DHTML Calendar Widget

Mihai Bazon, `<mihai_bazon@yahoo.com>`
© Dynarch.com 2002–2005, www.dynarch.com

March 7, 2005
calendar version: 1.0 "It is happening again"

```
$Id: reference.tex,v 1.23 2005/03/05 11:37:14 mishoo Exp $
```

Contents

1 Overview

The DHTML Calendar widget[1] is an (HTML) user interface element that gives end-users a friendly way to select date and time. It works in a web browser. The first versions only provided support for popup calendars, while starting with version 0.9 it also supports "flat" display. A "flat" calendar is a calendar that stays visible in the page all the time. In this mode it could be very useful for "blog" pages and other pages that require the calendar to be always present.

The calendar is compatible with most popular browsers nowadays. While it's created using web standards and it should generally work with any compliant browser, the following browsers were found to work: Mozilla/Firefox (the development platform), Netscape 6.0 or better, all other Gecko-based browsers, Internet Explorer 5.0 or better *for Windows*[2], Opera 7[3], Konqueror 3.1.2 and Apple Safari for MacOSX.

You can find the latest info and version at the calendar homepage:

<div align="center">

www.dynarch.com/projects/calendar

</div>

1.1 How does this thing work?

DHTML is not "another kind of HTML". It's merely a naming convention. DHTML refers to the combination of HTML, CSS, JavaScript and DOM. DOM (Document Object Model) is a set of interfaces that glues the other three together. In other words, DOM allows dynamic modification of an HTML page through a program. JavaScript is our programming language, since that's what browsers like. CSS is a way to make it look good ;-). So all this soup is generically known as DHTML.

Using DOM calls, the program dynamically creates a `<table>` element that contains a calendar for the given date and then inserts it in the document body. Then it shows this table at a specified position. Usually the position is related to some element in which the date needs to be displayed/entered, such as an input field.

By assigning a certain CSS class to the table we can control the look of the calendar through an external CSS file; therefore, in order to change the colors, backgrounds, rollover effects and other stuff, you can only change a CSS file—modification of the program itself is not necessary.

1.2 Project files

Here's a description of the project files, excluding documentation and example files.

- the main program file (`calendar.js`). This defines all the logic behind the calendar widget.

- the CSS files (`calendar-*.css`). Loading one of them is necessary in order to see the calendar as intended.

---

[1] by the term "widget" I understand a single element of user interface. But that's in Linux world. For those that did lots of Windows programming the term "control" might be more familiar

[2] people report that the calendar does not work with IE5/Mac. However, this browser was discontinued and we believe that supporting it doesn't worth the efforts, given the fact that it has the worst, buggiest implementation for DOM I've ever seen.

[3] under Opera 7 the calendar still lacks some functionality, such as keyboard navigation; also Opera doesn't seem to allow disabling text selection when one drags the mouse on the page; despite all that, the calendar is still highly functional under Opera 7 and looks as good as in other supported browsers.

- the language definition files (lang/calendar-*.js). They are plain JavaScript files that contain all texts that are displayed by the calendar. Loading one of them is necessary.

- helper functions for quick setup of the calendar (calendar-setup.js). You can do fine without it, but starting with version 0.9.3 this is the recommended way to setup a calendar.

1.3 License

© Dynarch.com 2002–2005, www.dynarch.com
Author: Mihai Bazon

The calendar is released under the GNU Lesser General Public License.

2 Quick startup

Installing the calendar used to be quite a task until version 0.9.3. Starting with 0.9.3 I have included the file calendar-setup.js whose goal is to assist you to setup a popup or flat calendar in minutes. You are encouraged to modify this file and *not* calendar.js if you need extra customization, but you're on your own.

First you have to include the needed scripts and style-sheet. Make sure you do this in your document's <head> section, also make sure you put the correct paths to the scripts.

```
<style type="text/css">@import url(calendar-win2k-1.css);</style>
<script type="text/javascript" src="calendar.js"></script>
<script type="text/javascript" src="lang/calendar-en.js"></script>
<script type="text/javascript" src="calendar-setup.js"></script>
```

2.1 Installing a popup calendar

Now suppose you have the following HTML:

```
<form ...>
  <input type="text" id="data" name="data" />
  <button id="trigger">...</button>
</form>
```

You want the button to popup a calendar widget when clicked? Just insert the following code immediately *after* the HTML form:

```
<script type="text/javascript">
  Calendar.setup(
    {
      inputField  : "data",        // ID of the input field
      ifFormat    : "%m %d, %Y",   // the date format
      button      : "trigger"      // ID of the button
    }
  );
</script>
```

The Calendar.setup function, defined in calendar-setup.js takes care of "patching" the button to display a calendar when clicked. The calendar is by default in single-click mode and linked with the given input field, so that when the end-user selects a date it will update the input field with the date in the given format and close the calendar. If you are a long-term user of the calendar you probably remember that for doing this you needed to write a couple functions and add an "onclick" handler for the button by hand.

