

90-01 All views – Basics - Calendars

The example shows how calendars are created and used and how the bar length changes when moving the bars due to the days off when the calendar is taken into account.

A Calendar object defines working and non-working times.

The example contains a function **createCalendarEntries**(startDate, endDate, publicHolidays), which defines the weekdays Monday, Tuesday, Wednesday, Thursday and Friday as working days and the weekends as non-working days for the specified period. In addition, an array can be used to specify which public holidays are to be considered non-working days.

Of course, the example can be modified as required and extended to include times of day.

In addition to the entries, the calendar definition only contains an ID that is required to be able to assign the calendar to the ActivityView with the **defaultCalendarID** option.

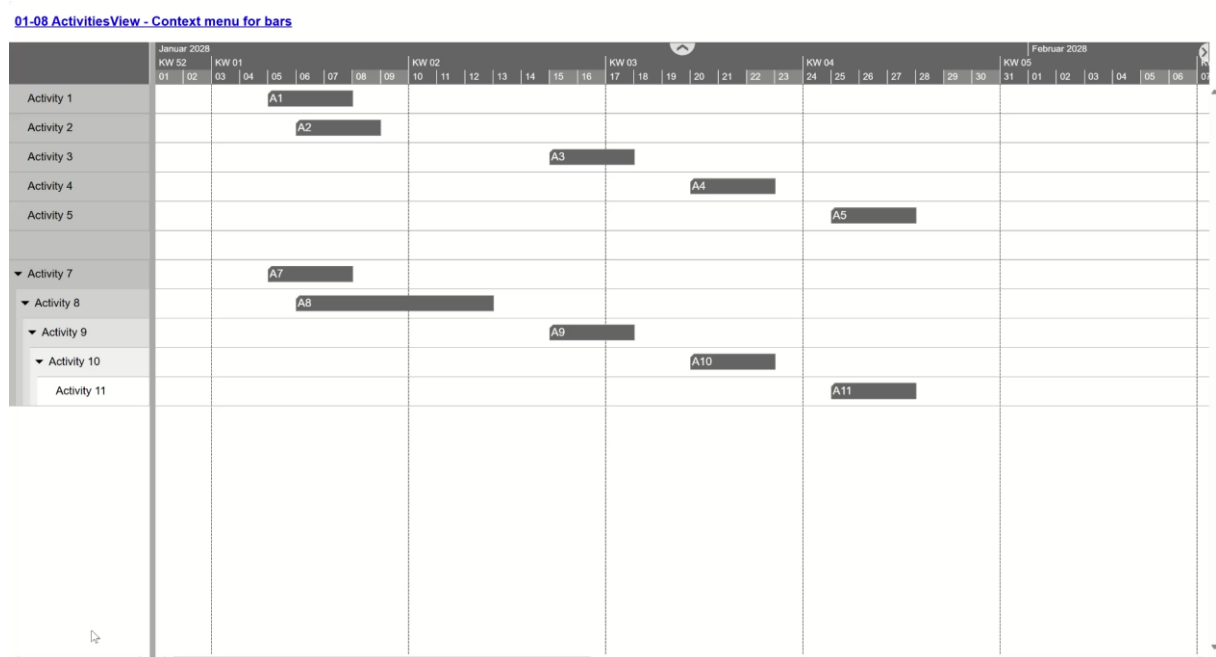
```
const calendars = [{
  ID: "Cal1",
  Entries: createCalendarEntries(timeAreaStart, timeAreaEnd, [new Date(2028,0,1)])
}];

vsWidget.option("defaultCalendarID", "Cal1");
```

Like all other objects, the Calendar must be made known to the **vsWidget** before the render method is called.

```
vsWidget.addCalendars(calendars);
vsWidget.addActivities(activities);
vsWidget.render();
```

A calendar is only fully visible if the flag **RowDesigns.CalendarGrid** is set for **defaultActivityExpandRowDesign** and **defaultActivityCollapsedRowDesign**.



Bars can be moved horizontally by setting the option

```
vsWidget.option("defaultActivityAllowedBarDragModes", BarDragModes.DragHorizontally);
```

To make the movements smoother, the **SnapTargets** are deactivated.

```
vsWidget.option({defaultActivitySnapTargetsForStart: SnapTargets.None,
    defaultActivitySnapTargetsForEnd: SnapTargets.None});
```

If the bars are only to be moved daily, the **timeStepUnit** option must be set accordingly.

```
vsWidget.option({timeStepUnit: TimeUnit.Days});
```

With interactive shifting, non-working times can be accounted for in the calendar if the option is enabled. This means the bars automatically extend to include the non-working time they contain.

```
vsWidget.option("ignoreCalendarOnActivityBarInteraction", false);
```

