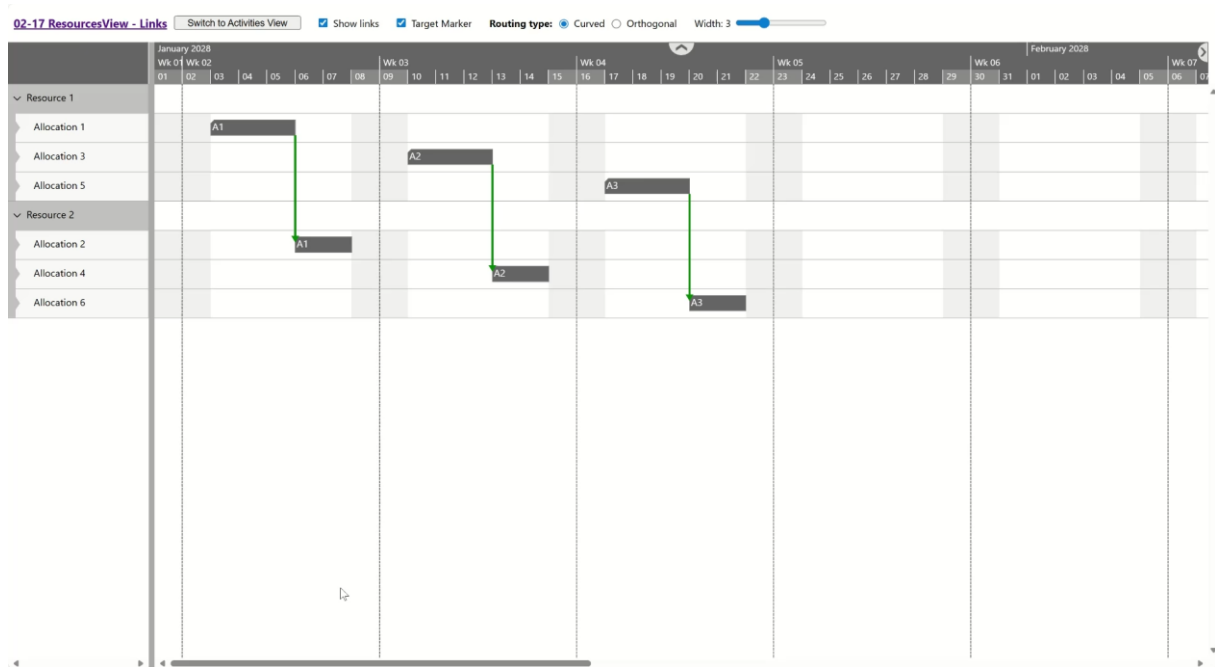


20-03 Resources view – Basics – Links between allocations

This example illustrates how to utilise links between allocations in the ResourceView and between activities in the ActivityView.



Links are used to establish connections between activities or between allocations. These connections are directional (with a defined source and target), and the target end may optionally be indicated with an arrowhead (**TargetMarker**). Only one connection can exist between any two elements at a time.

Links serve a purely graphical purpose and are not semantically constrained within the library. It is the responsibility of the implementing application to utilize them appropriately according to the intended use case.

The general visibility of links within the **ResourceView** can be controlled via an option:

```
linksVisibleInResourceView: true,
```

The appearance of a link can be customized through properties such as **Color**, **Width**, and **DashArray**. The routing of the connection (**RoutingType**) can be either curved or orthogonal (right-angled).

In the **ResourceView**, you can specify whether allocations should appear in separate lines or in the resource line.

