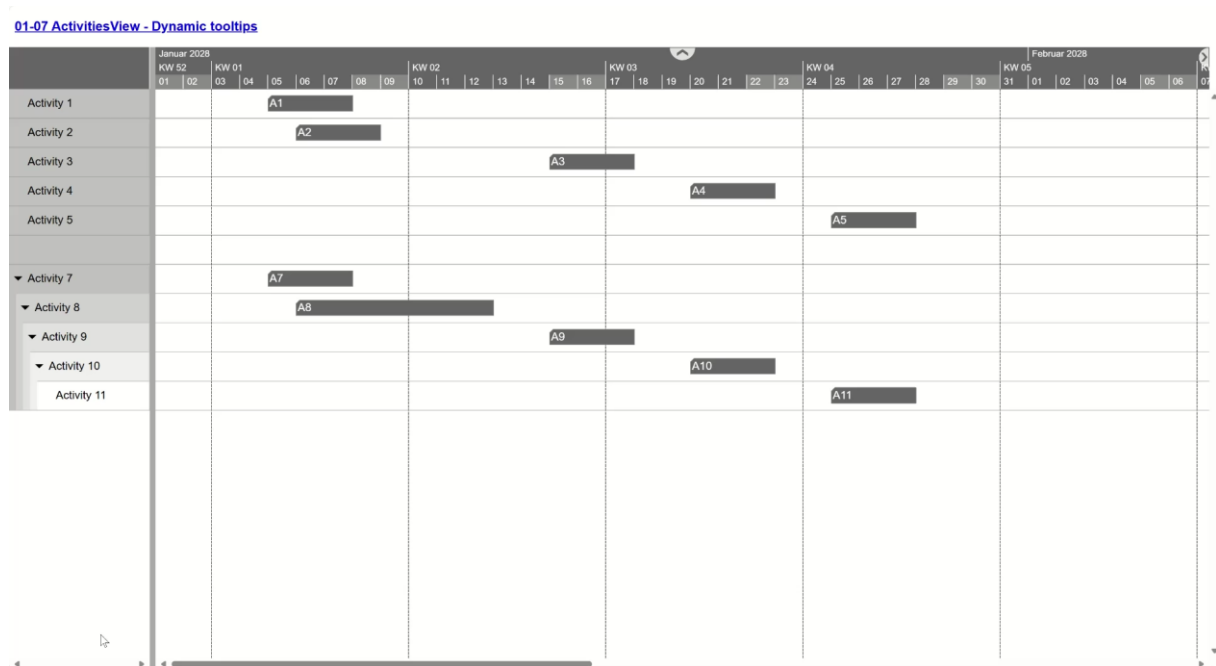


94-02 All views – Tooltips – Advanced tooltips

This example shows how tooltips can be created from a description using a support method.



The signature of the support method is as follows:

createTooltipContent(left, top, obj, tooltipDef)

createTooltipContent(left, top, obj, tooltipDef, customFormatters)

left	X-coordinate of the position of the tooltip
top	Y-coordinate of the position of the tooltip
obj	Object that contains the information to be displayed
tooltipDef	Description of the structure of the tooltip
customFormatters	Formatting functions that convert the value to be displayed into display form. The parameter can be omitted, it is optional.

Activity	
ID	Act2
Name	Activity 2
Start	06.01.2028
End	09.01.2028
Working time	3 days
Running time	3 days
pdf	Dynamic Tooltips

The structure of the tooltip definition object is as follows:

```
const tooltipDefActivity = {
  Items: [
    {Header: 'Activity'},
    {Header: 'ID', Source: 'ID'},
    {Header: 'Name', Source: 'TableText'},
    {Header: 'Start', Source: 'Start', Formatter: 'fmtDate'},
    {Header: 'End', Source: 'End', Formatter: 'fmtDate'},
    {Header: 'Working time', Source: 'WorkingDays', Suffix: " days"}
  ]
}
```

```

    {Header: 'Running time', Source: 'RunningDays', Suffix: " days"},
    {Header: 'pdf', Source: ['DocUrl', 'DocName'], Hyperlink: true}
  ]
};

```

The first of the items in the tooltip definition describes the title line of the tooltip. Each additional item describes a line in the tooltip. The possible item entries are:

