PHP iCalWriter

19/02/08

A set of classes to create iCalender-files (conforming to RFC 2445).

Contents

Introduction

What is it?

This is a set of PHP-classes to allow easy creation of iCalender-files. The definition is given in RFC 2445 (http://tools.ietf.org/html/rfc2445), which is rather long an complicated. iCalWriter should give you a set of easy methods to create standard-conform files. It is not intended to include every tiny bit of possibility that RFC 2445 describes, but to give easy access to the important and most used things. It allows three output-ways: download, save in the server-file system or show as a HTML-page.

If you need to add additional things which iCalWriter does not include yet, you can also do this. But in this case you need to figure out by yourself how to implement the RFC 2445 standard.

Version: This is the documentation for version 0.8. The first beta.

At the moment the following objects are included:

- iCalWriter – the base class
- iCalEvent – a single event
- iCalAudioAlarm, iCalDisplayAlarm, iCalEmailAlarm and iCalProcedureAlarm
- iCalRecurrence – repeating events or other iCalWriter-objects

Not included or to-do and other items. They will be added later.

Where to get it?

iCalWriter is hosted on sourceforge:
http://sourceforge.net/projects/phpicalwriter/

License

iCalwriter is licensed under the GNU Lesser General Public License:

iCalWriter

This is the base-class. It is in the file *iCalWriter.php*. You only need to import this file into your own scripts. All other files are included in this here.

Basic usage

To use iCalWriter you need to include (or require) this file first:

```
include(“iCalWriter.php”);
```

Then create a object and add some information:

```
$myCal = new iCalWriter();                   // create new object
$myCal->setDownloadOutput();                 // for download
$myCal->setFileName(“calender.ics”);         // set filename for download
...
```

After this you could start the output, add some events and finish output. That's it:

```
$myCal->start();                             // starts output
$myCal->add(myEvent);                        // adds event-data to the output
                                             // (see section about iCalEvent)
$myCal->end();                               // ends output
```

Methods

Now follows a more detailed explanation of each method.

add(*object*)

Adds the data of the object to the output. Only works, after output has started!

The *object* must be an iCalEvent (other objects might be added later).

Returns true if it could add the data to the output. Returns false, if it could not.

```
if ($myCal->start())
{
        $myCal->add($myEvent);
        ...
        $myCal->end();
}
```

end()

Ends the output. Always end the output using this method. Otherwise the result will not be a complete iCalender-file.

Returns true if it could end the output. Returns false, if it could not.

```
$myCal->end();
```

getiCalCalScale()

Returns the used calender scale.

```
echo($this->getiCalCalScale());
```

getiCalMethod()

Returns the used iCalMethod.

```
echo($this->getiCalMethod());
```

getiCalVersion()

Returns the iCalVersion-information. That is the version of the created iCalender file.

```
echo($this->getiCalVersion());
```

getOutputMethod()

Returns the used output-method. There are three possible return-values:

- html for HTML-output
- file for saving in the local (server) file system
- download for download

```
$outputMethod = $myCal->getOutputMethod();
```

getVersion()

Returns the version of the iCalenderWriter-class.

This is not the version of the iCalender-file-format (see *getiCalVersion*).

```
echo($this->getVersion());
```

isOutputStarted()

Returns true (1), if the output did start or false (0) if not.

```
$didOutputStart = $myCal->isOutputStarted();
```

replaceSpecialChars(*value*)

Should special chars (like German äöü) be replaced?

Replacing works as follows: ä -> ae; ö-> oe; u-> ue; ß->ss and so on.

value should be *0* or *false* for no and *1* or *true* for yes. Default is no.

This might be necessary, because some programs have trouble reading data with this special chars.

```
$myCal->replaceSpecialChars(1);
```

setDownloadOutput()

Sets output to download. This means the iCalender file is sent to the browser as a download. Normally the browser should ask the user, what to do with it.

Can only be used before the output starts!

If you use this, please also set the file-name with *setFileName()*!

```
$myCal->setDownloadOutput();
$myCal->setFileName("test.ics");
```

setFileName(*name*)

Sets the file-name for download or saving as a file.

In case of download you should only give the file name and no path information. There is no guarantee, that the file name will be used.

In case of saving as a file please give the full path and file name for saving under the local file system. This is **not** the URL but it must be a path!

```
$myCal->setFileName("/var/www/iCalWriter/test.ics");   // for saving as a file
```

setFileOutput()

Sets output to a file in the server-file-system. The file must be writeable. If not, then starting the output will fail.

Can only be used before the output starts!

If you use this, please also set the file-name and path with *setFileName()*!

```
$myCal->setFileOutput();
$myCal->setFileName("/var/www/iCalWriter/test.ics");
```

setHTMLOutput()

Sets output to html. This means it will be displayed as a website (without any <html> or <head> or <body> tags). This should be mainly used for testing purposes.

Can only be used before the output starts!

```
$myCal->setHTMLOutput();
```

setiCalCalScale(*scale*)

Sets the used calender-scale. The default value is GREGORIAN. At the moment there should be no need to change this, because (to my knowledge) not other scales exist. If you still want to change it, *scale* must be a string.

```
$myCal->setiCalCalScale(“SomeFutureScale”);
```

setiCalMethod(*method*)

Sets the method for the iCalender-object. The default method is PUBLISH. If you want to use another method, you can set it here.

```
$myCal->setiCalMethod(“REQUEST”);
```

setiCalVersion(*version*)

Sets the used iCalVersion. The default version is 2.0. If you want to use another version, than you can set it here. *version* must be a string. You should normally leave the iCalVersion to the default value!

```
$myCal->setiCalVersion(“5.2”);       // sets version to 5.2 (this version does not
                                     // exist yet!)
```

start()

Starts output. Before using this you should set the output method (or default HTML output will be used) and other things.
Returns true if it could start the output. Returns false, if it could not.

```
$myCal->start();
```

iCalEvent

This is the class that represents a single event.

It is in the *event.php* file. This file is included in *iCalWriter.php* so you do not need to include it a second time!