By looking at the example above we can see that the function Calendar.setup receives only one parameter: a JavaScript object. Further, that object can have lots of properties that tell to the setup function how would we like to have the calendar. For instance, if we would like a calendar that closes at double-click instead of single-click we would also include the following: singleClick:false.

For a list of all supported parameters please see the section 2.3.

2.2 Installing a flat calendar

Here's how to configure a flat calendar, using the same `Calendar.setup` function. First, you should have an empty element with an ID. This element will act as a container for the calendar. It can be any block-level element, such as DIV, TABLE, etc. We will use a DIV in this example.

```
<div id="calendar-container"></div>
```

Then there is the JavaScript code that sets up the calendar into the "calendar-container" DIV. The code can occur anywhere in HTML *after* the DIV element.

```
<script type="text/javascript">
  function dateChanged(calendar) {
    // Beware that this function is called even if the end-user only
    // changed the month/year.  In order to determine if a date was
    // clicked you can use the dateClicked property of the calendar:
    if (calendar.dateClicked) {
      // OK, a date was clicked, redirect to /yyyy/mm/dd/index.php
      var y = calendar.date.getFullYear();
      var m = calendar.date.getMonth();      // integer, 0..11
      var d = calendar.date.getDate();       // integer, 1..31
      // redirect...
      window.location = "/" + y + "/" + m + "/" + d + "/index.php";
    }
  };

  Calendar.setup(
    {
      flat         : "calendar-container", // ID of the parent element
      flatCallback : dateChanged           // our callback function
    }
  );
</script>
```

2.3 `Calendar.setup` in detail

Following there is the complete list of properties interpreted by Calendar.setup. All of them have default values, so you can pass only those which you would like to customize. Anyway, you *must* pass at least one of `inputField`, `displayArea` or `button`, for a popup calendar, or `flat` for a flat calendar. Otherwise you will get a warning message saying that there's nothing to setup.

property	type	description	default
inputField	string	The ID of your input field.	null
displayArea	string	This is the ID of a <span>, <div>, or any other element that you would like to use to display the current date. This is generally useful only if the input field is hidden, as an area to display the date.	null
button	string	The ID of the calendar "trigger". This is an element (ordinarily a button or an image) that will dispatch a certain event (usually "click") to the function that creates and displays the calendar.	null
eventName	string	The name of the event that will trigger the calendar. The name should be without the "on" prefix, such as "click" instead of "onclick". Virtually all users will want to let this have the default value ("click"). Anyway, it could be useful if, say, you want the calendar to appear when the input field is focused and have no trigger button (in this case use "focus" as the event name).	"click"

property	type	description	default
ifFormat	string	The format string that will be used to enter the date in the input field. This format will be honored even if the input field is hidden.	"%Y/%m/%d"
daFormat	string	Format of the date displayed in the displayArea (if specified).	"%Y/%m/%d"
singleClick	boolean	Wether the calendar is in "single-click mode" or "double-click mode". If true (the default) the calendar will be created in single-click mode.	true
disableFunc	function	A function that receives a JS Date object. It should return `true` if that date has to be disabled, `false` otherwise. DEPRECATED (see below).	null
dateStatusFunc	function	A function that receives a JS Date object and returns a boolean or a string. This function allows one to set a certain CSS class to some date, therefore making it look different. If it returns `true` then the date will be disabled. If it returns `false` nothing special happens with the given date. If it returns a string then that will be taken as a CSS class and appended to the date element. If this string is "disabled" then the date is also disabled (therefore is like returning `true`). For more information please also refer to section 5.3.8.	null
firstDay	integer	Specifies which day is to be displayed as the first day of week. Possible values are 0 to 6; 0 means Sunday, 1 means Monday, ..., 6 means Saturday. The end user can easily change this too, by clicking on the day name in the calendar header.	0
weekNumbers	boolean	If "true" then the calendar will display week numbers.	true
align	string	Alignment of the calendar, relative to the reference element. The reference element is dynamically chosen like this: if a displayArea is specified then it will be the reference element. Otherwise, the input field is the reference element. For the meaning of the alignment characters please section 5.3.11.	"Bl"
range	array	An array having exactly 2 elements, integers. (!) The first [0] element is the minimum year that is available, and the second [1] element is the maximum year that the calendar will allow.	[1900, 2999]
flat	string	If you want a flat calendar, pass the ID of the parent object in this property. If not, pass `null` here (or nothing at all as `null` is the default value).	null
flatCallback	function	You should provide this function if the calendar is flat. It will be called when the date in the calendar is changed with a reference to the calendar object. See section 2.2 for an example of how to setup a flat calendar.	null
onSelect	function	If you provide a function handler here then you have to manage the "click-on-date" event by yourself. Look in the calendar-setup.js and take as an example the onSelect handler that you can see there.	null
onClose	function	This handler will be called when the calendar needs to close. You don't need to provide one, but if you do it's your responsibility to hide/destroy the calendar. You're on your own. Check the calendar-setup.js file for an example.	null

