_param1: 'something',
                custom_param2: 'somethingelse'
            },
            error: function() {
                alert('there was an error while fetching events!');
            },
            color: 'yellow',   // a non-ajax option
            textColor: 'black' // a non-ajax option
        }


        // any other sources...
```

```
    ]


});
```

Here is the same example, but using the single-source `events` option instead:

```
$('#calendar').fullCalendar({



    events: {

        url: '/myfeed.php',

        type: 'POST',

        data: {

            custom_param1: 'something',

            custom_param2: 'somethingelse'

        },

        error: function() {

            alert('there was an error while fetching events!');

        },

        color: 'yellow',   // a non-ajax option

        textColor: 'black' // a non-ajax option

    }



});
```

Caching

By default, FullCalendar will insert a `_` parameter into the URL of the request to prevent the browser from caching the response. FullCalendar achieves this internally by setting the $.ajax parameter to `false`.

If you would like to counteract this and prevent the `_` parameter, you can set the `cache` option to `true`:

```
$('#calendar').fullCalendar({

    events: {
        url: '/myfeed.php',
        cache: true
    }

});
```

events (as a function)

A custom function for programmatically generating Event Objects.

```
function( start, end, callback ) { }
```

FullCalendar will call this function whenever it needs new event data. This is triggered when the user clicks prev/next or switches views.

This function will be given `start` and `end` parameters, which are Date objects denoting the range the calendar needs events for.

It will also be given `callback`, a function that must be called when the custom event function has generated its events. It is the event function's responsibility to make sure `callback` is being called with an array ofEvent Objects.

Here is an example showing how to use an event function to fetch events from a hypothetical XML feed:

```
$('#calendar').fullCalendar({

    events: function(start, end, callback) {

        $.ajax({

            url: 'myxmlfeed.php',

            dataType: 'xml',

            data: {

                // our hypothetical feed requires UNIX timestamps

                start: Math.round(start.getTime() / 1000),

                end: Math.round(end.getTime() / 1000)

            },

            success: function(doc) {

                var events = [];

                $(doc).find('event').each(function() {

                    events.push({

                        title: $(this).attr('title'),

                        start: $(this).attr('start') // will be parsed

                    });

                });
```

```
                callback(events);

            }

        });

    }

});
```

However, if you have the choice, JSON is a better idea because you can just specify afeed URL.

Extended Form

Since version 1.5, you are able to specify Event Source options. This often comes in handy when you are using the eventSources option to specify multiple event sources and you want certain options to only apply to certain sources. However, to do this, you must write things a little differently:

```
$('#calendar').fullCalendar({

    eventSources: [

        // your event source
        {
            events: function(start, end, callback) {
                // ...
            },
            color: 'yellow',   // an option!
            textColor: 'black' // an option!
        }
```

```
        // any other sources...

    ]

});
```

eventSources 1.2

A way to specify multiple event sources.

```
Array
```

This option is used instead of the `events` option.

You can put any number of event arrays, functions,JSON feed URLs, or full-out Event Source Objectsinto the `eventSources` array.

Here is an example calendar that displays two JSON feeds:

```
$('#calendar').fullCalendar({

    eventSources: [

        '/feed1.php',

        '/feed2.php'

    ]

});
```

allDayDefault 1.3.1

Determines the default value for each Event Object's `allDay` property, when it is unspecified.

```
Boolean, default: true
```

ignoreTimezone 1.4.8

When parsing ISO8601 dates, whether UTC offsets should be ignored while processing event source data.

```
Boolean, default: true
```

The default is `true`, which means the UTC offset for all ISO8601 dates will be ignored. For example, the date `"2008-11-05T08:15:30-05:00"` will be processed as November 5th, 2008 at 8:15am *in the local offset of the browser*.

If you are using ISO8601 dates with UTC offsets, chances are you want them processed. You must set this option to `false`. In the future, the default for this option will probably be changed to `false`.

startParam

A GET parameter of this name will be inserted into each JSON feed's URL.