Basic usage

To use the iCalEvent-class you need to include the file *iCalWriter.php* (if you did not already do this).

```
include(“iCalWriter.php”);
```

Please do not include the file *event.php* directly!

Then you need to create the event-object and add everything you want to have in there.

```
$myEvent = new iCalEvent();
$myEvent->setStart("2008", "01", "20", 0, 1, "", 1, "14", "12");
                                        // sets start to 20/01/2008 at 14:12:00 local time
$myEvent->setEnd("2008", "01", "22", 1, 0, "", 1, "12");
                                        // sets end to 22/01/2008 at 12:00:00 (noon)
$myEvent->setShortDescription("Short description");
$myEvent->setLongDescription("A much longer description");
```

Now you just need to add the event to the iCalWriter-class you already created. Before adding the event you must start the output!

```
$myCal = new iCalWriter();
$myCal->setHTMLOutput();

$myCal->start();            // start output
$myCal->add($myEvent);      // add event after output started!
$myCal->add($myEvent2);     // you can add more than one event
$myCal->end();              // end output
```

That's it. You've added your first event. Have a look at the HTML output and you will see, the event-data is there.

Methods

All methods must be used on the iEvent-object you created first. As in the example under Basic usage or on the method*clear()*.

addAdditional(*text*)

Adds an additional property.

Use this to use features of RFC 2445 that are not implemented yet or are not implemented for reasons of easy using.

text must be a line of text giving one property that conformes to RFC 2445.

Each event can have more than one additional property.

Use only if you know, what you are doing!

```
addAdditional(“SOMEPROPERTY;PARAMETER:VALUE”);
```

addAlarm(alarmObject)

Adds an alarm to the event.

The alarm must be an object of one of the 4 classes:

- iCalAudioAlarm
- iCalDisplayAlarm
- iCalEmailAlarm
- iCalProcedureAlarm

See the description of these classes for more details. Never use the iCalAlarm class directly!

Each event can have multiple alarms of multiple types added.

```
$myAlarm1 = new iCalDisplayAlarm();
// set a few things
...
// and add the alarm
$myEvent->addAlarm($myAlarm1);

// another alarm
$myAlarm2 = new iCalAudioAlarm();
// set a few things
...
// and add the alarm
$myEvent->addAlarm($myAlarm2);
```

addAttachment(*attachment*)

Adds an attachment to event. Only the URL given is added but not a file or something else.

attachment must be a string and give the full URL to the attachment.

Please specify the schema when adding the attachment (i.e. http://). Keep in mind that someone else might not be able to read a file for access or security reasons.

Each event can have more than one attachment.

```
addAttachment(“http://www.sebastiankleine.de/attachment.xls”);
```

addAttendee(*attendee*)

Adds an attendee to the event. Each event can have multiple attendees added.

```
addAttendee(“Carl”);
```

addCategories(*cat*)

Adds categories to the event.

Each event can belong to multiple categories. You can either use this method several times or give a comma seperated list of the categories (or mix both ways).

```
addCategories(“Sport, Home”);
```

addComment(*text*)

Adds a comment to the event.

Each event can have multiple comments. Depending on the used program to view the iCalender-file the comments might be shown or not.

```
addComment(“Please find a different time for the meeting.”);
```

addContact(*name*, *alternate*)

Adds a contact information to the event.

Both values must be string. *alternate* is optional and can give the URL to a LDAP server entry or a vCard.

Each event can have multiple contacts.

```
addContact(“Sebastian Kleine”);
```

addRecurrence(recurrenceObject)

Adds a recurrence object (iCalRecurrence) to the event.

Each event can have multiple recurrence objects added. The combination of all will be used to compute all recurrences.

recurrenceObject must be of class iCalRecurrence.

```
$myRecurrence = new iCalRecurrence();
// set a few things
...
// and add the recurrence
$myEvent->addRecurrence($myRecurrence);
```

addRelated(*UID*)

Adds a related object. This could be a event or some other iCalender-object.

Each event can have multiple related objects.

Please give the UID of the related object.

```
addRelated(“myUniqueUID”);
```

addResource(*text*)

Adds a resource to the event.

Each event can have multiple resources. You can use this method several times or give a comma seperated list.

There is no action taken to take care, that the required resources are available!

```
addResource(“beamer, whiteboard”);
```

addxProp(*name*, *text*, *parameter*)

Adds a non standard property.

All values must be string. *name* gives the name of the property. A X- will be added at the beginning of the name to show a non standard property. *text* gives the text or values for the property. *parameter* is optional and gives additional parameters.

Each event can have multiple non standard properties (xProps).

```
addxProp(“ShowUntil”, “2008-05-31”);
```

clear()

Clears all used data. After using clear you can reuse the object for another event.

```
// first event
$myEvent->setStart(...);
$myEvent->setEnd(...);
$myEvent->setShortDescription(“event 1”);

// now add the event to the output
$myCal->add($myEvent);

// clear all data
$myEvent->clear();

// and reuse it
$myEvent->setStart(...);
$myEvent->setEnd(...);
```

```
$myEvent->setShortDescription("event 2");

// now add the second event to the output
$myCal->add($myEvent);
```

clearAdditional()

Removes all additional properties.

```
clearAdditional();
```

clearAlarm()

Removes all alarms.

```
clearAlarm();
```

clearAttachment()

Removes all attachments.

```
clearAttachment();
```

clearAttendee()

Removes all attendees.

```
clearAttendee();
```

clearCategories()

Removes all categories.

```
clearCategories();
```

clearComments()

Removes all comments.

```
clearComments();
```

clearContacts()

Removes all contacs.

```
clearContacts();
```

clearRecurrence()

Removes all recurrences from the event.

```
clearRecurrence();
```

clearRelated()

Removes all related objects.

```
clearRelated();
```

clearResources()

Removes all resources.

```
clearResources();
```

clearxProp()

Removes all non standard properties (xProps).

```
clearxProp();
```

getVersion()

Returns the version of the iCalEvent-class

```
echo($myEvent->getVersion());
```

removeRequestStatus()

Removes the request status completely.

```
removeRequestStatus();
```

setCategories(*cat*)

A synonym for *addCategories()*.