Note that this code does more actually; the only *required* fields are those marked with "//*"—that is, the ID of the input field and the ID of the button need to be passed to Calendar.setup in order for the calendar to be properly assigned to this input field. As one can easily guess from the argument names, the other arguments configure a certain date format, instruct the calendar to also include a time selector and display every year in the drop-down boxes (the "step" parameter)—instead of showing every other year as the default calendar does.

3.1.2 Simple field with calendar attached to an image

Same as the above, but the element that triggers the calendar is this time an image, not a button.

```
<input type="text" name="date" id="f_date_c" readonly="1" />
<img src="img.gif" id="f_trigger_c"
     style="cursor: pointer; border: 1px solid red;"
     title="Date selector"
     onmouseover="this.style.background='red';"
     onmouseout="this.style.background=''" />
<script type="text/javascript">
    Calendar.setup({
        inputField     :     "f_date_c",
        ifFormat       :     "%B %e, %Y",
        button         :     "f_trigger_c",
        align          :     "Tl",
        singleClick    :     false
    });
</script>
```

Note that the same 2 parameters are required as in the previous case; the difference is that the 'button' parameter now gets the ID of the image instead of the ID of the button. But the event is the same: at 'onclick' on the element that is passed as 'button', the calendar will be shown.

The above code additionally sets an alignment mode—the parameters are described in 5.3.11.

3.1.3 Hidden field, plain text triggers

Sometimes, to assure that the date is well formatted, you might want not to allow the end user to write a date manually. This can easily be achieved with an input field by setting its readonly attribute, which is defined by the HTML4 standard; however, here's an even nicer approach: our calendar widget allows you to use a hidden field as the way to pass data to server, and a "display area" to show the end user the selected date. The "display area" can be any HTML element, such as a DIV or a SPAN or whatever—we will use a SPAN in our sample.

```
<input type="hidden" name="date" id="f_date_d" />

<p>Your birthday:
    <span style="background-color: #ff8; cursor: default;"
          onmouseover="this.style.backgroundColor='#ff0';"
          onmouseout="this.style.backgroundColor='#ff8';"
          id="show_d"
    >Click to open date selector</span>.</p>

<script type="text/javascript">
    Calendar.setup({
        inputField     :     "f_date_d",
        ifFormat       :     "%Y/%d/%m",
        displayArea    :     "show_d",
```

```
        daFormat         :       "%A, %B %d, %Y",
    });
</script>
```

The above code will configure a calendar attached to the hidden field and to the SPAN having the id="show_d". When the SPAN element is clicked, the calendar opens and allows the end user to chose a date. When the date is chosen, the input field will be updated with the value in the format "%Y/%d/%m", and the SPAN element will display the date in a friendlier format (defined by "daFormat").

Beware that using this approach will make your page unfunctional in browsers that do not support JavaScript or our calendar.

3.1.4   2 Linked fields, no trigger buttons

Supposing you want to create 2 fields that hold an interval of exactly one week. The first is the starting date, and the second is the ending date. You want the fields to be automatically updated when some date is clicked in one or the other, in order to keep exactly one week difference between them.

```
<input type="text" name="date" id="f_date_a" />
<input type="text" name="date" id="f_calcdate" />

<script type="text/javascript">
    function catcalc(cal) {
        var date = cal.date;
        var time = date.getTime()
        // use the _other_ field
        var field = document.getElementById("f_calcdate");
        if (field == cal.params.inputField) {
            field = document.getElementById("f_date_a");
            time -= Date.WEEK; // substract one week
        } else {
            time += Date.WEEK; // add one week
        }
        var date2 = new Date(time);
        field.value = date2.print("%Y-%m-%d %H:%M");
    }
    Calendar.setup({
        inputField       :       "f_date_a",
        ifFormat         :       "%Y-%m-%d %H:%M",
        showsTime        :       true,
        timeFormat       :       "24",
        onUpdate         :       catcalc
    });
    Calendar.setup({
        inputField       :       "f_calcdate",
        ifFormat         :       "%Y-%m-%d %H:%M",
        showsTime        :       true,
        timeFormat       :       "24",
        onUpdate         :       catcalc
    });
</script>
```