To make the presentation more compact, the non-working weekends, on which no activities usually take place, can be hidden.

```
vsWidget.option("nonworkingTimeVisible", false);
```



```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>01-09 NETRONIC VSW SE</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <link href="../../VSWidget/nwaf-apptools.min.css" rel="stylesheet"/>
  <link href="../../VSWidget/nwaf-table.min.css" rel="stylesheet"/>
  <link href="../../VSWidget/nwaf-gantt.min.css" rel="stylesheet"/>
  <link href="../../VSWidget/nwaf-rab.min.css" rel="stylesheet"/>
  <script src="https://cdn.jsdelivr.net/npm/hammerjs@2.0.8/hammer.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/d3@7.9.0/dist/d3.min.js"></script>
  <script src="https://cdn.jsdelivr.net/npm/tinycolor2@1.6.0/cjs/tinycolor.min.js"></script>
  <script src="../../VSWidget/nwaf-apptools.min.js"></script>
  <script src="../../VSWidget/nwaf-table.min.js"></script>
  <script src="../../VSWidget/nwaf-gantt.min.js"></script>
  <script src="../../VSWidget/nwaf-rab.min.js"></script>
  <script src="../../LicenseKey.js"></script>
  <script src="Miscellaneous.js"></script>
</head>
<body>
  <div id="toolbar">
```

```

<h3>
  <a href="docs/90-01_AllViews-Basics-Calendars.pdf"
    target="_blank">01-09 ActivitiesView - Calendars</a>
</h3>
<input id="weekendCalendar" type="checkbox" style="margin-left: 15px;"/>
<label for="weekendCalendar">Use weekend off calendar</label>
<input id="workingTimeOnly" type="checkbox"/>
<label for="workingTimeOnly">Show working time only</label>
<input id="takeCalendar" type="checkbox" style="margin-left: 15px;"/>
<label for="takeCalendar">Bars take the calendar into account</label>
<input id="dailyIncrements" type="checkbox"/>
<label for="dailyIncrements">Move in daily increments only</label>
</div>
<div id="VSWidget"></div>
<script>
  Miscellaneous.ready(() => {
    const { BarDragModes, RowDesigns, SnapTargets, TimeType, TimeUnit } = netronic.nVSW;

    document.getElementById('weekendCalendar').addEventListener('click', changeCalendar);
    document.getElementById('workingTimeOnly').addEventListener('click', changeCalendar);
    document.getElementById('takeCalendar').addEventListener('click', changeCalendar);
    document.getElementById('dailyIncrements').addEventListener('click', changeCalendar);

    const timeAreaStart = new Date(2028, 0, 1);
    const timeAreaEnd = new Date(2028, 3, 1);

    let options = {
      licenseKey: window.VSW_LICENSE_KEY,

      start: timeAreaStart,
      end: timeAreaEnd,

      defaultCalendarID: "Cal1",
      ignoreCalendarOnActivityBarInteractions: true,

      defaultActivitySnapTargetsForStart: SnapTargets.None,
      defaultActivitySnapTargetsForEnd: SnapTargets.None,

      defaultActivityExpandedRowDesign: RowDesigns.Bars |
                                         RowDesigns.CalendarGrid,
      defaultActivityCollapsedRowDesign: RowDesigns.Bars |
                                         RowDesigns.BarsInHiddenDescendantRows |
                                         RowDesigns.BarsStacked |
                                         RowDesigns.CalendarGrid,

      defaultActivityAllowedBarDragModes: BarDragModes.DragHorizontally |
                                           BarDragModes.DragEnd,

      defaultActivityBarSelectable: false,
      defaultActivityRowSelectable: false,
      multipleSelectionEnabled: false,

      maximumTimeResolutionUnit: TimeUnit.Hours,
      maximumTimeResolutionUnitFactor: 1,

      onDrop: (args) => {
        const activity = args.object;
        activity.Start = args.newStart;
        activity.End = args.newEnd;
        vsWidget.updateActivities([activity]);
        vsWidget.render();
      },
    };

    const vsWidget = Miscellaneous.instantiateVSWidget(document.querySelector("#VSWidget"), options);

    const calendars = createCalendars();
    vsWidget.addCalendars(calendars);
    const activities = createActivities();
    mapActivitiesToCurrentCalendar("Cal1");
    vsWidget.addActivities(activities);
    vsWidget.render();

    function createCalendars() {

```

```

const calendars = [
  {
    ID: "Cal1",
    Entries: [{Start: timeAreaStart, End: timeAreaEnd, TimeType: TimeType.WorkingTime}],
  },
  {
    ID: "Cal2",
    Entries: createCalendarEntries(timeAreaStart, timeAreaEnd, [new Date(2028,0,1)]),
  }
];

return calendars;
}

function createCalendarEntries(startDate, endDate, publicHolidays) {
  let currentDate = new Date(startDate);
  const lastDate = new Date(endDate);
  const uniqueDateStrings = Array.from(new Set(publicHolidays));
  const dateObjects = uniqueDateStrings.map(date => new Date(date));
  dateObjects.sort((a, b) => a - b);

  const calendarEntries = [];

  while (currentDate <= lastDate) {
    const dayOfWeek = currentDate.getDay();
    const startDay = new Date(currentDate);
    const endDay = new Date(new Date(currentDate).setDate(currentDate.getDate() + 1));

    if (dateObjects.length > 0 && dateObjects[0].getTime() === currentDate.getTime()){
      calendarEntries.push({
        Start: startDay,
        End: endDay,
        TimeType: TimeType.NonWorkingTime
      });
      dateObjects.shift();
    }
    else {
      switch (dayOfWeek) {
        case 0: // Sunday
          calendarEntries.push({
            Start: startDay,
            End: endDay,
            TimeType: TimeType.NonWorkingTime
          });
          break;

        case 1: // Monday
          calendarEntries.push({
            Start: startDay,
            End: endDay,
            TimeType: TimeType.WorkingTime
          });
          break;

        case 2: // Tuesday
          calendarEntries.push({
            Start: startDay,
            End: endDay,
            TimeType: TimeType.WorkingTime
          });
          break;

        case 3: // Wednesday
          calendarEntries.push({
            Start: startDay,
            End: endDay,
            TimeType: TimeType.WorkingTime
          });
          break;

        case 4: // Thursday
          calendarEntries.push({
            Start: startDay,
            End: endDay,