Please use *addCategories()* instead!

setClassification(*access*)

Sets access classifications.

There is no way to enforce or ensure access restrictions using this property.

```
setClassification(“PUBLIC”);
setClassification(“project leaders”);
```

setComment(*text*)

A synonym for *addComment()*.

Please use addComment*()* instead!

setContact(*text, alternate*)

A synonym for *addContact()*.

Please use addContact*()* instead!

setCreated(*year*, *month*, *day*, *hour*, *minute*, *second*)

Sets the time the event information was first created. This is not the time when the used iCalEvent-object is created (*setDtStamp*) but when the event was first created in another event-handling-system (it might be the same as *setDtStamp*).

All values are two-digits strings (except year as 4-digit string). They must give the time in UTC.

```
setCreated(“2008”, “01”, “22”, “18”, “30”, “36”);              // UTC-time!
```

setDtStamp(*year*, *month*, *day*, *hour*, *minute*, *second*)

Sets the DtStamp value. This is a date and time that the instance of the event-object was created.

Because this property must be set, it is set automatically when creating an iCalEvent-object and is reset when using *clear()*. There should be no need to set this manually.

If you still want to do this, then give ALL parameters as two-digit string (except year as 4-digit string). The values must be UTC-time!

```
setDtStamp(“2008”, “01”, “22”, “18”, “30”, “36”);              // UTC-time!
```

setDuration(*week*, *day*, *hour*, *minute*, *second*)

Sets the duration for the event.

All parameters are integer values.

There are two ways to define the end of a event. You could either take *setEnd()* or *setDuration()*. But you must not use both.

Please be aware, that some programs seem to ignore *setDuration()* and require *setEnd()*.

Examples:

```
setDuration(0, 0, 0, 30);          // sets duration to 30 minutes
setDuration(0, 0, 1);              // sets duration to 1 hour
setDuration(1, 1);                 // sets duration to 8 days (one week and one day)
setDuration(0, 0, 2, 15, 30);      // sets duration to 2 hours, 15 minutes and 30
                                   // seconds
```

setEnd(*year*, *month*, *day*, *isUTC*, *isLocalTime*, *localTimeZone*, *useTime*, *hour*, *minute*, *second*)

Sets the end for the event.

Parameters are the same as for *setStart()*. So please look there.

There are two ways to define the end of a event. You could either take *setEnd()* or *setDuration()*. But you must not use both.

Please be aware, that some programs seem to ignore *setDuration()* and require *setEnd()*.

If you want to set the end time for an event, that has just a date but no time, than you might need to add one day to the end of the event. Here's an example:

```
setStart(“2008”, “01”, “20”);
setEnd(“2008”, “01”, “22”);
```

This event will be shown on the 20th and 21st of January 2008. But NOT on the 22nd.

setGeo(lon, lat)

Sets geographic coordinates. Both values must be float.

```
setGeo(12.1532, 5.4582);           // no idea where this is
```

setLastModified(*year*, *month*, *day*, *hour*, *minute*, *second*)

Sets the time the event information was last modified. Could be useful if different versions of the event exist.

All values are two-digits strings (except year as 4-digit string). They must give the time in UTC.

```
setLastModified(“2008”, “01”, “23”, “12”, “23”, “08”);                    // UTC-time!
```

setLocation(*location*, *alternate*)

Sets the location for the event.

Both parameters are string values. *alternate* is optional. It could give the URI to either an LDAP server entry or a vCard.

```
setLocation(“Room 77”);
```

setLongDescription(*text*)

Sets a long description for the event.

```
setLongDescription(“This is the montly staff meeting. We will discuss the advantage of the
new time management system and other very important things. Or maybe we will just
drink a lot of coffee.”);
```

setNoTransparency()

Sets the event to be not transparent.

That means that during the event you do not have time to do other things. Your time is blocked . (During a meeting you are not able to drive someone to the airport.)

The opposite is *setTransparency()*.

```
setNoTransparency()
```

setOrganizer(*organizer*)

Sets organizer information. Value must be string.

```
setOrganizer(“Someone else”);
```

setPriority(*priority*)

Sets priority-information.

Values are integer ranging from 0 to 9. 9 is the highest priority.

```
setPriority(6);
```

setRequestStatusClientError(*description*, *substatus*)

This sets the status of a request as client error. There has been a syntax or semantic error.

description could give a more detailed status description as a string. *substatus* is a optional integer value that could give a more detailed status code. Default value is 0.

The main status code for client error is 3.

```
setRequestStatusClientError(“Could not read event details”);
```

setRequestStatusPending(*description*, *substatus*)

This sets the status of a request as pending (request has been received and initially processed but completion is pending).

description could give a more detailed status description as a string. *substatus* is a optional integer value that could give a more detailed status code. Default value is 0.

The main status code for pending is 1.

```
setRequestStatusPending(“Waiting for clearance from boss”);
```

setRequestStatusSchedulingError(*description*, *substatus*)

This sets the status of a request as scheduling error.

description could give a more detailed status description as a string. *substatus* is a optional integer value that could give a more detailed status code. Default value is 0.

The main status code for scheduling error is 4.

```
setRequestStatusSchedulingError(“The event is cancelled.”);
```

setRequestStatusSuccessful(*description*, *substatus*)

This sets the status of a request as successful.

description could give a more detailed status description as a string. *substatus* is a optional integer value that could give a more detailed status code. Default value is 0.

The main status code for successful is 2.

```
setRequestStatusSuccessful(“We will participate”, 1);
```

setSequence(*revision*)

Sets a revision number.

revision is a integer number starting from 0. You are responsible for incrementing the revision number. This is not done automatically.

A higher revision number means this event has changed and the higher the number the more up-to-date the information is.

```
setSequence(18);                // would be revision number 19
```

setShortDescription(*text*)

Sets a short description for the event.

```
setShortDescription(“Staff meeting”);
```

setStart(*year*, *month*, *day*, *isUTC*, *isLocalTime*, *localTimeZone*, *useTime*, *hour*, *minute*, *second*)

Sets the start of the event.