```
String, default: 'start'
```

The value of this GET parameter will be a UNIX timestamp denoting the start of the first visible day (inclusive).

endParam

A GET parameter of this name will be inserted into each JSON feed's URL.

```
String, default: 'end'
```

The value of this GET parameter will be a UNIX timestamp denoting the end of the last visible day (exclusive).

lazyFetching 1.4.5

Determines when event fetching should occur.

```
Boolean, default: true
```

When set to `true` (the default), the calendar will only fetch events when it absolutely needs to, minimizing AJAX calls. For example, say your calendar starts out in month view, in February. FullCalendar will fetch events for the entire month of February and store them in its internal cache. Then, say the user switches to week view and begins browsing the weeks in February. The calendar will avoid fetching events because it already has this information stored.

When set to `false`, the calendar will fetch events any time the view is switched, or any time the current date changes (for example, as a result of the user clicking prev/next).

Before this option existed, FullCalendar would always do "lazy" event fetching, as if `lazyFetching` were set to `true`.

eventDataTransform (callback) 1.6

Transforms custom data into a standard Event Object.

```
function( eventData ) {}
```

This hook allows you to receive arbitrary event data from a JSON feed or any otherEvent Source and transform it into the type of data FullCalendar accepts. This let's you easily accept alternative data formats without having to write a completely customevents function.

This function is called once per received event. `eventData` is the event data that has been receieved. The function must return a new object in the Event Object format

loading (callback)

Triggered when event fetching starts/stops.

```
function( isLoading, view )
```

Triggered with a `true` argument when the calendar begins fetching events via AJAX. Triggered with `false` when done.

`view` is the current View Object.

This function is often used to show/hide a loading indicator.

updateEvent (method) 1.2

Reports changes to an event and renders them on the calendar.

```
.fullCalendar( 'updateEvent', event )
```

`event` must be the original Event Object for an event,**not** merely a reconstructed object. The original Event Object can obtained by callbacks such as eventClick, or by the clientEvents method.

Here is how you might update an event after a click:

```
$('#calendar').fullCalendar({

    eventClick: function(event, element) {

        event.title = "CLICKED!";

        $('#calendar').fullCalendar('updateEvent', event);
```

```
    }

});
```

clientEvents (method) 1.3

Retrieves events that FullCalendar has in memory.

```
.fullCalendar( 'clientEvents' [, idOrFilter ] ) -> Array
```

This method will return an array of Event Objects that FullCalendar has stored in client-side memory.

If `idOrFilter` is omitted, *all* events will be returned.

If `idOrFilter` is an ID, all events with the same ID will be returned.

`idOrFilter` may also be a filter function that accepts one Event Objectargument and returns `true` if it should be included in the result set.

In versions 1.2 and 1.2.1, this option was known as *getEventsByID*

removeEvents (method) 1.2

Removes events from the calendar.

```
.fullCalendar( 'removeEvents' [, idOrFilter ] )
```

If `idOrFilter` is omitted, *all* events are removed.

If `idOrFilter` is an ID, all events with the same ID will be removed.

`idOrFilter` may also be a filter function that accepts one Event Objectargument and returns `true` if it should be removed.

refetchEvents (method) 1.3

Refetches events from all sources and rerenders them on the screen.

```
.fullCalendar( 'refetchEvents' )
```

Prior to version 1.3, the same effect was achieved by the *refresh* method.

addEventSource (method) 1.2

Dynamically adds an event source.

```
.fullCalendar( 'addEventSource', source )
```

Source may be an Array/URL/Function just as in the `events` option. Events will be immediately fetched from this source and placed on the calendar.

removeEventSource (method) 1.2

Dynamically removes an event source.

