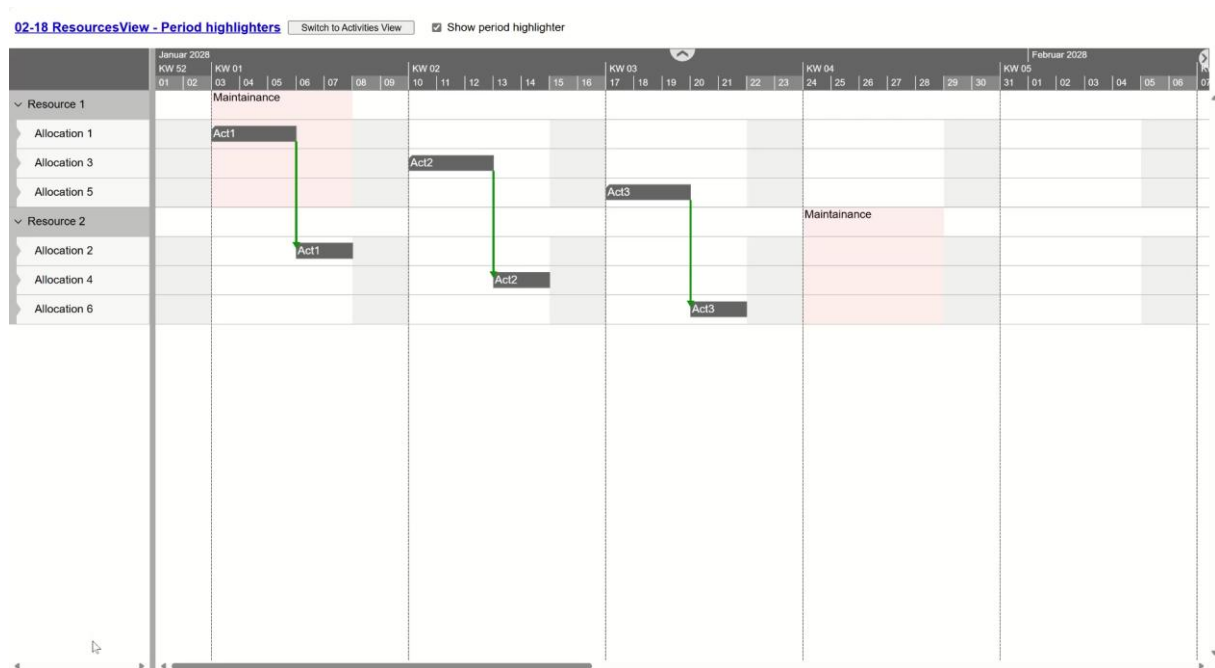


## 90-03 All views – Basics – Period highlighters

This example illustrates how to make use of period highlighters.

The **PeriodHighlighter** visually emphasises specific time periods and thus contributes to a better overview in the planning context. In this way, for example, planned shifts, company holidays or maintenance windows can be clearly marked.



PeriodHighlighters can be assigned to activities, resources and allocations. They are created centrally and managed via **addPeriodHighlighters**, **updatePeriodHighlighters**, **removePeriodHighlighters** in the VSWidget. Period Highlighters are a supplement to the use of calendars as **PeriodHighlighters** can also be used independently of the calendar mechanism.

The semantic meaning of highlighted time periods can be conveyed either through labels or associated tooltips. Especially in cases with overlapping visual elements, tooltips are often the preferred choice, as textual labels may otherwise be obscured by bars in the diagram.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>02-18 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/90-03_AllViews-Basics-Period_highlighters.pdf"
    target="_blank">90-03 All views - Basics - Period highlighters</a>
</h3>
<input id="activitiesView" type="button" value="Switch to Activities View"
  style="margin-left: 5px; height: 20px; width: 180px"/>
<input style="margin-left: 20px" id="showPeriodHighlighter" type="checkbox" checked/>
<label for="showPeriodHighlighter">Show period highlighter</label>
</div>
<div id="VSWidget"></div>
<script>
  Miscellaneous.ready(() => {
    const { BarDragModes, BarSortMode, CollapseState, LinkRoutingType,
      LinkMarker, RelationType, RowDesigns, TimeType, ViewType } = netronic.nVSW;

    document.getElementById("activitiesView").addEventListener("click", changeViewType);
    document.getElementById("showPeriodHighlighter").addEventListener("click",
      changePeriodHighlighterVisibility);