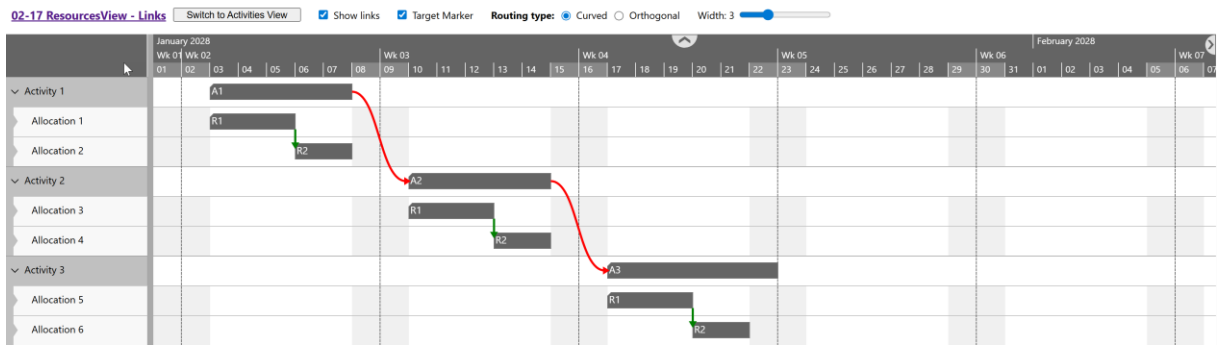
```
allocationRowsVisibleInResourcesView: true,
```

Allocations lines can also be displayed in the **ActivitiesView** if desired:

```
allocationRowsVisibleInActivitiesView: true,
```

This can be done with or without links:

```
definedAllocationLinksVisibleInActivitiesView: true,
```



```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>02-17 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/20-03_ResourcesView-Basics-Links_between_allocations.pdf"
        target="_blank">20-03 Resources view - Basics - Links between allocations</a>
    </h3>
    <input id="viewType" type="button" value="Switch to Activities View"
      style="margin-left: 5px; height: 20px; width: 180px"/>
    <input style="margin-left: 20px" id="showLinks" type="checkbox" checked=""/>
    <label for="showLinks">Show links</label>
    <input style="margin-left: 20px" id="showTargetMarker" type="checkbox" checked=""/>
    <label for="showTargetMarker">Target Marker</label>
    <label style="margin-left: 20px">
      <b>Routing type:</b>
    </label>
    <input type="radio" id="linkTypeCurved" name="routingType" checked=""/>
    <label for="linkTypeCurved">Curved</label>
    <input type="radio" id="linkTypeOrthogonal" name="routingType"/>
    <label for="linkTypeOrthogonal">Orthogonal</label>
    <label style="margin-left: 20px" for="linkWidth">
      Width:
      <span class="slider-output" id="output1">3</span>
    </label>
    <input type="range" id="linkWidth" min="1" max="8" value="3"
      data-output="output1" data-option="Width"/>
  </div>
  <div id="VSWidget"></div>
  <script>
    Miscellaneous.ready(() => {
      const { BarDragModes, BarSortMode, CollapseState, LinkRoutingType, LinkMarker,
        RelationType, RowDesigns, TimeType, ViewType } = netronic.nVSW;

      document.getElementById("viewType").addEventListener("click", changeViewType);
      document.getElementById("showLinks").addEventListener("click", changeLinkDesign);
      document.getElementById("showTargetMarker").addEventListener("click", changeLinkDesign);
      document.getElementById("linkTypeCurved").addEventListener("click", changeLinkDesign);
      document.getElementById("linkTypeOrthogonal").addEventListener("click", changeLinkDesign);
      document.getElementById("linkWidth").addEventListener("input", changeLinkWidth);

      const timeAreaStart = new Date(2028, 0, 1);
      const timeAreaEnd = new Date(2028, 3, 1);
    });
  </script>
</body>
</html>
```

```

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,
  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,
};

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

const calendars = createCalendars();
const resources = createResources();
const allocations = createAllocations();
const activities = createActivities();
const links = createLinks();

vsWidget.addCalendars(calendars);
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 millisecondsPerDay = 24 * 60 * 60 * 1000;

  const calendarEntries = [];
  while (currentDate <= lastDate) {
    const dayOfWeek = currentDate.getDay();
    const startDay = new Date(currentDate);
    const endDay = new Date(currentDate.getTime() + millisecondsPerDay); // Add one day

    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;
      }
    }
    currentDate = endDay;
  }
}

