
Table of Contents

Creating a general dialog in R	1
Validators	5
.....	5

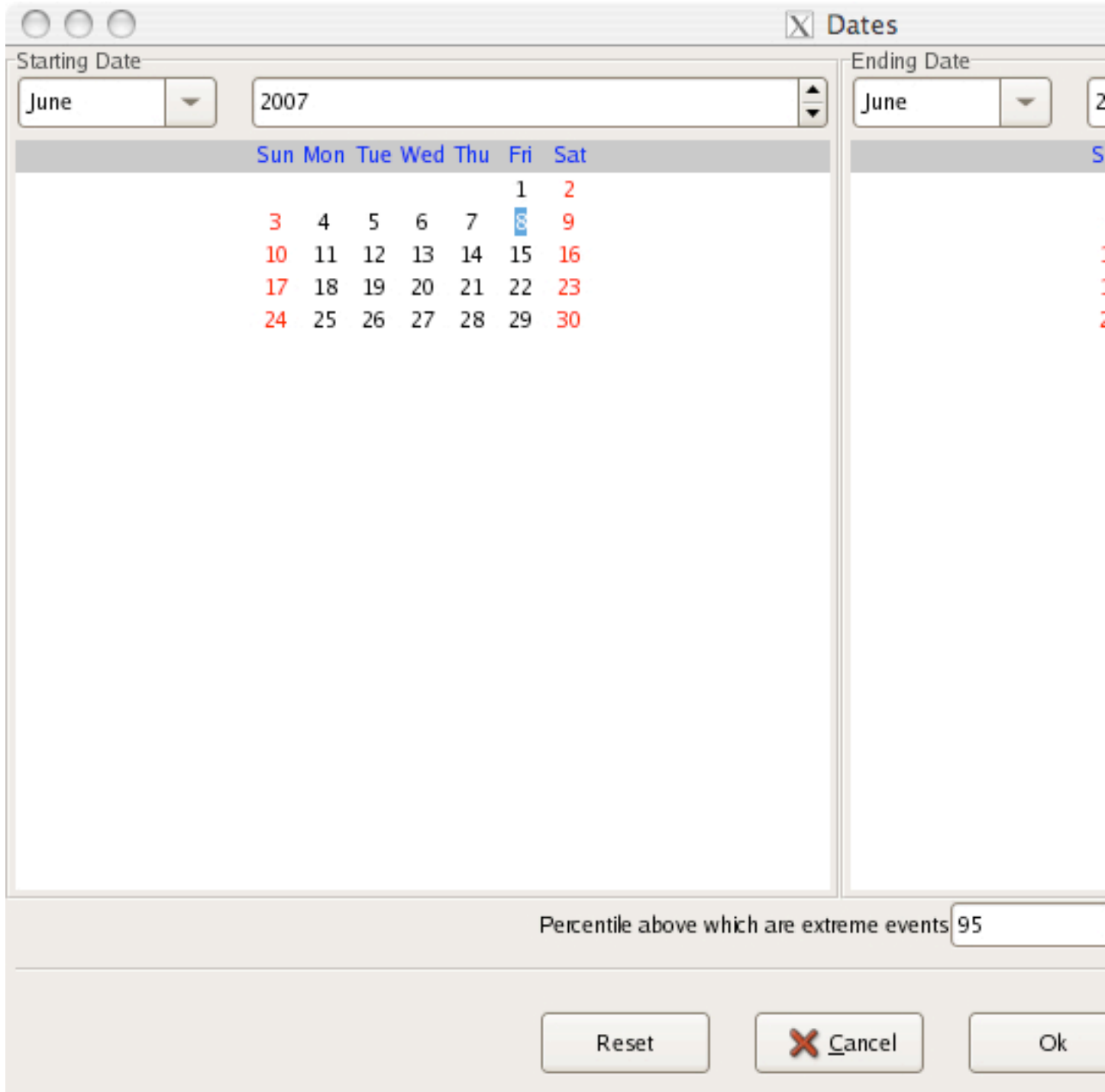
Note

To run this example, use the *xmlSource* function available by doing a regular *source* on the URL `xmlInternalSource.R` [<http://www.omegahat.org/xmlInternalSource.R>]. Then,

```
xmlSource("http://www.omegahat.org/RwxWidgets/ExampleDocuments/RwxDialog.xml")
```

Creating a general dialog in R

In this article, we create a dialog in R using the general `wxDialog` C++ class in `wxWidgets`. This follows from chapter 9 in the ???. We will create a relatively simple dialog that allows the user to specify the start and end dates of interest for a particular collection of data and to specify a value for the percentile of interest that defines an extreme event. We use a `wxCalendarCtrl` widget for each of the start and end dates and `xSpinCtrl` for specifying the percentile of interest. The resulting dialog looks like