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

    const options = {
      licenseKey: window.VSW_LICENSE_KEY,

      viewType: ViewType.Resources,

      start: timeAreaStart,
      end: timeAreaEnd,

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

      defaultResourceExpandedRowDesign: RowDesigns.Bars |
        RowDesigns.BarsStacked,
      defaultResourceCollapsedRowDesign: RowDesigns.Bars |
        RowDesigns.BarsInHiddenDescendantRows |
        RowDesigns.BarsStacked,

      defaultCalendarID: "Cal2",
      allocationBarSortModeForStackedRowDesign: BarSortMode.StartAndEnd,
      defaultAllocationAllowedBarDragModes: BarDragModes.Horizontal |
        BarDragModes.Vertical |
        BarDragModes.ResizeStart |
        BarDragModes.ResizeEnd,

      defaultAllocationBarSelectable: true,
      defaultActivityBarSelectable: false,
      defaultResourceRowSelectable: false,
      defaultAllocationRowSelectable: false,
      defaultActivityRowSelectable: false,
      multipleSelectionEnabled: false,

      linksVisibleInResourcesView: true,
      linksVisibleInActivitiesView: true,
      allocationRowsVisibleInResourcesView: true,
      definedAllocationLinksVisibleInResourcesView: true,
      allocationRowsVisibleInActivitiesView: true,
      definedAllocationLinksVisibleInActivitiesView: true,

      ignoreCalendarOnAllocationBarInteractions: true,
      ignoreCalendarOnActivityBarInteractions: true,

      defaultLinkRoutingType: LinkRoutingType.Curved,
      defaultLinkSelectable: false,
      defaultLinkTargetMarker: LinkMarker.FilledArrow,
      defaultLinkTooltipTemplateID: "",
      fixedTableColumnWidth: 0,

      onCollapseStateChanged: (args) => {
        if ((vsWidget.option("viewType") === ViewType.Resources) && args.periodHighlighter) {
          args.periodHighlighter.Entries[0].Caption = args.newCollapseState ===
            CollapseState.Collapsed ? "" : "Maintenance";
        }
      }
    };
  });

```

```

        vsWidget.updatePeriodHighlighters(periodHighlighters);
        vsWidget.render();
    }
},
};

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

const tooltips = createTooltipTemplates();
const calendars = createCalendars();
const periodHighlighters = createPeriodHighlighters();
const resources = createResources();
const allocations = createAllocations();
const activities = createActivities();
const links = createLinks();

vsWidget.addTooltipTemplates(tooltips);
vsWidget.addCalendars(calendars);
vsWidget.addPeriodHighlighters(periodHighlighters);
vsWidget.addResources(resources);
vsWidget.addAllocations(allocations);
vsWidget.addActivities(activities);
vsWidget.addLinks(links);
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);

    if (currentDate > lastDate) {
        //console.error("The end date must be after the start date.");
        return;
    }

    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 createResources() {
    const resources = [
        {
            ID: "Res1",
            TableText: "Resource 1",
            CalendarId: "Cal2",
            AllocationRowsCollapseState: CollapseState.Expanded,
            PeriodHighlighterID: "P1",
        },
        {
            ID: "Res2",
            TableText: "Resource 2",
            CalendarId: "Cal2",
            AllocationRowsCollapseState: CollapseState.Expanded,

```

```

        PeriodHighlighterID: "P2",
    },
];

return resources;
}

function createAllocations() {
    const allocations = [
        {
            ID: "Alloc1",
            ParentID: "",
            ResourceID: "Res1",
            ActivityID: "Act1",
            TableText: "Allocation 1",
            BarText: "A1",
            Start: new Date(2028, 0, 3),
            my_WorkingDays: 3,
        },
        {
            ID: "Alloc2",
            ParentID: "",
            ResourceID: "Res2",
            ActivityID: "Act1",
            TableText: "Allocation 2",
            BarText: "A1",
            Start: new Date(2028, 0, 6),
            my_WorkingDays: 2,
        },
        {
            ID: "Alloc3",
            ParentID: "",
            ResourceID: "Res1",
            ActivityID: "Act2",
            TableText: "Allocation 3",
            BarText: "A2",
            Start: new Date(2028, 0, 10),
            my_WorkingDays: 3,
        },
        {
            ID: "Alloc4",
            ParentID: "",
            ResourceID: "Res2",
            ActivityID: "Act2",
            TableText: "Allocation 4",
            BarText: "A2",
            Start: new Date(2028, 0, 13),
            my_WorkingDays: 2,
        },
        {
            ID: "Alloc5",
            ParentID: "",
            ResourceID: "Res1",
            ActivityID: "Act3",
            TableText: "Allocation 5",
            BarText: "A3",
            Start: new Date(2028, 0, 17),
            my_WorkingDays: 3,
        },
        {
            ID: "Alloc6",
            ParentID: "",
            ResourceID: "Res2",
            ActivityID: "Act3",
            TableText: "Allocation 6",
            BarText: "A3",
            Start: new Date(2028, 0, 20),
            my_WorkingDays: 2,
        },
    ];

    const millisecondsPerDay = 24 * 60 * 60 * 1000;
    for (const allocation of allocations) {
        allocation.End = new Date(allocation.Start.getTime() +

```