```
.fullCalendar( 'removeEventSource', source )
```

Events from the source will immediately be removed from the calendar.

Since version 1.5, the `source` parameter has become rather relaxed. You can provide an event source's Array/URL/Function**or** you can specify the full Event Source Object.

Prior to version 1.5, things were more strict. You must specify a reference to the original Array/URL/Function.

See also

: renderEvent (for adding an event)

10. Event Rendering

Colors

Coloring Events

You can change the color of all events on the calendar using the eventColor option like so:

```
$('#calendar').fullCalendar({

    events: [

        // my event data

    ],

    eventColor: '#378006'

});
```

The eventBackgroundColor, eventBorderColor, and eventTextColor options can also be used for coloring events.

If you want to change the color for events in a specific Event Source, please look at the Event Source options documentation.

If you want to change the color for individual events, please look at theEvent Object color options.

[deprecated] Coloring Events prior to version 1.5

You can modify the default color that affects all events by adding some css in the following form:

```
.fc-event,

.fc-agenda .fc-event-time,

.fc-event a {
```

```
    background-color: black; /* background color */

    border-color: black;     /* border color */

    color: red;              /* text color */

    }
```

You can also change the color for certain events by using the `className` property of each Event Object. Here is an example of the CSS you would write if your `className` was "holiday":

```
.holiday,

.fc-agenda .holiday .fc-event-time,

.holiday a {

    background-color: green; /* background color */

    border-color: green;     /* border color */

    color: yellow;           /* text color */

    }
```

If you are using the "default" and "className" techniques together, make sure the CSS for the default technique comes first.

Versions prior to 1.3 had different ways of setting colors through CSS. Check your version's *fullcalendar.css* for an example.

eventColor

Sets the background *and* border colors for all events on the calendar.

```
String
```

You can use any of the CSS color formats such `#f00`, `#ff0000`, `rgb(255,0,0)`, or `red`.

This option can be overridden on a per-source basis with the`color` Event Source Object option or on a per-event basis with the`color` Event Object option.

eventBackgroundColor

Sets the background color for all events on the calendar.

```
String
```

You can use any of the CSS color formats such `#f00`, `#ff0000`, `rgb(255, 0, 0)`, or `red`.

This option can be overridden on a per-source basis with the`backgroundColor` Event Source Object option or on a per-event basis with the`backgroundColor` Event Object option.

eventBorderColor

Sets the border color for all events on the calendar.

```
String
```

You can use any of the CSS color formats such `#f00`, `#ff0000`, `rgb(255, 0, 0)`, or `red`.

This option can be overridden on a per-source basis with the`borderColor` Event Source Object option or on a per-event basis with the`borderColor` Event Object option.

eventTextColor

Sets the text color for all events on the calendar.

```
String
```

You can use any of the CSS color formats such `#f00`, `#ff0000`, `rgb(255, 0, 0)`, or `red`.

This option can be overridden on a per-source basis with the`textColor` Event Source Object option or on a per-event basis with the`textColor` Event Object option.

eventRender (callback)

Triggered while an event is being rendered.

```
function( event, element, view ) { }
```

`event` is the Event Object that is attempting to be rendered.

`element` is a newly created jQuery `<div>` that will be used for rendering. It has already been populated with the correct time/title.

The `eventRender` callback function can modify `element`, return a brand new DOM element that will be used for rendering instead, or it can return `false`, which will prevent the event from being rendered at all.

`eventRender` is a great way to attach other jQuery plugins to event elements, such as a qTip tooltip effect:

```
$('#calendar').fullCalendar({
    events: [
        {
            title: 'My Event',
            start: '2010-01-01',
            description: 'This is a cool event'
        }
        // more events here
    ],
    eventRender: function(event, element) {
        element.qtip({
```

```
            content: event.description

        });

    }

});
```

Note that `description` is a non-standard Event Object field, which is allowed.

eventAfterRender (callback) 1.4.2

Triggered after an event has been placed on the calendar in its final position.

```
function( event, element, view ) { }
```

eventAfterAllRender (callback) 1.6

Triggered after all events have finished rendering.

```
function( view ) { }
```

renderEvent (method) 1.3

Renders a new event on the calendar.

```
.fullCalendar( 'renderEvent', event [, stick ] )
```

`event` must be an Event Object with a `title` and `start` at the very least.

Normally, the event will disappear once the calendar refetches its event sources (example: when prev/next is clicked). However, specifying `stick` as `true` will cause the event to be permanently fixed to the calendar.

rerenderEvents (method) 1.3

Rerenders all events on the calendar.