You must always give *year*, *month* and *day*. The other parameters are optional, but if you give them, you must give them in order!

year, *month*, *day*, *hour*, *minute* and *second* must be given as strings with a length of two (except *year* with 4 digits). *localTimeZone* must be string, too.

isUTC, *isLocalTime* and *useTime* could be 0/false or 1/true.

You could set only the date and no time information:

```
setStart(“2008”, “01”, “20”);
```

This means an event that happens on a specific day but not at a specific time.

If you want to give time information, you have two possibilities. You can give them in local time or in UTC-time (GMT). If you want to have UTC-time then set *isUTC* to 1 and *isLocalTime* to 1. Also *useTime* must be 1.

```
setStart("2008", "01", "20", 1, 0, "", 1, "14", "12");
```

This is a event starting at 20/01/2008 at 14:12:00 UTC-time. To get UTC-time you could use the *gmdate()* function of PHP.

```
setStart(gmdate("Y"), gmdate("m"), gmdate("d"), 1, 0, "", 1, "gmdate("H"), _
          gmdate("i"), gmdate("s"));
```

This would set the start to present date and time in UTC (GMT).

If you don't want to use UTC time but rather want local time then set *isUTC* to 0 and *isLocalTime* and *useTime* to 1. You can specify which time zone you mean using the *localTimeZone* parameter. To figure out how a specific time zone is called you can look here: http://twiki.org/cgi-bin/xtra/tzdatepick.html. Don't use *gmdate()* in this case but use *date()* for local time.

```
setStart(date("Y"), date("m"), date("d"), 0, 1, "", 1, "date("H"), date("i"), date("s"));
```

If *useTime* is 0 then no time information is added to the event. Even if it is given.

If you are sure, that all who will look at this calender have the same time zone you have, then it is save to use local time and give no time zone. If you are not sure than either give the information in UTC time or specifiy your time zone.

Some more examples:

```
setStart("2008", "01", "20", 0, 1, "Europe/Berlin", 1, "20", "15");
```

Sets the start to 20/01/2008 at 20:15:00 local time in Berlin, Germany.

```
setStart("2008", "01", "20", 0, 1, "", 0, "20", "15", "30";
```

Sets start to 20/01/2008. Time information is dropped.

setStatus(*status*)

Sets the status for the event. *status* could be either none , tentative , confirmed or cancelled .

None: no status set

Tentative: event is not confirmed

Confirmed: should be clear

Cancelled: should be clear

If none of the above is used as status then it will be set to none.

```
setStatus(“tentative”);                    // sets status to tentative
setStatus(“none”);                         // sets status to none
setStatus(“myStatus”);                     // sets status to none!
```

setTransparency()

Sets the event to be transparent.

This means that you could to other things during this event. (While waiting for some delivery you might be able to do something else. Your time is not blocked .)

```
setTransparency();
```

setUID(*text*)

Sets the unique UID-value.

This value is a string and must be globally unique!

It is required in every event-object. Because of this it is automatically set when creating the object and reset when using *clear()*. The default value consists of the actual timestamp + a 4-digit random value + @ + the server name as given by $_SERVER['SERVER_NAME'] .

If you want to set this on your own, you are responsible for creating a globally unique UID!

```
setUID(“thisIsMyGloballyUniqueUID”);
```

setURL(*url*)

Sets a URL that is somehow associated with the event (it could give more information). Please include the scheme of the URL (i.e. http://).

```
setURL(“http://www.sebastiankleine.de”);
```

iCalAlarm

This is the basic class for alarms. Never use this class. Instead use one of its children: iCalAudioAlarm, iCalDisplayAlarm, iCalEmailAlarm or iCalProcedureAlarm.

Basic usage

Never use directly. Only use the children: iCalAudioAlarm, iCalDisplayAlarm, iCalEmailAlarm or iCalProcedureAlarm.

Methods

These methods might or must be called when using an iCalAlarm child.

addAdditional(*text*)

Adds an additional property.

Use this to use features of RFC 2445 that are not implemented yet or are not implemented for reasons of easy using.

text must be a line of text giving one property that conformes to RFC 2445.

Each event can have more than one additional property.

Use only if you know, what you are doing!

```
addAdditional(“SOMEPROPERTY;PARAMETER:VALUE”);
```

addxProp(*name*, *text*, *parameter*)

Adds a non standard property.

All values must be string. *name* gives the name of the property. A X- will be added at the beginning of the name to show a non standard property. *text* gives the text or values for the property. *parameter* is optional and gives additional parameters.

Each event can have multiple non standard properties (xProps).

```
addxProp(“Intensity”, “10”);
```

clear()

Clears all used data. After using clear you can reuse the object for another alarm (but only of the same type an display alarm cannot become an audio alarm).

```
clear();
```

clearAdditional()

Removes all additional properties.

```
clearAdditional();
```

clearxProp()

Removes all non standard properties (xProps).

```
clearxProp();
```

getVersion()

Returns the version of the iCalAlarm-child-class

```
echo($myDisplayAlarm->getVersion());
```

setAlarmAfterEnd(*day*, *hour*, *minute*)

Sets the alarm to start a certain time after the end of the corresponding object (iCalEvent). A end time or duration must be given in the corresponding object.

If you use this, than do not use *setAlarmDateTime()*, *setAlarmBeforeStart()*, *setAlarmBeforeEnd()* or *setAlarmAfterStart()*!

All parameters are integer values. Use only one and set the other two to 0.

```
setAlarmAfterEnd(0, 0, 30);         // alarm starts 30 minutes after event ends
setAlarmAfterEnd(0, 2, 0);          // alarms start 2 hours after events ends
setAlarmAfterEnd(1, 0, 0);          // alarms starts 1 day after events ends
```

setAlarmAfterStart(*day*, *hour*, *minute*)

Sets the alarm to start a certain time after the start of the corresponding object (iCalEvent). A start time must be given in the corresponding object.