The above code will configure 2 input fields with calendars attached, as usual. The first thing to note is that there's no trigger button—in such case, the calendar will popup when one clicks into the input field. Using the onUpdate parameter, we pass a reference to a function of ours that will get called after a date was selected. In that function we determine what field was updated and we compute the date in the other input field such that it keeps a one week difference between the two. Enjoy! :-)

3.2   Flat calendars

This sample can be found in "simple-2.html". It will configure a flat calendar that is always displayed in the page, in the DIV having the id="calendar-container". When a date is clicked our function hander gets called (dateChanged) and it will compute an URL to jump to based on the selected date, then use window.location to visit the new link.

```
<div style="float: right; margin-left: 1em; margin-bottom: 1em;"
id="calendar-container"></div>

<script type="text/javascript">
  function dateChanged(calendar) {
    // Beware that this function is called even if the end-user only
    // changed the month/year.  In order to determine if a date was
    // clicked you can use the dateClicked property of the calendar:
    if (calendar.dateClicked) {
      // OK, a date was clicked, redirect to /yyyy/mm/dd/index.php
      var y = calendar.date.getFullYear();
      var m = calendar.date.getMonth();      // integer, 0..11
      var d = calendar.date.getDate();       // integer, 1..31
      // redirect...
      window.location = "/" + y + "/" + m + "/" + d + "/index.php";
    }
  };

  Calendar.setup(
    {
      flat         : "calendar-container", // ID of the parent element
      flatCallback : dateChanged           // our callback function
    }
  );
</script>
```

3.3   Highlight special dates

So you want to display certain dates in a di erent color, or with bold font, or whatever, right? Well, no problem—our calendar can do this as well. It doesn't matter if it's a flat or popup calendar—we'll use a flat one for this sample. The idea, however, is that you need to have the dates in an array or a JavaScript object—whatever is suitable for your way of thinking—and use it from a function that returns a value, telling the calendar what kind of date is the passed one.

   Too much talking, here's the code ;-)

```
<!-- this goes into the <head> tag -->
<style type="text/css">
  .special { background-color: #000; color: #fff; }
</style>

<!-- and the rest inside the <body> -->
<div style="float: right; margin-left: 1em; margin-bottom: 1em;"
id="calendar-container"></div>

<script type="text/javascript">
  var SPECIAL_DAYS = {
    0 : [ 13, 24 ],// special days in January
    2 : [ 1, 6, 8, 12, 18 ],// special days in March
    8 : [ 21, 11 ] // special days in September
  };

  function dateIsSpecial(year, month, day) {
```

```
      var m = SPECIAL_DAYS[month];
      if (!m) return false;
      for (var i in m) if (m[i] == day) return true;
      return false;
    };

  function dateChanged(calendar) {
    // Beware that this function is called even if the end-user only
    // changed the month/year.  In order to determine if a date was
    // clicked you can use the dateClicked property of the calendar:
    if (calendar.dateClicked) {
      // OK, a date was clicked, redirect to /yyyy/mm/dd/index.php
      var y = calendar.date.getFullYear();
      var m = calendar.date.getMonth();      // integer, 0..11
      var d = calendar.date.getDate();       // integer, 1..31
      // redirect...
      window.location = "/" + y + "/" + m + "/" + d + "/index.php";
    }
  };

  function ourDateStatusFunc(date, y, m, d) {
    if (dateIsSpecial(y, m, d))
      return "special";
    else
      return false; // other dates are enabled
      // return true if you want to disable other dates
  };

  Calendar.setup(
    {
      flat           : "calendar-container", // ID of the parent element
      flatCallback : dateChanged,            // our callback function
      dateStatusFunc : ourDateStatusFunc
    }
  );
</script>
```

So the above code creates a normal flat calendar, like in the previous sample. We hook into it with the function "ourDateStatusFunc", which receives a date object as the first argument, and also the year, month, date as the next 3 arguments (normally, you can extract year, month, date from the first parameter too, but we pass them separately for convenience, as it's very likely that they are going to be used in this function).

So, this function receives a date. It can return false if you want no special action to be taken on that date, true if that date should be disabled (unselectable), or a string if you want to assign a special CSS class to that date. We return "special" for the dates that we want to highlight—and note that we defined a "special" look for them in the CSS section.