Header	Name to be used in the tooltip for the field.
Source	The name of the field whose content is to be displayed. It is possible to specify a single field or multiple fields in an array. This is necessary, for example, if the formatting function requires multiple entries to be merged.
Formatter	This is an object with one or more functions for formatting the output. The functions fmtDate and fmtDateTime are already included in the support function, but can be overwritten if you use the same function name. The signature of the formatting function is: <code>functionName: function(header, args, country) {...}</code>
Prefix	The specified text is placed in front of the formatted entry.
Suffix	The specified text is appended to the formatted entry at the end.
Hyperlink	If the value is set to true, the entry is displayed as a link. If only one field is specified, then the visible content is the URL, otherwise a different name can be displayed. Hyperlinks are only useful if the tooltips appear as a window.
Decimals	For numeric fields, the decimal places to be displayed can be specified.
Limit	This limits the length of the output to the specified number of characters. An ellipsis character is added to indicate the omission.
Default	If no entry exists and the default value does not fit, a different value can be entered here.
OmitEmpty	By entering a Boolean value, you can specify whether you want empty entries to appear in the tooltip or not.
Separator	If an array of entries exist, they can be linked together with the specified separators.

There are three properties that can be set at the top level:

OmitEmpty	By specifying a Boolean value, you can define globally whether empty entries should appear in the tooltip for all entries. You can deviate from this value for each field by setting it.
ShowAsWindow	The tooltip appears as a window. It disappears by pressing the ESC key or by closing it with x.
Items	[...] A line description object is required for each line in the tooltip:

```

const tooltipDefActivity = {
  OmitEmpty: true,
  ShowAsWindow: false,
  Items: [...]
};

```

Here is an extended example with its own formatter object and a modified title line object:

```

{Header: 'Activity', Source: 'ID', Formatter: 'fmtTitle'}

```

Alternatively, it is also possible to specify the self-defined formatting functions directly as a function reference. In this case, there is no need to pass the formatter object.

```

{Header: 'Activity', Source: 'ID', Formatter: formatters.fmtTitle}

```

Activity #2		✕	
ID	Act2		
Name	Activity 2		
Start	2028-01-10		
End	2028-01-13		
Working time	3 days		
Running time	3 days		
pdf	Dynamic Tooltips		

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>01-07 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/94-02_AllViews-Tooltips-Advanced_tooltips.pdf"
        target="_blank">94-02 All views - Tooltips - Advanced tooltips</a>
    </h3>
  </div>
  <div id="VSWidget"></div>
  <script>
    Miscellaneous.ready(() => {
      const { BarDragModes, ObjectType, RowDesigns,
        SnapTargets, TimeType, VisualType } = netronic.nVSW;

      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,

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

        onShowTooltip: (args) => {
          const tooltip = document.querySelector(".tooltip");
          const delay = tooltipDefinition.ShowAsWindow === false ? vsWidget.option("tooltipDelay") : 0;

```

```

        if (args.objectType === ObjectType.Activity && args.visualType === VisualType.Bar) {
            tooltip?.remove();
            setTimeout(() => {createTooltipContent(args.event.pageX - 130, args.event.pageY + 35,
                                                    args.object, tooltipDefinition);}, delay);
        }
        else {
            if (tooltipDefinition.ShowAsWindow === false)
                tooltip?.remove();
        }
    },

    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 defaultFormatters = {
    fmtDate: (_header, args, locale = "en-US") =>
        args.length === 0 ? "" : new Date(args[0]).toLocaleDateString(locale, {
            year: "numeric",
            month: "2-digit",
            day: "2-digit",
        })),

    fmtDateTime: (_header, args, locale = "en-US") => {
        if (!args.length)
            return "";

        const date = new Date(args[0]);
        return `${date.toLocaleDateString(locale, {
            year: "numeric",
            month: "2-digit",
            day: "2-digit",
        })} ${date.toLocaleTimeString(locale, {
            hour: "2-digit",
            minute: "2-digit",
        })}`;
    },
};

const tooltipDefinition = createTooltipDefinition();
const calendars = createCalendars();
const activities = createActivities();
vsWidget.addCalendars(calendars);
vsWidget.addActivities(activities);
vsWidget.render();