```
.fullCalendar( 'rerenderEvents' )
```

Prior to version 1.3, the same effect could be achieved by the *refresh* method.

11. Event Dragging & Resizing

Requirements for Event Dragging & Resizing

Event dragging and resizing requires the following jQuery UI modules:

- Draggable (the Droppable module is **not** required)
- Resizable

These modules can be conveniently bundled/compressed/downloaded at the jQuery UIdownloads page. They are also included in the FullCalendar download in "jquery/jquery-ui-custom.js", all bundled and compressed into one file.

After you have gathered the dependencies, **you still need to enable dragging/resizing**by settings the editable option to `true`.

editable 1.3

Determines whether the events on the calendar can be modified.

```
Boolean, default: false
```

This determines if the events can be *dragged* and *resized*. Enables/disables both at the same time. If you don't want both, use `editable` in conjunction with disableDragging and disableResizing.

This option can be overridden on a per-event basis with theEvent Object `editable` property.

Prior to version 1.3, the *draggable* option was used instead.

disableDragging 1.3

Disables all event dragging, even when events are editable.

```
Boolean, default: false
```

disableResizing 1.3

Disables all event resizing, even when events are editable.

```
Boolean, default: false
```

dragRevertDuration 1.3

Time it takes for an event to revert to its original position after an unsuccessful drag.

```
Integer, default: 500
```

Time is in milliseconds (1 second = 1000 milliseconds).

Prior to version 1.3, this option was known as *eventRevertDuration*

dragOpacity 1.3

The opacity of an event while it is being dragged.

```
Float/View Option Hash, default:
```

```
{
    // for agendaWeek and agendaDay
    agenda: .5,

    // for all other views
    '': 1.0
```

```
}
```

Float values range from 0.0 to 1.0.

Specify a single number to affect all views, or a View Option Hash to target specific views (which is what the default does).

A *View Option Hash* can only be provided in versions 1.4 and later.

Prior to version 1.3, this option was known as *eventDragOpacity*.

eventDragStart (callback)

Triggered when event dragging begins.

```
function( event, jsEvent, ui, view ) { }
```

`event` is an Event Object that hold the event's information (date, title, etc).

`jsEvent` holds the native javascript event with low-level information such as click coordinates.

`ui` holds the jQuery UI object.

`view` holds the current View Object.

eventDragStop (callback)

Triggered when event dragging stops.

```
function( event, jsEvent, ui, view ) { }
```

This callback is *guaranteed* to be triggered after the user drags an event, even if the event doesn't change date/time. It is triggered before the event's information has been modified (if moved to a new date/time) and before theeventDrop callback is triggered.

`event` is an Event Object that hold the event's information (date, title, etc).

`jsEvent` holds the native JavaScript event with low-level information such as mouse coordinates.

`ui` holds the jQuery UI object.

`view` holds the current View Object.

eventDrop (callback)

Triggered when dragging stops and the event has moved to a *different* day/time.

```
function( event, dayDelta, minuteDelta, allDay, revertFunc, jsEvent, ui, view ) { }
```

`event` is an Event Object that hold the event's information (date, title, etc).

`dayDelta` holds the number of days the event was moved forward (a positive number) or backwards (a negative number).

`minuteDelta` holds the number of minutes the event was moved forward (a positive number) or backwards (a negative number). Only useful for the agenda views. In other views, `0` is passed in.

`dayDelta` and `minuteDelta` are elegant for dealing with multi-day and repeating events. If updating a remote database, just add these values to the start and end times of all events with the given `event.id`.

`allDay` is `true` if the event has been dropped on a day in month view, or the "all-day" slot in the agenda views. It will be `false` if dropped on a slot in the agenda views (meaning it has been assigned a time).

`revertFunc` is a function that, if called, reverts the event's start/end date to the values before the drag. This is useful if an ajax call should fail.

`jsEvent` holds the native JavaScript event with low-level information such as mouse coordinates.

`ui` holds the jQuery UI object.

`view` holds the current View Object.

Here is an example demonstrating most of these arguments:

```
$('#calendar').fullCalendar({

    events: [

        // events here

    ],

    editable: true,

    eventDrop: function(event,dayDelta,minuteDelta,allDay,revertFunc) {



        alert(

            event.title + " was moved " +

            dayDelta + " days and " +

            minuteDelta + " minutes."