If you use this, than do not use *setAlarmDateTime()*, *setAlarmBeforeStart()*, *setAlarmBeforeEnd()* or *setAlarmAfterEnd()*!

All parameters are integer values. Use only one and set the other two to 0.

```
setAlarmAfterStart(0, 0, 30);       // alarm starts 30 minutes after event starts
setAlarmAfterStart(0, 2, 0);        // alarms start 2 hours after events starts
setAlarmAfterStart(1, 0, 0);        // alarms starts 1 day after events starts
```

setAlarmBeforeEnd(*day*, *hour*, *minute*)

Sets the alarm to start a certain time before the end of the corresponding object (iCalEvent). A end time or duration must be given in the corresponding object.

If you use this, than do not use *setAlarmDateTime()*, *setAlarmAfterStart()*, *setAlarmAfterStart()* or *setAlarmAfterEnd()*!

All parameters are integer values. Use only one and set the other two to 0.

```
setAlarmBeforeEnd(0, 0, 30);        // alarm starts 30 minutes before event ends
setAlarmBeforeEnd(0, 2, 0);         // alarms start 2 hours before events ends
setAlarmBeforeEnd(1, 0, 0);         // alarms starts 1 day before events ends
```

setAlarmBeforeStart(*day*, *hour*, *minute*)

Sets the alarm to start a certain time before the start of the corresponding object (iCalEvent). A start time must be given in the corresponding object.

If you use this, than do not use *setAlarmDateTime()*, *setAlarmAfterStart()*, *setAlarmBeforeEnd()* or *setAlarmAfterEnd()*!

All parameters are integer values. Use only one and set the other two to 0.

```
setAlarmBeforeStart(0, 0, 30);      // alarm starts 30 minutes before event starts
setAlarmBeforeStart(0, 2, 0);       // alarms start 2 hours before events starts
setAlarmBeforeStart(1, 0, 0);       // alarms starts 1 day before events starts
```

setAlarmDateTime(*year*, *month*, *day*, *hour*, *minute*, *second*, isUTC)

Sets a specific date and time for the alarm to start.

If you use this, than do not use *setAlarmBeforeStart()*, *setAlarmAfterStart()*, *setAlarmBeforeEnd()* or *setAlarmAfterEnd()*!

month, *day*, *hour*, *minute* and *second* must be given as two-digit strings. *year* is a 4-digit string. *isUTC* should be 1 when using UTC-time or 0 if not.

The following will start the alarm at 30/01/2008 at 10:00:00 (am) local time.

```
setAlarmDateTime("2008", "01", "30", "10", "00", "00", 0);
```

setAlarmRepeat(*number*, *day*, *hour*, *minute*)

Set the alarm to repeat *number* times after the initial trigger.

number must be a integer value greater than 0. day, hour and minute must be integer values. They give the interval after which the alarms triggers again. Use only one of the three and set the other two to 0.

```
setAlarmRepeat(3, 0, 1, 0);   // alarm repeats 3 times after one hour delay
```

iCalAudioAlarm

This is one of the four alarm classes. When the alarm triggers, an audio file should be played. There is no guarantee, that the audio file will be played!

Basic usage

Create a iCalAudioAlarm object:

```
$myAudioAlarm = new iCalAudioAlarm();
```

Set all required properties. You must set either *setAlarmDateTime()* or one of the *setAlarmBefore/After-* methods.

All other things are optional. *setAudioFile()* and *setAlarmRepeat()* must not occur more than once. add*xProp()* and add*Additional()* can occure multiple times.

```
$myAudioAlarm->setAlarmBeforeStart(0, 3, 0);
$myAudioAlarm->setAudioFile("http://www.sebastiankleine.de/alarm.mp3");
$myAudioAlarm->setRepeat(2, 0, 0, 30);
```

Add the alarm to the event:

```
$myEvent->addAlarm($myAudioAlarm);
```

Methods

All iCalAlarm methods can be used.

setAudioFile(*file*)

Gives the URL to a sound file that should be played, if the alarm triggers.

There is no guarantee, that this file can be played.

```
setAudioFile("http://www.sebastiankleine.de/alarm.mp3");
```

iCalDisplayAlarm

This is one of the four alarm classes. When the alarm triggers, an text message should be displayed.

Basic usage

Create a iCalDisplayAlarm object:

```
$myDisplAlarm = new iCalDisplayAlarm();
```

Set all required properties. You must set either *setAlarmDateTime()* or one of the *setAlarmBefore/After-* methods and *setText()*.

All other things are optional. *setAlarmRepeat()* must not occur more than once. add*xProp()* and add*Additional()* can occure multiple times.

```
$myDisplAlarm->setAlarmBeforeStart(0, 3, 0);
$myDisplAlarm->setText(“Important meeting”);
$myDisplAlarm->setRepeat(2, 0, 0, 30);
```

Add the alarm to the event:

```
$myEvent->addAlarm($myDisplAlarm);
```

Methods

All iCalAlarm methods can be used.

setText(*message*)

Sets the message that should be displayed, if the alarm triggers.

```
setText(“Important meeting”);
```

iCalEmailAlarm

This is one of the four alarm classes. When the alarm triggers, an email message should be send to the recipents

Basic usage

Create a iCalEmailAlarm object:

```
$myEmailAlarm = new iCalEmailAlarm();
```

Set all required properties. You must set either *setAlarmDateTime()* or one of the *setAlarmBefore/After-* methods and *setText()*. You must at least set one recipient with *addEmailAddress()* or no email could be send.

All other things are optional. *setAlarmRepeat()* must not occur more than once. *addAttachment()*, add*xProp()* and add*Additional()* can occure multiple times.

```
$myEmailAlarm->setAlarmBeforeStart(0, 3, 0);
$myEmailAlarm->setText("Important meeting with Mr Jones", "important meeting");
$myEmailAlarm->addEmailAddress("noreply@sebastiankleine.de");
$myEmailAlarm->setRepeat(2, 0, 0, 30);
```

Add the alarm to the event:

```
$myEvent->addAlarm($myEmailAlarm);
```

Methods

All iCalAlarm methods can be used.

addAttachment(file)

Gives the URL of a file, that should be send with the email.