```

        (allocation.my_WorkingDays ?? 1) * millisecondsPerDay);
    allocation.Entries = [
        {
            Start: allocation.Start,
            End: allocation.End,
        },
    ];
}

return allocations;
}

function createActivities() {
    const activities = [
        {
            ID: "Act1",
            TableText: "Activity 1",
            BarText: "A1",
            Start: new Date(2028, 0, 3),
            End: new Date(2028, 0, 8),
        },
        {
            ID: "Act2",
            TableText: "Activity 2",
            BarText: "A2",
            Start: new Date(2028, 0, 10),
            End: new Date(2028, 0, 15),
        },
        {
            ID: "Act3",
            TableText: "Activity 3",
            BarText: "A3",
            Start: new Date(2028, 0, 17),
            End: new Date(2028, 0, 23),
        },
    ],
    ];

    return activities;
}

function createLinks() {
    const links = [
        {
            ID: "L1",
            SourceActivityID: "Act1",
            TargetActivityID: "Act2",
            Width: 3,
            Color: "red",
        },
        {
            ID: "L2",
            SourceActivityID: "Act2",
            TargetActivityID: "Act3",
            Color: "red",
            Width: 3,
        },
        {
            ID: "L3",
            SourceAllocationID: "Alloc1",
            TargetAllocationID: "Alloc2",
            Color: "green",
            Width: 3,
            RelationType: RelationType.FinishToStart,
        },
        {
            ID: "L4",
            SourceAllocationID: "Alloc3",
            TargetAllocationID: "Alloc4",
            Color: "green",
            Width: 3,
        },
        {
            ID: "L5",
            SourceAllocationID: "Alloc5",

```

```

        TargetAllocationID: "Alloc6",
        Color: "green",
        Width: 3,
    },
];

return links;
}

function createPeriodHighlighters() {
    const periodHighlighters = [
        {
            ID: "P1",
            TooltipText: "Maintenance",
            Entries: [
                {
                    Start: new Date(2028, 0, 3),
                    End: new Date(2028, 0, 8),
                    Color: "rgba(255,200,200,0.3)",
                    Caption: "Maintenance",
                    CaptionColor: "black",
                    TooltipTemplateID: "Tooltip1",
                },
            ],
        },
        {
            ID: "P2",
            TooltipText: "Maintenance",
            Entries: [
                {
                    Start: new Date(2028, 0, 24),
                    End: new Date(2028, 0, 29),
                    Color: "rgba(255,200,200,0.3)",
                    Caption: "Maintenance",
                    CaptionColor: "black",
                    TooltipTemplateID: "Tooltip1",
                },
            ],
        },
    ];

    return periodHighlighters;
}

function createTooltipTemplates() {
    const tooltips = [
        {
            ID: "Tooltip1",
            IsInteractive: false,
            HTMLFormat: `{{TooltipText}}`,
        },
    ];

    return tooltips;
}

function changeViewType(event) {
    const btn = event.target;

    const view = vsWidget.option("viewType");
    if (view === ViewType.Resources) {
        btn.value = "Switch to Resource View";

        vsWidget.option("viewType", ViewType.Activities);

        allocations.forEach((allocation) => {
            allocation.BarText = `R${allocation.ResourceID.substring(3)}`;
        });
        periodHighlighters.forEach((periodHighlighter) => {
            periodHighlighter.Entries[0].Caption = "";
        });
    }
    else {
        btn.value = "Switch to Activities View";
    }
}

```

```

        vsWidget.option("viewType", ViewType.Resources);
        allocations.forEach((allocation) => {
            allocation.TableText = `Allocation ${allocation.ID.substring(5)}`;
            allocation.BarText = `A${allocation.ActivityID.substring(3)}`;
        });
        periodHighlighters.forEach((periodHighlighter) => {
            periodHighlighter.Entries[0].Caption = "Maintenance";
        });
    }

    vsWidget.updateAllocations(allocations);
    vsWidget.updatePeriodHighlighters(periodHighlighters);
    vsWidget.render();
}

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

    if (checkbox.checked) {
        vsWidget.addPeriodHighlighters(periodHighlighters);
    }
    else {
        vsWidget.removePeriodHighlighters(periodHighlighters);
    }

    vsWidget.render();
}
});
</script>
</body>
</html>

```