```

```

        TimeType: TimeType.WorkingTime
    });
    break;

    case 5: // Friday
        calendarEntries.push({
            Start: startDay,
            End: endDay,
            TimeType: TimeType.WorkingTime
        });
        break;

    case 6: // Saturday
        calendarEntries.push({
            Start: startDay,
            End: endDay,
            TimeType: TimeType.NonWorkingTime
        });
        break;
    }
}
currentDate.setDate(currentDate.getDate() + 1);
}
return calendarEntries;
}

function createActivities() {
    const activities = [
        {
            ID: "Act1",
            TableText: "Activity 1",
            BarText: "A1",
            Start: new Date(2028, 0, 5),
            my_WorkingDays: 3,
            ParentID: ""
        },
        {
            ID: "Act2",
            TableText: "Activity 2",
            BarText: "A2",
            Start: new Date(2028, 0, 6),
            my_WorkingDays: 3,
            ParentID: ""
        },
        {
            ID: "Act3",
            TableText: "Activity 3",
            BarText: "A3",
            Start: new Date(2028, 0, 15),
            my_WorkingDays: 3,
            ParentID: ""
        },
        {
            ID: "Act4",
            TableText: "Activity 4",
            BarText: "A4",
            Start: new Date(2028, 0, 20),
            my_WorkingDays: 3,
            ParentID: ""
        },
        {
            ID: "Act5",
            TableText: "Activity 5",
            BarText: "A5",
            Start: new Date(2028, 0, 25),
            my_WorkingDays: 3,
            ParentID: ""
        },
        {
            ID: "Act6",
            ParentID: ""
        },
        {
            ID: "Act7",

```

```

        TableText: "Activity 7",
        BarText: "A7",
        Start: new Date(2028, 0, 5),
        my_WorkingDays: 3,
        ParentID: ""
    },
    {
        ID: "Act8",
        TableText: "Activity 8",
        BarText: "A8",
        Start: new Date(2028, 0, 6),
        my_WorkingDays: 7,
        ParentID: "Act7"
    },
    {
        ID: "Act9",
        TableText: "Activity 9",
        BarText: "A9",
        Start: new Date(2028, 0, 15),
        my_WorkingDays: 3,
        ParentID: "Act8"
    },
    {
        ID: "Act10",
        TableText: "Activity 10",
        BarText: "A10",
        Start: new Date(2028, 0, 20),
        my_WorkingDays: 3,
        ParentID: "Act9"
    },
    {
        ID: "Act11",
        TableText: "Activity 11",
        BarText: "A11",
        Start: new Date(2028, 0, 25),
        my_WorkingDays: 3,
        ParentID: "Act10"
    }
];

const millisecondsPerDay = 24 * 60 * 60 * 1000;
for (const activity of activities)
    if (activity.Start) {
        activity.End = new Date(activity.Start.getTime() +
                                (activity.my_WorkingDays ?? 1) * millisecondsPerDay);
    }

return activities;
}

function changeCalendar(event) {
    const checkbox = event.target;

    if (checkbox.id === "weekendCalendar")
        vsWidget.option("defaultCalendarID", checkbox.checked ? 'Cal2' : 'Cal1');

    if (checkbox.id === "takeCalendar") {
        vsWidget.option("ignoreCalendarOnActivityBarInteractions", !checkbox.checked);
        mapActivitiesToCurrentCalendar(checkbox.checked ? "Cal2" : "Cal1");
        vsWidget.updateActivities(activities);
        vsWidget.render();
    }

    if (checkbox.id === "dailyIncrements")
        vsWidget.option("timeStepUnit", checkbox.checked ? TimeUnit.Days : TimeUnit.Seconds);

    if (checkbox.id === "workingTimeOnly")
        vsWidget.option("nonworkingTimeVisible", !checkbox.checked);
}

function mapActivitiesToCurrentCalendar(cal) {
    const millisecondsPerDay = 24 * 60 * 60 * 1000;
    for (const activity of activities)
        if (activity.Start) {

```

```
        activity.End = vsWidget.addWorkingTime(cal, activity.Start,  
                                                (activity.my_WorkingDays ?? 1) * millisecondsPerDay);  
    }  
    });  
</script>  
</body>  
</html>
```