There is no guarantee, that the file will be send as an attachment.

Each email alarm could have multiple attachments.

```
addAttachment("http://www.sebastiankleine.de/somefile.pdf");
```

addEmailAddress(email)

Adds a recipient. Each email alarm must have at least one recipient but could have more than one.

Must give the email-address.

```
addEmailAddress("noreply@sebastiankleine.de");
```

clearAttachments()

Removes all attachments

```
clearAttachments();
```

clearEmailAddresses()

Removes all email addresses from the list

```
clearEmailAddresses();
```

setText(*message, subject*)

Sets the message and subject for the email.

```
setText(“some text for the body of the email”, “Important meeting”);
```

iCalProcedureAlarm

This is one of the four alarm classes. When the alarm triggers, an procedure or program should be executed.

Running external programs might not work because of access restrictions or security issues.

Basic usage

Create a iCalProcedureAlarm object:

```
$myProcedureAlarm = new iCalProcedureAlarm();
```

Set all required properties. You must set either *setAlarmDateTime()* or one of the *setAlarmBefore/After-* methods and *setProcedure().*

All other things are optional. *setAlarmRepeat()* must not occur more than once. add*xProp()* and add*Additional()* can occure multiple times.

```
$myProcedureAlarm->setAlarmBeforeStart(0, 3, 0);
$myProcedureAlarm->setProcedure("http://www.sebastiankleine.de/go.php");
$myProcedureAlarm->setRepeat(2, 0, 0, 30);
```

Add the alarm to the event:

```
$myEvent->addAlarm($myProcedureAlarm);
```

Methods

All iCalAlarm methods can be used.

setProcedure(*procedure, paramters*)

procedure is the procedure or file that should be executed. *parameters* is optional and gives parameters to pass to *procedure*.

```
setProcedure("alarm.exe", "1");
```

iCalRecurrence

Used to set recurrences in events and other iCalWriter components. (At the moment only events are available.)

Recurrence rules are quite complicated in the iCalender-standard. This class tries to order them into a simple and usable way. Still you can do very difficult things with recurrences. Please do not think this is a completely easy topic. It might need some work to fully understand and use its potential.

Recurrences are computed using the start and end times of the corresponding object (or start and duration).

Basic usage

Create an iCalRecurrence object:

```
$myRecurrence = new iCalRecurrence();
```

Set the properties you need. There are no required properties.

The following is an example of repeating something each day for 3 days:

```
$myRecurrence->setRecurrenceFrequenceDaily();
$myRecurrence->setRecurrenceCount(3);
```

And then add the recurrence to the event (or another iCalwriter component, that can handle recurrences):

```
$myEvent->addRecurrence($myRecurrence);
```

Methods

There are 4 big groups of methods:

- Recurrence Dates/Times
- Exception Dates/Times
- Recurrence Rules
- Exception Rules

The recurrences used will be computed using all given information. Keep in mind, that you can use multiple recurrence objects to define recurrence more in detail. None of the above group is required. You could have a recurrence object that just defines exception dates and another recurrence object that define one exception rule.

Common methods

These are methods that are independent of the 4 groups.

clear()

Use this to clear all settings and reuse the iCalRecurrence object.

```
$myRecurrence->clear();
```

getVersion()

Returns the version of the iCalRecurrence class.

```
$myRecurrence->getVersion();
```

Recurrence Dates/Times

Use this to define dates and times for recurrence.

Hint: Use this, if you have a small number of recurrences and there is no pattern in these recurrences.

addDate(year, month, day)

Adds a single date to the recurrence.

All values must be given as string. *Month* and *day* must be 2 digit strings and *year* a 4 digit string. You can add multiple dates to a recurrence.

Recurrence dates should not be earlier than the corresponding object (i.e. the start of an event).

```
$myRecurrence->addDate("2008", "04", "25");      // adds a recurrence at April, 25th 2008
```

addDateTime(year, month, day, hour, minute, second, isUTC)

Adds a single date and time to the recurrence.

All values must be given as string (except isUTC). *Month*, *day*, *hour*, *minute* and *second* must be 2 digit strings and *year* a 4 digit string. Set *isUTC* to 0 if using local time or to 1 if using UTC-time.

You can add multiple date-times to a recurrence.

Recurrence dates should not be earlier than the corresponding object (i.e. the start of an event).

```
$myRecurrence->addDateTime("2008", "04", "25", "14", "00", "00", 0);  // adds a
                                        // recurrence at April, 25th 2008 at 14:00:00 local time
```

addPeriod(startyear, startmonth, startday, starthour, startminute, startsecond, endyear, endmonth, endday, endhour, endminute, endsecond, isUTC)

Adds a period to the recurrence by giving start and end date-times.

All values must be given as string (except isUTC). *Month*, *day*, *hour*, *minute* and *second* must be 2 digit strings and *year* a 4 digit string. Set *isUTC* to 0 if using local time or to 1 if using UTC-time.

You can add multiple periods and you can mix *addPeriod* and *addPeriodDuration*.

The start of the period must be before the end of the period but should be after the start of the corresponding object (i.e. the start of the event).

```
$myRecurrence->addPeriod(“2008”, “04”, “25”, “14”, “00”, “00”, “2008”, “05”, “25”, _
“14”, “00”, “00”, 0);                    // Adds a period starting at April, 25th 2008 at
                                         // 14:00:00 and ending one month later
                                         // local time
```

addPeriodDuration(startyear, startmonth, startday, starthour, startminute, startsecond, weeks, days, hours, minutes, isUTC)

startmonth, start*day*, start*hour*, start*minute* and start*second* must be 2 digit strings and start*year* a 4 digit string. Set *isUTC* to 0 if using local time or to 1 if using UTC-time. *weeks*, *days*, *hours*, and *minutes* gives the duration for the period as integer values.

You can add multiple periods and you can mix *addPeriod* and *addPeriodDuration*.