function createTooltipDefinition() {
    return {
        ShowAsWindow: true,
        Items: [
            { Header: "Activity" },
            { Header: "ID", Source: "ID" },
            { Header: "Name", Source: "TableText" },
            { Header: "Start", Source: "Start", Formatter: "fmtDateTime" },
            { Header: "End", Source: "End", Formatter: "fmtDateTime" },
            { Header: "Working time", Source: "my_WorkingDays", Suffix: " days" },
            { Header: "Running time", Source: "my_RunningDays", Suffix: " days" },
            { Header: "pdf", Source: ["my_DocUrl", "my_DocName"], Hyperlink: true },
        ],
    };
}

function createCalendars() {
    const calendars = [
        {
            ID: "Call",
            Entries: [

```

```

        {
            Start: timeAreaStart,
            End: timeAreaEnd,
            TimeType: TimeType.WorkingTime,
        },
    ],
},
];

return calendars;
}

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);
    activity.my_RunningDays = activity.my_WorkingDays;
  }

for (let i = 0; i < 3; i++) {
  activities[i].my_DocUrl = "pdf/01-06_ActivitiesView_Tooltips.pdf";
  activities[i].my_DocName = "Dynamic Tooltips";
}

return activities;
}

function createTooltipContent(left, top, obj, tooltipDefinition, customFormatters) {
  if (!obj ||
    !tooltipDefinition?.Items?.length ||
    typeof left !== "number" || !Number.isFinite(left) ||
    typeof top !== "number" || !Number.isFinite(top))
    return;

  if (!document.getElementById("custom-tooltip-style"))
    injectTooltipStyles();

  const formatters = { ...defaultFormatters, ...customFormatters };

  const tooltip = document.createElement("div");
  tooltip.className = "tooltip";
  tooltip.id = "tooltip";
  tooltip.style.display = "block";

  const header = document.createElement("div");
  header.className = "tooltip-header";

  const title = document.createElement("span");
  let titleSeparator = getPropVal(tooltipDefinition.Items[0], "Separator");
  titleSeparator = titleSeparator === "" ? " " : titleSeparator;

  let limitTitle = getPropVal(tooltipDefinition.Items[0], "Limit");
  limitTitle = limitTitle === "" ? 0 : limitTitle;

  title.textContent = applyFormatter(
    formatters,
    getPropVal(tooltipDefinition.Items[0], "Formatter"),
    getPropVal(tooltipDefinition.Items[0], "Header"),
    getValues(obj, getPropVal(tooltipDefinition.Items[0], "Source")),
  );

```

```

    0,
    titleSeparator
);

if (limitTitle > 0 && title.textContent.length > limitTitle)
    title.textContent = title.textContent.slice(0, limitTitle) + "...";

if (title.textContent === "")
    title.textContent = getPropVal(tooltipDefinition.Items[0], "Header");

header.appendChild(title);

if (getPropVal(tooltipDefinition, "ShowAsWindow")) {
    const closeBtn = document.createElement("span");
    closeBtn.className = "close-btn";
    closeBtn.id = "closeBtn";
    closeBtn.textContent = "✕";
    closeBtn.onclick = () => {
        tooltip.style.display = "none";
    };
    header.appendChild(closeBtn);
}

let container = document.createElement("div");
container.className = "container";

let table = document.createElement("table");
table.className = "table-container";

const omitEmptyGlobal = getPropVal(tooltipDefinition, "OmitEmpty", true);
for (let i = 1; i < tooltipDefinition.Items.length; i++) {
    const omitEmpty = getPropVal(
        tooltipDefinition.Items[i],
        "OmitEmpty",
        omitEmptyGlobal
    );
    const header = getPropVal(tooltipDefinition.Items[i], "Header");
    const separator = getPropVal(tooltipDefinition.Items[i], "Separator", " ");
    const decimals = getPropVal(tooltipDefinition.Items[i], "Decimals", 0);
    const limit = getPropVal(tooltipDefinition.Items[i], "Limit", 0);
    const defaultVal = getPropVal(tooltipDefinition.Items[i], "Default", "");