        );



        if (allDay) {

            alert("Event is now all-day");

        }else{

            alert("Event has a time-of-day");

        }
```

```
        if (!confirm("Are you sure about this change?")) {

            revertFunc();

        }



    }

});
```

Prior to version 1.4, the *allDay* paramameter was not included.

Prior to version 1.3, the *revertFunc* parameter was not included.

eventResizeStart (callback) 1.3

Triggered when event resizing begins.

```
function( event, jsEvent, ui, view ) { }
```

`event` is an Event Object that hold the event's information (date, title, etc).

`jsEvent` holds the native JavaScript event with low-level information such as mouse coordinates.

`ui` holds the jQuery UI object.

`view` holds the current View Object.

eventResizeStop (callback) 1.3

Triggered when event resizing stops.

```
function( event, jsEvent, ui, view ) { }
```

This callback is *guaranteed* to be triggered after the user resizes an event, even if the event doesn't change in duration. It is triggered before the event's information has been modified (if changed in duration) and before theeventResize callback is triggered.

`event` is an Event Object that hold the event's information (date, title, etc).

`jsEvent` holds the native JavaScript event with low-level information such as mouse coordinates.

`ui` holds the jQuery UI object.

`view` holds the current View Object.

eventResize (callback) 1.3

Triggered when resizing stops and the event has changed in duration.

```
function( event, dayDelta, minuteDelta, revertFunc, jsEvent, ui, view ) { }
```

`event` is an Event Object that hold the event's information (date, title, etc).

`dayDelta` holds the number of days the event's end date was moved forward (a positive number) or backwards (a negative number).

`minuteDelta` holds the number of minutes the event's end time was moved forward (a positive number) or backwards (a negative number). Only useful for the agenda views. In other views, `0` is passed in.

`dayDelta` and `minuteDelta` are elegant for dealing with multi-day and repeating events. If updating a remote database, just add these values to the *end* of all events with the given `event.id`.

`revertFunc` is a function that, if called, reverts the event's end date to the value before the drag. This is useful if an ajax call should fail.

`jsEvent` holds the native javascript event with low-level information such as mouse coordinates.

`ui` holds the jQuery UI object.

`view` holds the current View Object.

Here is an example demonstrating most of these arguments:

```
$('#calendar').fullCalendar({

    events: [

        // events here

    ],

    editable: true,

    eventResize: function(event,dayDelta,minuteDelta,revertFunc) {



        alert(

            "The end date of " + event.title + "has been moved " +

            dayDelta + " days and " +

            minuteDelta + " minutes."

        );



        if (!confirm("is this okay?")) {

            revertFunc();

        }



    }

});
```

12. Dropping External Elements

droppable 1.4.7

Determines if jQuery UI draggables can be dropped onto the calendar.

```
Boolean, default: false
```

This option operates with jQuery UI draggables. You must downloadthe appropriate jQuery UI files and initialize a draggable element. Additionally, you must set the calendar's `droppable` option to `true`.

Here is how you might initialize an element that can be dragged onto a calendar:

```
$('#my-draggable').draggable({

    revert: true,      // immediately snap back to original position

    revertDuration: 0  //

});
```

```
$('#calendar').fullCalendar({

    droppable: true,

    drop: function(date, allDay) {

        alert("Dropped on " + date + " with allDay=" + allDay);

    }

});
```

How can I use this to add events???

Good question. It is a common need to have an "external list of events" that can be dragged onto the calendar.

While the `droppable` option deals with generic jQuery UI draggables and is not specifically tailored to adding events, it is possible to achieve this with a few lines of code. Follow the **external-dragging.html** example in FullCalendar's download. You can also view the example online.

In short, you must call renderEvent yourself in the drop callback.

Hopefully, this task will become more convenient with future API changes.

dropAccept 1.4.7

Provides a way to filter which elements can be dropped onto the calendar.