The start of the period must be before the end of the period but should be after the start of the corresponding object (i.e. the start of the event).

```
$myRecurrence->addPeriodDuration(“2008”, “04”, “25”, “14”, “00”, “00”, 2, 1, 0, 0, 0);
                                   // adds a period starting at April, 25th 2008 at
                                   // 14:00:00 and ending 15 days later
                                   // (2 weeks and 1 day) local time
```

clearDates()

Removes all set recurrence dates.

```
$myRecurrence->clearDates();
```

clearDateTimes()

Removes all set recurrence date-times.

```
$myRecurrence->clearDateTimes();
```

clearPeriods()

Removes alle recurrence periods.

$myRecurrence->clearPeriods();

Exception Dates/Times

Use this to define dates and times for exceptions.

Hint: Use this, if you have a small number of exceptions and there is no pattern in these exceptions.

The following methods available here. They work the same way as the *recurrence date/times methods* (these do not have the word exception in it). And of course they exclude the given information.

- addExceptionDate()
- addExceptionDateTime()
- clearExceptionDates()
- clearExceptionDateTimes()

There are no exception periods.

You can exclude the initial start date or time (of the corresponding object i.e. the event start). Although behaviour in this case is not completely clear.

Recurrence rules

Use these methods to add recurrences by giving rules.

Hint: Use this if there is a pattern, that could be applied to the recurrences.

If you want to use recurrence rules, you have to set a frequence. This could be daily, weekly, monthly, yearly, hourly and minutely. You can set only one frequence. If you want to give recurrences with different frequencies you have to use multiple iCalRecurrence objects.

If you want to give multiple recurrence rules you have to use multiple iCalRecurrence objects.

Depending on the frequence you set, other options might make sense or not. If you do not set a recurrence frequence, no recurrence rule will be generated.

setRecurrenceFrequenceDaily()

Sets a daily frequence.

Do not set multiple recurrence frequencies!

```
$myRecurrence->setRecurrenceFrequenceDaily();
```

setRecurrenceFrequenceHourly()

Sets a hourly frequence.

Do not set multiple recurrence frequencies!

```
$myRecurrence->setRecurrenceFrequenceHourly();
```

setRecurrenceFrequenceMinutely()

Sets a minutely frequence.

Do not set multiple recurrence frequencies!

```
$myRecurrence->setRecurrenceFrequenceMinutely();
```

setRecurrenceFrequenceMonthly()

Sets a monthly frequence.

Do not set multiple recurrence frequencies!

```
$myRecurrence->setRecurrenceFrequenceMonthly();
```

setRecurrenceFrequenceWeekly()

Sets a weekly frequence.

Do not set multiple recurrence frequencies!

```
$myRecurrence->setRecurrenceFrequenceWeekly();
```

setRecurrenceFrequenceYearly()

Sets a yearly frequence.

Do not set multiple recurrence frequencies!

```
$myRecurrence->setRecurrenceFrequenceYearly();
```

addRecurrenceByDay(day, specific)

Adds a recurrence for a certain day.

day must have one of the following values (as string): MO, TU, WE, TH, FR, SA, SU. specific is optional and means the nth occurance of the day (i.e. in a week, month or year). specific can also be negative, which means starting from the end of the month

Each recurrence rule can have multiple by day values.

```
$myRecurrence->addRecurrenceByDay("MO");          // each monday
$myRecurrence->addRecurrenceByDay("FR");          // and each friday
$myRecurrence->addRecurrenceByDay("TU", 3);       // and the third tuesday
$myRecurrence->addRecurrenceByDay("SA", -1);      // and the last saturday
```

addRecurrenceByHour(number)

Adds a recurrence for the hour *number*.

number must be between 0 and 23.

You can add multiple hours by calling this method multiple times.

```
$myRecurrence->addRecurrenceByHour(4);
$myRecurrence->addRecurrenceByHour(5);
                         // adds recurrences in hours 4 and 5 (am)
```

addRecurrenceByMinute(number)

Adds a recurrence for the minute *number*.

number must be between 0 and 59.

You can add multiple minutes by calling this method multiple times.

```
$myRecurrence->addRecurrenceByMinute(29);
$myRecurrence->addRecurrenceByMinute(43);
                         // adds recurrences in minutes 29 and 43
```

addRecurrenceByMonth(number)

Adds a recurrence for the month *number*.

1 = january, 2 = february, ...

You can add multiple months by calling this method multiple times.

```
$myRecurrence->addRecurrenceByMonth(4);
$myRecurrence->addRecurrenceByMonth(5);
                         // adds recurrences in month April and May
```

addRecurrenceByMonthDay(day)

Adds a recurrence for the *day* every month

day must be between 1 and 31 or -1 and -31. Negative values mean starting from the end of the month.

You can add multiple days by calling this method multiple times.

```
$myRecurrence->addRecurrenceByMonthDay(10);         // every 10th of the month
$myRecurrence->addRecurrenceByMonthDay(-1);         // every last day of the month
```

addRecurrenceBySetPos(number)

Selects only the *number*th occurrence of another addRecurrenceByXXX method.

number must be a integer value.

You can call this method multiple times.

```
$myRecurrence->setRecurrenceFrequenceMonthly();
$myRecurrence->addRecurrenceByDay(“MO”);
$myRecurrence->addRecurenceByDay("TU");
$myRecurrence->addRecurenceByDay("WE");
$myRecurrence->addRecurenceByDay("TH");
$myRecurrence->addRecurenceByDay("FR");
$myRecurrence->addRecurrenceBySetPos(-1);
                                   // selects only the last workday of the month
```

addRecurrenceByWeek(weekNumber)

Adds a recurrence every certain week.

weekNumber must give the number of the week. Keep in mind, on which day the weeks starts (*setWkSt()*)

You can add multiple days by calling this method multiple times.

```
$myRecurrence->addRecurrenceByWeek(4);              // add the 4th week
```

addRecurrenceByYearDay(day)

Adds a recurrence for the *day* every year

day must be between 1 and 366 or -1 and -366. Negative values mean starting from the end of the year.