I used a simple approach here to define what dates are special. There's a JavaScript object (the SPECIAL_DAYS global variable) which holds an array of dates for each month. Month numbers start at zero (January). Months that don't contain special dates can be absent from this object. Note that the way to implement this is completely separated from the calendar code—therefore, feel free to use your imagination if you have better ideas. :-)

3.4   Select multiple dates

Starting version 1.0, the calendar is able to handle multiple dates selection. You just need to pass the "multiple" parameter to Calendar.setup and add some special code that interprets the selection once the calendar is closed.

```
<a id="trigger" href="#">[open calendar...]</a>
<div id="output"></div>
```

```
<script type="text/javascript">//<![CDATA[
    // the default multiple dates selected,
    // first time the calendar is displayed
    var MA = [];

    function closed(cal) {

      // here we'll write the output; this is only for example.  You
      // will normally fill an input field or something with the dates.
      var el = document.getElementById("output");

      // reset initial content.
      el.innerHTML = "";

      // Reset the "MA", in case one triggers the calendar again.
      // CAREFUL!  You don't want to do "MA = [];".  We need to modify
      // the value of the current array, instead of creating a new one.
      // Calendar.setup is called only once! :-)  So be careful.
      MA.length = 0;

      // walk the calendar's multiple dates selection hash
      for (var i in cal.multiple) {
        var d = cal.multiple[i];
        // sometimes the date is not actually selected,
        // so let's check
        if (d) {
          // OK, selected.  Fill an input field or something.
          el.innerHTML += d.print("%A, %Y %B %d") + "<br />";
          // and push it in the "MA", in case one triggers the calendar again.
          MA[MA.length] = d;
        }
      }
      cal.hide();
      return true;
    };

    Calendar.setup({
      align      : "BR",
      showOthers : true,
      multiple   : MA, // pass the initial or computed array of multiple dates
      onClose    : closed,
      button     : "trigger"
    });
//]]></script>
```

The above code creates a popup calendar and passes to it an array of dates, which is initially empty, in the "multiple" argument. When the calendar is closed it will call our "closed" function handler; in this handler we determine what dates were actually selected, inspecting the "cal.multiple" property, we display them in a DIV element right next to the <a> element that opens the calendar, and we reinitialize the global array of selected dates (which will be used if the end user opens the calendar again). I guess the code speaks for itself, right? :-)

4   The Calendar object overview

Basically you should be able to setup the calendar with the function presented in the previous section. However, if for some reason Calendar.setup doesn't provide all the functionality that you need and you want to tweak into the process of creating and configuring the calendar "by hand", then this section is the way to go.

The file `calendar.js` implements the functionality of the calendar. All (well, almost all) functions and variables are embedded in the JavaScript object "Calendar".

You can instantiate a Calendar object by calling the constructor, like this: `var cal = new Calendar(...)`. We will discuss the parameters later. After creating the object, the variable `cal` will contain a reference to it. You can use this reference to access further options of the calendar, for instance:

```
cal.weekNumbers = false; // do not display week numbers
cal.showsTime = true;     // include a time selector
cal.setDateFormat("%Y.%m.%d %H:%M"); // set this format: 2003.12.31 23:59
cal.setDisabledHandler(function(date, year, month, day) {
  // verify date and return true if it has to be disabled
  // ''date'' is a JS Date object, but if you only need the
  // year, month and/or day you can get them separately as
  // next 3 parameters, as you can see in the declaration
  if (year == 2004) {
    // disable all dates from 2004
    return true;
  }
  return false;
});
```

etc. Prior to version 0.9.3 this was the only way to configure it. The `Calendar.setup` function, documented in section 2, basically does the same things (actually more) in order to setup the calendar, based on the parameters that you provided.

4.1 Creating a calendar

The calendar is created by following some steps (even the function `Calendar.setup`, described in section 2, does the same). While you can skip optional (marked "opt") steps if you're happy with the defaults, please respect the order below.

1. *Instantiate* a `Calendar` object. Details about this in section 5.1.

opt Set the `weekNumbers` property to `false` if you don't want the calendar to display week numbers.

opt Set the `showsTime` property to `true` if you want the calendar to also provide a time selector.

opt Set the `time24` property to `false` if you want the time selector to be in 12-hour format. Default is 24-hour format. This property only has e ect if you also set `showsTime` to `true`.

opt Set the range of years available for selection (see section 5.3.15). The default range is [1970..2050].

opt Set the `getDateStatus` property. You should pass here a function that receives a JavaScript `Date` object and returns `true` if the given date should be disabled, false otherwise (details in section 5.3.7).

opt Set a date format. Your handler function, passed to the calendar constructor, will be called when a date is selected with a reference to the calendar and a date string in this format.

2. *Create* the HTML elements related to the calendar. This step practically puts the calendar in your HTML page. You simply call `Calendar.create()`. You can give an optional parameter if you wanna create a flat calendar (details in section 5.3.1).

opt Initialize the calendar to a certain date, for instance from the input field.