```

```

        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() {
    return [
        {
            ID: "Res1",
            TableText: "Resource 1",
            CalendarId: "Cal2",
            AllocationRowsCollapseState: CollapseState.Expanded,
        },
        {
            ID: "Res2",
            TableText: "Resource 2",
            CalendarId: "Cal2",
            AllocationRowsCollapseState: CollapseState.Expanded,
        },
    ];
}

function createAllocations() {
    const allocations = [
        {
            ID: "Alloc1",
            ParentID: "",
            ResourceID: "Res1",
            ActivityID: "Act1",
            TableText: "Allocation 1",
            BarText: "A1",
            Start: new Date(2028, 0, 3),
            WorkingDays: 3,
        },
        {
            ID: "Alloc2",
            ParentID: "",
            ResourceID: "Res2",
            ActivityID: "Act1",
            TableText: "Allocation 2",
            BarText: "A1",
            Start: new Date(2028, 0, 6),
            WorkingDays: 2,
        },
        {
            ID: "Alloc3",
            ParentID: "",

```

```

        ResourceID: "Res1",
        ActivityID: "Act2",
        TableText: "Allocation 3",
        BarText: "A2",
        Start: new Date(2028, 0, 10),
        WorkingDays: 3,
    },
    {
        ID: "Alloc4",
        ParentID: "",
        ResourceID: "Res2",
        ActivityID: "Act2",
        TableText: "Allocation 4",
        BarText: "A2",
        Start: new Date(2028, 0, 13),
        WorkingDays: 2,
    },
    {
        ID: "Alloc5",
        ParentID: "",
        ResourceID: "Res1",
        ActivityID: "Act3",
        TableText: "Allocation 5",
        BarText: "A3",
        Start: new Date(2028, 0, 17),
        WorkingDays: 3,
    },
    {
        ID: "Alloc6",
        ParentID: "",
        ResourceID: "Res2",
        ActivityID: "Act3",
        TableText: "Allocation 6",
        BarText: "A3",
        Start: new Date(2028, 0, 20),
        WorkingDays: 2,
    },
];

const millisecondsPerDay = 24 * 60 * 60 * 1000;
for (const allocation of allocations) {
    allocation.End = new Date(allocation.Start.getTime() + (allocation.WorkingDays ?? 1) *
millisecondsPerDay);
    allocation.EndDay = new Date(allocation.Start.getTime() + ((allocation.WorkingDays ?? 1) -
1) * millisecondsPerDay);
    allocation.RunningDays = allocation.WorkingDays;
    allocation.Entries = [
        {
            Start: allocation.Start,
            End: allocation.End,
        },
    ];
}

return allocations;
}

function createActivities() {
    return [
        {
            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),
    },
];
}

function createLinks() {
    return [
        {
            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,
        },
    ];
}

function changeViewType(btn) {
    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)}`;
        });
        vsWidget.updateAllocations(allocations);
        vsWidget.render();
    }
    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)}`;
        });
        vsWidget.updateAllocations(allocations);
    }
}

```

```

        vsWidget.render();
    }
}

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

    if (checkbox.id === "showLinks") {
        vsWidget.option("linksVisibleInActivitiesView", checkbox.checked);
        vsWidget.option("linksVisibleInResourcesView", checkbox.checked);
    } else if (checkbox.id === "showTargetMarker") {
        for (const link of links)
            link.TargetMarker = checkbox.checked ? LinkMarker.FilledArrow : LinkMarker.None;

        vsWidget.updateLinks(links);
        vsWidget.render();
    } else if (checkbox.id === "linkTypeCurved")
        vsWidget.option("defaultLinkRoutingType", LinkRoutingType.Curved);
    else if (checkbox.id === "linkTypeOrthogonal")
        vsWidget.option("defaultLinkRoutingType", LinkRoutingType.Orthogonal);
}

function changeLinkWidth(event) {
    const slider = event.target;

    const value = slider.value;
    document.getElementById(slider.dataset.output).innerText = value.toString().padStart(3, " ");
    for (const link of links)
        link.Width = +value;

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

```