You can add multiple days by calling this method multiple times.

```
$myRecurrence->addRecurrenceByYearDay(100);         // every 100th of the year
$myRecurrence->addRecurrenceByYearDay(-1);          // every last day of the year
```

addRecurrenceUntilDateTime(year, month, day, hour, minute, second, isUTC)

Adds a recurrence end date and time

All values must be given as string (except isUTC). *Month*, *day*, *hour*, *minute* and *second* must be 2 digit strings and *year* a 4 digit string. Set *isUTC* to 0 if using local time or to 1 if using UTC-time.

```
$myRecurrence->addRecurrenceUntilDateTime(“2008”, “04”, “25”, “14”, “00”, “00”, 0);
                    // Reccurences will repeat until April, 25th 2008 at 14:00:00 local time
```

clearRecurrenceByDay()

Removes all recurrence by day values.

```
$myRecurrence->clearRecurrenceByDay();
```

clearRecurrenceByHour()

Removes all recurrence by hour values.

```
$myRecurrence->clearRecurrenceByHour();
```

clearRecurrenceByMinute()

Removes all recurrence by minute values.

```
$myRecurrence->clearRecurrenceByMinute();
```

clearRecurrenceByMonth()

Removes all recurrence by month values.

```
$myRecurrence->clearRecurrenceByMonth();
```

clearRecurrenceByMonthDay()

Removes all recurrence by month-day values.

```
$myRecurrence->clearRecurrenceByMonthDay();
```

clearRecurrenceBySetPos()

Removes all recurrence by setpos values.

```
$myRecurrence->clearRecurrenceBySetPos();
```

clearRecurrenceByWeek()

Removes all recurrence by week values.

```
$myRecurrence->clearRecurrenceByWeek();
```

clearRecurrenceByYearDay()

Removes all recurrence by year-day values.

```
$myRecurrence->clearRecurrenceByYearDay();
```

setRecurrenceCount(number)

How many recurrences should be generated.

number gives the number of generated recurrences after the initial start.

Set this to 0 to clear it.

If you do not set this or set it to 0 recurrences might never end (depending on *addRecurrenceUntilDateTime()*).

```
$myRecurrence->setRecurrenceCount(5);          // 5 recurrences
```

setRecurrenceInterval(number)

The recurrence will happen every *number* days, weeks, months, ...

Number must be a integer value greater than 0. If it is 0 the interval is cleared.

If you do not set this the default interval is 1.

```
$myRecurrence->setRecurrenceInterval(2);       // every other day
```

setWeekStartDay(day)

Sets the day a week will begin. Default is monday.

day must have one of the following values (as string): MO, TU, WE, TH, FR, SA, SU.

Leave it empty to clear the week start.

```
$myRecurrence->setWeekStartDay(“SU”);          // sets week start to sunday
```

Exception rules

Use this rules to add exceptions to recurrences.

Hint: Use this if there is a pattern to exceptions.

Everything works exactly as for *recurrence rules*. All methods are the same as for the *recurrence rules*. Instead of *Recurrence* they have *Exception* in their name.

Examples

Lets give some more examples.

Daily for 10 days

```
$myRecurrence->setRecurrenceFrequenceDaily();
$myRecurrence->setRecurrenceCount(10);
```

Everyday in January for 3 years

```
$myRecurrence->setRecurrenceFrequenceYearly();
$myRecurrence->setRecurrenceUntilDateTime(“2011”, “01”, “31”, “10”, “00”, “00”);
$myRecurrence->setRecurrenceByMonth(1);
$myRecurrence->setRecurrenceByDay(“MO”);
$myRecurrence->setRecurrenceByDay(“TU”);
$myRecurrence->setRecurrenceByDay(“WE”);
$myRecurrence->setRecurrenceByDay(“TH”);
$myRecurrence->setRecurrenceByDay(“FR”);
$myRecurrence->setRecurrenceByDay(“SA”);
$myRecurrence->setRecurrenceByDay(“SU”);
```

or

```
$myRecurrence->setRecurrenceFrequenceDaily();
$myRecurrence->setRecurrenceUntilDateTime(“2011”, “01”, “31”, “10”, “00”, “00”);
$myRecurrence->setRecurrenceByMonth(1);
```

Weekly on Tuesday and Thursday for 5 weeks, except the 3rd Tuesday

```
$myRecurrence->setRecurrenceFrequenceWeekly();
$myRecurrence->setRecurrenceCount(10);
$myRecurrence->setRecurrenceByDay(“TU”);
$myRecurrence->setRecurrenceByDay(“TH”);
$myRecurrence->setExceptionFrequenceWeekly();  // frequencies could be different for
                                               // recurrences and exceptions
$myRecurrence->setExceptionByDay(“TU”);
$myRecurrence->setExceptionBySetPos(3);
```

Monthly on the first Friday 5 times

```
$myRecurrence->setRecurrenceFrequenceWeekly();
$myRecurrence->setRecurrenceCount(5);
$myRecurrence->setRecurrenceByDay(“FR”, 1);
```

Every other month on the first and last Sunday of the month for 10 months

```
$myRecurrence->setRecurrenceFrequenceMonthly();
$myRecurrence->setRecurrenceCount(20);
$myRecurrence->setRecurrenceInterval(2);
$myRecurrence->setRecurrenceByDay(“SU”, 1);
$myRecurrence->setRecurrenceByDay(“SU”, -1);
```

Every Friday for ever but not on the 3 given dates

```
$myRecurrence->setRecurrenceFrequenceWeekly();
$myRecurrence->setRecurrenceByDay(“FR”);
$myRecurrence->addExceptionDate(“2008”, “05”, “09”, 0);
$myRecurrence->addExceptionDate(“2008”, “05”, “23”, 0);
$myRecurrence->addExceptionDate(“2008”, “06”, “27”, 0);
```

Every Monday and every 10th of every month

Attention: This are two different frequencies. You need to iCalRecurrence-objects to do this!

```
$myRecurrence->setRecurrenceFrequenceWeekly();
$myRecurrence->setRecurrenceByDay(“MO”);
$myRecurrence2->setRecurrenceFrequenceMonthly();
$myRecurrence2->setRecurrenceByMonthDay(10);
```