3. Show the calendar (details in section 5.3.9).

4.2 Order does matter ;-)

As you could see in the previous section, there are more steps to be followed in order to setup the calendar. This happens because there are two different things that need to be accomplished: first there is the JavaScript object, that is created with new `Calendar(...)`. Secondly there are the HTML elements that actually lets you see and manipulate the calendar.

[Those that did UI[4] programming, no matter in what language and on what platform, may be familiar with this concept. First there is the object in memory that lets you manipulate the UI element, and secondly there is the UI element (known as "control", "window", "widget", etc.), also in memory but you don't usually access it directly.]

By instantiating the calendar we create the JavaScript object. It lets us configure some properties and it also knows how to create the UI element (the HTML elements actually) that will eventually be what the end-user sees on screen. Creation of the HTML element is accomplished by the function `Calendar.create`. It knows how to create popup or flat calendars. This function is described in section 5.3.1.

Some properties need to be set prior to creating the HTML elements, because otherwise they wouldn't have any effect. Such a property is weekNumbers—it has the default value "true", and if you don't want the calendar to display the week numbers you have to set it to false. If, however, you do that *after* calling `Calendar.create` the calendar would still display the week numbers, because the HTML elements are already created (including the `<td>`-s in the `<table>` element that should contain the week numbers). For this reason the order of the steps above is important.

Another example is when you want to show the calendar. The "create" function does create the HTML elements, but they are initially hidden (have the style "display: none") unless the calendar is a flat calendar that should be always visible in the page. Obviously, the `Calendar.show` function should be called *after* calling `Calendar.create`.

4.3 Caching the object

Suppose the end-user has popped up a calendar and selects a date. The calendar then closes. What really happens now?

There are two approaches. The first (used in very old versions of the calendar) was to drop completely the Calendar object and when the end-user pops up the calendar again to create another one. This approach is bad for more reasons:

- creating the JavaScript object and HTML elements is time-consuming

- we may loose some end-user preferences (i.e. he might prefer to have Monday for the first day of week and probably already clicked it the first time when the calendar was opened, but now he has to do it again)

The second approach, implemented by the `Calendar.setup` function, is to cache the JavaScript object. It does this by checking the global variable `window.calendar` and if it is not null it assumes it is the created Calendar object. When the end-user closes the calendar, our code will only call "hide" on it, therefore keeping the JavaScript object and the HTML elements in place.

CAVEAT: Since time selection support was introduced, this "object caching" mechanism has the following drawback: if you once created the calendar with the time selection support, then other items that may not require this functionality will still get a calendar with the time selection support enabled. And reciprocal. ;-) Hopefully this will be corrected in a later version, but for now it doesn't seem such a big problem.

4.4 Callback functions

You might rightfully wonder how is the calendar related to the input field? Who tells it that it has to update *that* input field when a date is selected, or that it has to jump to *that* URL when a date is clicked in flat mode?

All this magic is done through callback functions. The calendar doesn't know anything about the existence of an input field, nor does it know where to redirect the browser when a

---

[4]user interface

date is clicked in flat mode. It just calls your callback when a particular event is happening, and you're responsible to handle it from there. For a general purpose library I think this is the best model of making a truly reusable thing.

The calendar supports the following user callbacks:

onSelect — this gets called when the end-user changes the date in the calendar. Documented in section 5.1.

onClose — this gets called when the calendar should close. It's user's responsibility to close the calendar. Details in section 5.1.

getDateStatus — this function gets called for any day in a month, just before displaying the month. It is called with a JavaScript Date object and should return true if that date should be disabled, false if it's an ordinary date and no action should be taken, or it can return a string in which case the returned value will be appended to the element's CSS class (this way it provides a powerful way to make some dates "special", i.e. highlight them di erently). Details in section 5.3.8.

5   The Calendar object API reference

5.1   `Calendar` constructor

Synopsis:

```
var calendar = Calendar(firstDayOfWeek, date, onSelect, onClose);
```

Parameters are as follows:

firstDayOfWeek — specifies which day is to be displayed as the first day of week. Possible values are 0 to 6; 0 means Sunday, 1 means Monday, ..., 6 means Saturday.

date — a JavaScript Date object or `null`. If `null` is passed then the calendar will default to today date. Otherwise it will initialize on the given date.

onSelect — your callback for the "onChange" event. See above.

onClose — your callback for the "onClose" event. See above.

The `onSelect` **event**

Here is a typical implementation of this function:

```
function onSelect(calendar, date) {
  var input_field = document.getElementById("date");
  input_field.value = date;
};
```

date is in the format selected with `calendar.setDateFormat` (see section 5.3.5). This code simply updates the input field. If you want the calendar to be in single-click mode then you should also close the calendar after you updated the input field, so we come to the following version:

```
function onSelect(calendar, date) {
  var input_field = document.getElementById("date");
  input_field.value = date;
  if (calendar.dateClicked) {
    calendar.callCloseHandler(); // this calls "onClose" (see above)
  }
};
```

Note that we checked the member variable `dateClicked` and only hide the calendar if it's `true`. If this variable is `false` it means that no date was actually selected, but the user only changed the month/year using the navigation buttons or the menus. We don't want to hide the calendar in that case.