    const values = getValues(
        obj,
        getPropVal(tooltipDefinition.Items[i], "Source", ""),
        defaultVal
    );
    let content = [
        getPropVal(tooltipDefinition.Items[i], "Prefix"),
        applyFormatter(
            formatters,
            getPropVal(tooltipDefinition.Items[i], "Formatter"),
            header,
            values,
            decimals,
            separator
        ),
        getPropVal(tooltipDefinition.Items[i], "Suffix"),
    ]
        .join("")
        .trim();

    if (limit > 0 && content.length > limit)
        content = content.slice(0, limit) + "...";

    if (content === "" && omitEmpty)
        continue;

    let row = document.createElement("tr");
    let headerCell = document.createElement("td");
    headerCell.className = "table-rowHeader";
    headerCell.textContent = header;

```

```

let dataCell = document.createElement("td");
dataCell.className = "table-data";
dataCell.textContent = content;

row.appendChild(headerCell);

if (getPropVal(tooltipDefinition.Items[i], "Hyperlink")) {
  const link = document.createElement("a");
  const url = values[0] ?? "";
  const label = values.length == 2 ? values[1] : url;
  link.href = url;
  link.target = "_blank";
  link.textContent = label;
  link.className = "table-data";
  row.appendChild(link);
}
else
  row.appendChild(dataCell);

table.appendChild(row);
}

tooltip.appendChild(header);
tooltip.appendChild(container);
tooltip.style.left = `${left}px`;
tooltip.style.top = `${top}px`;

container.appendChild(table);
document.body.appendChild(tooltip);
}

function injectTooltipStyles() {
  const style = document.createElement("style");
  style.id = "custom-tooltip-style";
  style.textContent = `
    .tooltip {
      position: absolute;
      background: white;
      border-radius: 0px;
      display: none;
      z-index: 1000;
      box-shadow: 4px 4px 8px rgba(0, 0, 0, 0.6);
      border: 1px solid #999;
    }

    .tooltip-header {
      background-color: #ccc;
      color: #222;
      padding: 2px 6px;
      cursor: grab;
      display: flex;
      font-size: 13px;
      font-weight: bold;
      justify-content: space-between;
      align-items: center;
      user-select: none;
    }

    .close-btn {
      cursor: pointer;
      color: #555;
      font-size: 13px;
      margin-left: auto;
    }

    .close-btn:hover {
      color: red;
    }

    .table-container {
      width: 100%;
      max-width: 400px;
      border-collapse: collapse;
      box-shadow: 0 0 3px rgba(0, 0, 0, 0.1);
    }
  `;
  document.head.appendChild(style);
}

```



```

    }

    .table-rowHeader {
        padding: 2px 6px;
        background-color: #f1f1f1;
        text-align: left;
        border-bottom: 1px solid #ddd;
        font-size: 12px;
        line-height: 1.2;
    }

    .table-data {
        padding: 2px 6px;
        text-align: left;
        border-bottom: 1px solid #ddd;
        font-size: 12px;
        line-height: 1.2;
    }
}

`
;
document.head.appendChild(style);
}

function applyFormatter(formatters, formatter, header, values,
                        decimals = 0, separator = "", locale = "en-US") {
    const args = Array.isArray(values) ? values : [values];

    //if (!formatters?.length)
    if (Object.entries(formatters).length === 0)
        return roundArrayNumbers(args, decimals).join(separator);
    else if (typeof formatter === "function")
        return formatter(header, args, locale);
    else if (typeof formatter === "string") {
        const formatterFunction = formatters[formatter];
        if (typeof formatterFunction === "function")
            return formatterFunction(header, args, locale);
    }
    else if (args.every((val) => typeof val === "number"))
        return roundArrayNumbers(args, decimals).join(separator);

    return args.join(separator);
}

function getPropVal(obj, propertyName, defaultVal = "") {
    const value = obj?.[propertyName];
    return (value ?? "") === "" ? defaultVal : value;
}

function getValues(obj, fields, defaultVal) {
    const dataFields = Array.isArray(fields) ? fields : [fields];
    return dataFields.map((x) => getPropVal(obj, x, defaultVal));
}

function roundArrayNumbers(arr, decimals) {
    return arr.map((item) => typeof item === "number" &&
        !isNaN(item) ? +item.toFixed(decimals) : item);
}

});
</script>
</body>
</html>

```