```
String/Function, default: "*"
```

By default, after setting a calendar' droppable option to `true`, the calendar will accept any draggables that are dropped onto the calendar. The `dropAccept` option allows the calendar be more selective about which elements can/can't be dropped.

The value of `dropAccept` can be a string jQuery selector. It can also be a function that accepts the draggable item as a single argument, and returns `true`if the element can be dropped onto the calendar.

In the following example, the first draggable (with id `"draggable1"`) can be dropped on the calendar, while the second draggable (with id `"draggable2"`) cannot:

```
...

<div id='calendar'></div>



<div id='draggable1' class='cool-event'></div>

<div id='draggable2'></div>
```

```
...
```

and here is the JavaScript:

```
$('#calendar').fullCalendar({

    droppable: true,

    dropAccept: '.cool-event',

    drop: function() {

        alert('dropped!');

    }

});



$('#draggable1').draggable();

$('#draggable2').draggable();
```

drop (callback) 1.4.7

Called when a valid jQuery UI draggable has been dropped onto the calendar.

```
function( date, allDay, jsEvent, ui ) { }
```

`date` holds the JavaScript Date object of where the draggable was dropped.

`allDay` holds a boolean of whether the draggable was dropped on an all-day cell (like in month view) or in a slot with a specific time (like in agenda view).

`jsEvent` holds the primitive JavaScript event, with information like mouse coordinates.

`ui` holds the jQuery UI information.

`this` holds the DOM element that has been dropped.

To see this callback function in action, view the droppable article or look atthis example.

13. Utilities

formatDate

Formats a Date object into a string.

```
$.fullCalendar.formatDate( date, formatString [, options ] ) -> String
```

Prior to version 1.3, formatDate accepted a very different format. See here.

`formatString` is a combination of any of the following commands:

- **s** - seconds
- **ss** - seconds, 2 digits
- **m** - minutes
- **mm** - minutes, 2 digits
- **h** - hours, 12-hour format
- **hh** - hours, 12-hour format, 2 digits
- **H** - hours, 24-hour format
- **HH** - hours, 24-hour format, 2 digits
- **d** - date number
- **dd** - date number, 2 digits
- **ddd** - date name, short
- **dddd** - date name, full
- **M** - month number

- **MM** - month number, 2 digits
- **MMM** - month name, short
- **MMMM** - month name, full
- **yy** - year, 2 digits
- **yyyy** - year, 4 digits
- **t** - 'a' or 'p'
- **tt** - 'am' or 'pm'
- **T** - 'A' or 'P'
- **TT** - 'AM' or 'PM'
- **u** - ISO8601 format
- **S** - 'st', 'nd', 'rd', 'th' for the date
- **W** - the ISO8601 week number

Special Characters:

`'...'`

literal text

`''`

single quote (represented by two single quotes)

`(...)`

only displays format if one of the enclosed variables is non-zero

The `options` parameter can be used to override default locale options, such as`monthNames`,`monthNamesShort`,`dayNames`, and `dayNamesShort`.

formatDates 1.3

Formats a date range (two Date objects) into a string.

```
$.fullCalendar.formatDates( date1, date2, formatString [, options ] ) -> String
```

`formatDates` uses all the commands from formatDate, but with two special forms:

`{...}`

switches to formatting the 2nd date

`[...]`

only displays the enclosed format if the current date is different from the alternate date in the same regards

parseDate

Parses a string into a Date object.

```
$.fullCalendar.parseDate( string ) -> Date
```

The string may be in ISO8601 format, IETF format, or a UNIX timestamp (in either integer or string form).

IETF format only correctly started working in version 1.3.2.

parseISO8601

Parses an ISO8601 string into a Date object.