The onClose **event**

This event is triggered when the calendar should close. It should hide or destroy the calendar object—the calendar itself just triggers the event, but it won't close itself.

A typical implementation of this function is the following:

```
function onClose(calendar) {
  calendar.hide();
  // or calendar.destroy();
};
```

5.2 Useful member variables (properties)

After creating the Calendar object you can access the following properties:

- date — is a JavaScript Date object. It will always reflect the date shown in the calendar (yes, even if the calendar is hidden).

- isPopup — if this is true then the current Calendar object is a popup calendar. Otherwise (false) we have a flat calendar. This variable is set from Calendar.create and has no meaning before this function was called.

- dateClicked — particularly useful in the onSelect handler, this variable tells us if a date was really clicked. That's because the onSelect handler is called even if the end-user only changed the month/year but did not select a date. We don't want to close the calendar in that case.

- weekNumbers — if true (default) then the calendar displays week numbers. If you don't want week numbers you have to set this variable to false *before* calling Calendar.create.

- showsTime – if you set this to true (it is false by default) then the calendar will also include a time selector.

- time24 – if you set this to false then the time selector will be in 12-hour format. It is in 24-hour format by default.

- firstDayOfWeek — specifies the first day of week (0 to 6, pass 0 for Sunday, 1 for Monday, ..., 6 for Saturday). This variable is set from constructor, but you still have a chance to modify it *before* calling Calendar.create.

There are lots of other member variables, but one should access them only through member functions so I won't document them here.

5.3 Public methods

5.3.1 Calendar.create

This function creates the a erent HTML elements that are needed to display the calendar. You should call it after setting the calendar properties. Synopsis:

```
calendar.create(); // creates a popup calendar
  // -- or --
calendar.create(document.getElementById(parent_id)); // makes a flat calendar
```

It can create a popup calendar or a flat calendar. If the "parent" argument is present (it should be a *reference*—not ID—to an HTML element) then a flat calendar is created and it is inserted in the given element.

At any moment, given a reference to a calendar object, we can inspect if it's a popup or a flat calendar by checking the boolean member variable isPopup:

```
if (calendar.isPopup) {
  // this is a popup calendar
} else {
  // this is a flat calendar
}
```

5.3.2 Calendar.callHandler

This function calls the first user callback (the onSelect handler) with the required parameters.

5.3.3 Calendar.callCloseHandler

This function calls the second user callback (the onClose handler). It's useful when you want to have a "single-click" calendar—just call this in your onSelect handler, if a date was clicked.

5.3.4 Calendar.hide

Call this function to hide the calendar. The calendar object and HTML elements will not be destroyed, thus you can later call one of the show functions on the same element.

5.3.5 Calendar.setDateFormat

This function configures the format in which the calendar reports the date to your "onSelect" handler. Call it like this:

calendar.setDateFormat("%y/%m/%d");

As you can see, it receives only one parameter, the required format. The magic characters are the following:

%a	abbreviated weekday name
%A	full weekday name
%b	abbreviated month name
%B	full month name
%C	century number
%d	the day of the month (00 .. 31)
%e	the day of the month (0 .. 31)
%H	hour (00 .. 23)
%I	hour (01 .. 12)
%j	day of the year (000 .. 366)
%k	hour (0 .. 23)
%l	hour (1 .. 12)
%m	month (01 .. 12)
%M	minute (00 .. 59)
%n	a newline character
%p	"PM" or "AM"
%P	"pm" or "am"
%S	second (00 .. 59)
%s	number of seconds since Epoch (since Jan 01 1970 00:00:00 UTC)
%t	a tab character
%U, %W, %V	the week number[5]
%u	the day of the week (1 .. 7, 1 = MON)
%w	the day of the week (0 .. 6, 0 = SUN)
%y	year without the century (00 .. 99)
%Y	year including the century (ex. 1979)
%%	a literal % character

5.3.6 Calendar.setTtDateFormat

Has the same prototype as Calendar.setDateFormat, but refers to the format of the date displayed in the "status bar" when the mouse is over some date.

---

[5]There are more algorithms for computing the week number. All three specifiers currently implement the same one, as defined by ISO 8601: "the week 01 is the week that has the Thursday in the current year, which is equivalent to the week that contains the fourth day of January. Weeks start on Monday."