We start by writing a function to create the dialog display

```
ID_RESET = wxID_ANY
```

```
dlg = RwxDialog(NULL, wxID_ANY, "Dates", size = as.integer(c(200, 100)))  
mainSizer = wxBoxSizer(wxVERTICAL)
```

```

start = end = wxDefaultDateTime()
a = wxCalendarCtrl(dlg, date = start)
b = wxCalendarCtrl(dlg, date = end)

sz = wxBoxSizer(wxHORIZONTAL)
tmp = wxStaticBoxSizer(wxVERTICAL, dlg, "Starting Date")
tmp$Add(a)
sz$Add(tmp, 1, c(wxEXPAND, wxALL), 10)

tmp = wxStaticBoxSizer(wxVERTICAL, dlg, "Ending Date")
tmp$Add(b)
sz$Add(tmp, 1, c(wxEXPAND, wxALL), 10)

mainSizer$Add(sz, 1, wxEXPAND)

tmp = wxBoxSizer(wxHORIZONTAL)
tmp$Add(wxStaticText(dlg, wxID_ANY, "Percentile above which are extreme events"),
spin = wxSpinCtrl(dlg, min = 0, max = 100, initial = 95)
tmp$Add(spin, 1, wxEXPAND, 5)
mainSizer$Add(tmp, 0, c( wxALIGN_CENTER, wxALL), 10)

line = wxStaticLine(dlg, wxID_STATIC, style = wxLI_HORIZONTAL, size = as.integer(c
mainSizer$Add(line, 0, c(wxGROW, wxALL, wxALIGN_CENTER), 5)

btnSizer = wxBoxSizer(wxHORIZONTAL)
btnIds = c("Reset" = ID_RESET, "Cancel" = wxID_CANCEL, "Ok" = wxID_OK)
sapply(names(btnIds),
      function(id) {
        b = wxButton(dlg, btnIds[id], id)
        btnSizer$Add(b, 0, c(wxGROW, wxALIGN_CENTER, wxALL), 10)
        b
      })

mainSizer$Add(btnSizer, 0, wxALIGN_CENTER)

dlg$SetSizer(mainSizer)
mainSizer$SetSizeHints(dlg)

```

1

Note

This needs to be tidied up.

¹The wxCAL_SEQUENTIAL_MONTH_SELECTION style causes the month buttons not to work.

Now that we have the basic Dialog displayed we need to be able to get the data from it. Since we are constructed it, we have access to the different widgets and can query their values. So we can have a corresponding R function that fetches the values given the dialog and the associated widgets.

```
dlg$ShowModal()

p = spin$GetValue()
start = as(a$GetDate(), "POSIXct")
end = as(b$GetDate(), "POSIXct")
print(list(p, start, end))
```

So at this point, we have all that we want and we can move on. When we are using a modal approach, these other variables such as **spin** and **a** and **b** will be within the scope of the function call when we return from ShowModal. However, if we are using a non-modal approach, we have to arrange to have a callback function that would have access to these. This is typically not a big problem as we define the callback within the function that creates the dialog and so has access to these variables. But there are potentially useful things we can do to simplify this.

One idea is to specify identifiers for the work components - **a**, **b** and **spin**. Then, given the dialog, we can get access to these values using the FindWindow method of all wxWindow objects. For example, we could define symbolic identifiers such as **ID_START**, **ID_END** and **ID_PERCENTILE** and use these when creating the widgets, e.g.

```
ID_START = 200
ID_END = 201
ID_PERCENTILE = 202

wxCalendarCtrl(dlg, ID_START, start)
wxCalendarCtrl(dlg, ID_END, end)
```

Then, we can access these with

```
tmp = dlg$FindWindow(ID_START)
as(tmp$GetDate(), "POSIXct")
```

This approach allows others to more easily use and understand your dialogs, or indeed yourself in different circumstances. It supports code reuse as there are easy ways to find the relevant widgets once we have the dialogs.

Another approach is to use an explicit environment in which we assign variables and get fetch them later in other calls. This is what we get implicitly with a closure/lexical scoping. You can see how we use this approach in drag and drop with a tree control [treeCtrl.xml]. Basically, we can create a new environment in R with *new.env* and then assign variables into this using, e.g.

```
e = new.env()
e$spin = spin
```

And then later, we can fetch the values as long as we have access to **e** with

```
val = e$spin$GetValue()
```

and so on.

Validators

One of the difficulties we get into with this dialog (and with many that involve multiple, related controls) is ensuring that the values are consistent. For example, it is very easy for the user to set the starting date to be later in time than the end date, or vice versa. We have built in limits for the percentile, but in other cases we might want to ensure that the values were limited to a particular disjoint range, or that any text input was only characters and not numbers or only hexadecimal 'digits'. To do this, we need either to put event handlers on the individual controls, e.g. the two calendar controls, or alternatively use a validator.

```
h =  
function(ev) {  
    if(a$GetDate()$IsLaterThan(b$GetDate())) {  
        warning("start is later than end date")  
        ev$Skip(FALSE)  
        TRUE  
    } else  
        FALSE  
}  
a$AddCallback(wxEVT_CALENDAR_SEL_CHANGED, h)  
b$AddCallback(wxEVT_CALENDAR_SEL_CHANGED, h)
```

This function does not prohibit the selection, just issues a warning. We may want to change the color of the event widget to illustrate that it is invalid as it stands or explicitly reset the value to an appropriate time and date. But the latter is annoying when a user wants to set one and then other and allow them to be temporarily inconsistent.

We could use a wxWidgets validator to enforce this also. However, it is not clear that this saves us any work at this point in time and so we ignore it for now.