```
$.fullCalendar.parseISO8601( string [, ignoreTimezone ] ) -> Date
```

More information about the ISO8601 format can be found here.

14. Google Calendar

Google Calendar

FullCalendar can display events from a public Google Calendar. It can serve as a backend that manages and persistently stores event data (a feature that FullCalendar currently lacks).

Before you code...

You must first make your Google Calendar public:

1. In the Google Calendar interface, locate the "My Calendar" box on the left.
2. Click the arrow next to the calendar you need.
3. A menu will appear. Click "Share this calendar."
4. Check "Make this calendar public."
5. Make sure "Share only my free/busy information" is **unchecked**.
6. Click "Save."

Then, you must obtain your calendar's XML feed URL:

1. In the Google Calendar interface, locate the "My Calendar" box on the left
2. Click the arrow next to the calendar you need.
3. A menu will appear. Click "Calendar settings."
4. In the "Calendar Address" section of the screen, click the XML badge.
5. Your feed's URL will appear.

Dependencies

Next, you must have all the required js/css files. This includes the standard fullcalendar.js and fullcalendar.css, **in addition to gcal.js**:

```
<script type='text/javascript' src='fullcalendar/gcal.js'></script>
```

Writing the code (version 1.5 and above)

Now it's time to initialize your calendar in JavaScript. You can simply put your feed's URL in the `events` option:

```
<script type='text/javascript'>



$(document).ready(function() {
```

```
    $('#calendar').fullCalendar({

        events: 'http://www.google.com/your_feed_url/'

    });

});



</script>
```

If you want to specify some Event Source options, you need to write things a little differently:

```
<script type='text/javascript'>



$(document).ready(function() {

    $('#calendar').fullCalendar({

        events: {

            url: 'http://www.google.com/your_feed_url/',

            className: 'gcal-event',           // an option!

            currentTimezone: 'America/Chicago' // an option!

        }

    });

});



</script>
```

Options

currentTimezone	a string like "America/Chicago". Consult

	http://php.net/manual/en/timezones.php for a full list.
editable	whether to allow dragging/resizing (default: `false`)
className	CSS class to attach to each event
	Any of the other Event Source options...

Timezones Gotchas

Sometimes it can be confusing as to why FullCalendar displays event times differently than the Google Calendar interface. There are the two factors involved in this:

- *the calendar's timezone*, accessed through "Calendar settings" after clicking the arrow next to the calendar's name
- *your Google Account's timezone*, accessed through the "Settings" link at the top right of the Google Calendar screen (near the "Sign out" link)

When both timezones are the same, you should have no problems. When they are different, FullCalendar will display times in the *calendar's* timezone. Thus, times will be different than what you see in the Google Calendar interface because they are being adjusted to the GMT of the calendar. The solution is to use the `currentTimezone` option. If this is set to the same timezone as your Google Account, all dates should appear consistent.

Multiple Google Calendars

You can specify multiple Google Calendars by using the `eventSources` option:

```
<script type='text/javascript'>



$(document).ready(function() {

    $('#calendar').fullCalendar({

        eventSources: [
```

```
            // source with no options
            "http://www.google.com/your_feed_url1/",

            // source with no options
            "http://www.google.com/your_feed_url2/",

            // source WITH options
            {
                url: "http://www.google.com/your_feed_url3/",
                className: 'nice-event'
            }
        ]
    });
});

</script>
```

Writing the Code (prior to version 1.5)

Prior to version 1.5, Google Calendar feeds were defined differently. This way will still continue to work until version 2.0, but it is deprecated:

```
<script type='text/javascript'>

$(document).ready(function() {
    $('#calendar').fullCalendar({
```

```
        events: $.fullCalendar.gcalFeed(
            "http://www.google.com/your_feed_url/"
        )
    });
});


</script>
```

Here is how you'd do the same thing, but with options:

```
<script type='text/javascript'>


$(document).ready(function() {
    $('#calendar').fullCalendar({
        events: $.fullCalendar.gcalFeed(
            "http://www.google.com/your_feed_url/",
            {
                className: 'gcal-event',           // an option!
                currentTimezone: 'America/Chicago' // an option!
            }
        )
    });


});
```

```
</script>
```

Prior to version 1.5, the only options that are available are `className`, `currentTimezone`, and `editable`.