5.3.7 Calendar.setDisabledHandler

This function allows you to specify a callback function that checks if a certain date must be disabled by the calendar. You are responsible to write the callback function. Synopsis:

```
function disallowDate(date) {
  // date is a JS Date object
  if (  date.getFullYear() == 2003 &&
        date.getMonth()     == 6 /* July, it's zero-based */ &&
        date.getDate()      == 5  ) {
    return true; // disable July 5 2003
  }
  return false; // enable other dates
};
```

```
calendar.setDisabledHandler(disallowDate);
```

If you change this function in "real-time", meaning, without creating a new calendar, then you have to call calendar.refresh() to make it redisplay the month and take into account the new disabledHandler. Calendar.setup does this, so you have no such trouble with it.

Note that disallowDate should be very fast, as it is called for each date in the month. Thus, it gets called, say, 30 times before displaying the calendar, and 30 times when the month is changed. Tests I've done so far show that it's still good, but in the future I might switch it to a di erent design (for instance, to call it once per month and to return an array of dates that must be disabled).

This function should be considered deprecated in the favor of Calendar.setDateStatusHandler, described below.

5.3.8 Calendar.setDateStatusHandler

This function obsoletes Calendar.setDisabledHandler. You call it with a function parameter, but this function can return a boolean *or a string*. If the return value is a boolean (true or false) then it behaves just like setDisabledHandler, therefore disabling the date if the return value is true.

If the returned value is a string then the given date will gain an additional CSS class, namely the returned value. You can use this to highlight some dates in some way. Note that you are responsible for defining the CSS class that you return. If you return the string "disabled" then that date will be disabled, just as if you returned true.

Here is a simple scenario that shows what you can do with this function. The following should be present in some of your styles, or in the document head in a STYLE tag (but put it *after* the place where the calendar styles were loaded):

```
.special { background-color: #000; color: #fff; }
```

And you would use the following code before calling Calendar.create():

```
// this table holds your special days, so that we can automatize
// things a bit:
var SPECIAL_DAYS = {
    0 : [ 13, 24 ],             // special days in January
    2 : [ 1, 6, 8, 12, 18 ],    // special days in March
    8 : [ 21, 11 ],             // special days in September
    11 : [ 25, 28 ]             // special days in December
};
```

```
// this function returns true if the passed date is special
function dateIsSpecial(year, month, day) {
    var m = SPECIAL_DAYS[month];
    if (!m) return false;
    for (var i in m) if (m[i] == day) return true;
```

- c — the calendar displays vertically centered to the reference element. It might overlap it (that depends on the horizontal alignment).

- b — below the element but may overlap it (top margin of the calendar aligned to the top margin of the element).

- B — completely below the element (top margin of the calendar aligned to the bottom margin of the element).

Horizontal alignment   The second character in "`align`" can take one of the following values:

- L — completely to the left of the reference element (right margin of the calendar aligned to the left margin of the element).

- l — to the left of the element but may overlap it (left margin of the calendar aligned to the left margin of the element).

- c — horizontally centered to the element. Might overlap it, depending on the vertical alignment.

- r — to the right of the element but may overlap it (right margin of the calendar aligned to the right margin of the element).

- R — completely to the right of the element (left margin of the calendar aligned to the right margin of the element).

Default values   If the "`align`" parameter is missing the calendar will choose "`Br`".

5.3.12   `Calendar.setDate`

Receives a JavaScript Date object. Sets the given date in the calendar. If the calendar is visible the new date is displayed immediately.

```
calendar.setDate(new Date()); // go today
```

5.3.13   `Calendar.setFirstDayOfWeek`

Changes the first day of week. The parameter has to be a numeric value ranging from 0 to 6. Pass 0 for Sunday, 1 for Monday, ..., 6 for Saturday.

```
calendar.setFirstDayOfWeek(5); // start weeks on Friday
```

5.3.14   `Calendar.parseDate`

Use this function to parse a date given as string and to move the calendar to that date.
   The algorithm tries to parse the date according to the format that was previously set with `Calendar.setDateFormat`; if that fails, it still tries to get some valid date out of it (it doesn't read your thoughts, though).

```
calendar.parseDate("2003/07/06");
```

5.3.15   `Calendar.setRange`

Sets the range of years that are allowed in the calendar. Synopsis:

```
calendar.setRange(1970, 2050);
```

6 Side effects

The calendar code was intentionally embedded in an object to make it have as less as possible side effects. However, there are some—not harmful, after all. Here is a list of side effects; you can count they already happened after `calendar.js` was loaded.

1. The global variable `window.calendar` will be set to null. This variable is used by the calendar code, especially when doing drag & drop for moving the calendar. In the future I might get rid of it, but for now it didn't harm anyone.

2. The JavaScript `Date` object is modified. We add some properties and functions that