

## 30-01 Loads view – Basics – Capacity and load

This example shows the loads view, which allows you to visualize the load and utilization of resources or resource groups over time.



The **LoadsView** (viewType: `ViewType.Loads`) provides a focused perspective on resource utilization.

Unlike other views, no bars are displayed. Instead, you see capacity curves and load curves that illustrate how effectively resources are used within a given time span. This makes bottleneck analysis and capacity balancing straightforward:

- **CapacityCurves** show the available capacity.
- **LoadCurves** display the current resource usage.

Each resource offers data fields for these two curve types. Curves are added to the widget via the **addCurves()** method. Individual colors and line styles improve clarity, especially when stacking curves from a resource group. Special tooltips allow for a more detailed examination.

For planning purposes, the LoadsView can also highlight collisions in the form of overloads. Overload detection is automatic, though the actual curve values (usage calculations) must be provided by your software.

If you are already familiar with the curves from the **ResourcesView**, the **LoadsView** offers a more condensed overview of the current resource usage situation.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <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>
```

```

.tooltip-header {
  padding: 2px 6px;
  background-color: #ccc;
  color: #444;
  font-size: 13px;
  font-weight: bold;
}
</style>
</head>
<body>
  <div id="toolbar">
    <input style="margin-left: 20px;" id="StackLoadCurves" type="checkbox"/>
    <label for="StackLoadCurves">Stack load curves</label>
    <input style="margin-left: 20px;" id="ShowGridLines" type="checkbox"/>
    <label for="ShowGridLines">Show grid lines</label>
    <input style="margin-left: 20px;" id="CurvePanelsResizable" type="checkbox"/>
    <label for="CurvePanelsResizable">Curve panels resizable</label>
    <input style="margin-left: 20px;" id="ShowOverloadAsPattern" type="checkbox"/>
    <label for="ShowOverloadAsPattern">Show overload as pattern</label>
  </div>
  <div id="VSWidget"></div>
  <script>
    Miscellaneous.ready(() => {
      const { ViewType } = netronic.nVSW;

      document.getElementById('StackLoadCurves').addEventListener('click', checkStatus);
      document.getElementById('CurvePanelsResizable').addEventListener('click', checkStatus);
      document.getElementById('ShowGridLines').addEventListener('click', checkStatus);
      document.getElementById('ShowOverloadAsPattern').addEventListener('click', checkStatus);

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

      const options = {
        licenseKey: window.VSW_LICENSE_KEY,
        viewType: ViewType.Loads,
        start: timeAreaStart,
        end: timeAreaEnd,
        curvePanelsResizable : false,
        linesShownInLoadCurvePanels : false,
        defaultResourceRowTooltipTemplateID: "TooltipForResource",
        defaultResourceCurveTooltipTemplateID: "TooltipForCurve",
        defaultResourceRowSelectable: false,
        multipleSelectionEnabled: false,
        fixedTableColumnWidth: 0,
        defaultResourceLoadCurvePaneColor: "ivory",
        datelineGridColor: "#A7A7A7",
        patternShownOnOverloadCurves: false,

        onCurvePaneResized: handleOnCurvePaneResized,
      };

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

      const resources = createResources();
      const tooltips = createTooltipTemplates();
      const curves = createCurves();
      vsWidget.addCurves(curves);
      vsWidget.addResources(resources);
      vsWidget.addTooltipTemplates(tooltips);
      vsWidget.fitTimeAreaIntoView();
      vsWidget.render();

      function createCurves() {
        const noOfCurves = 4;
        const scaleMaximumValues = [22, 10, 10, 6];
        const fillColors = ["lightgrey", "#77C2D6", "#F5CF79", "#6ED96E"];

        const capacityCurves = [];
        for (let i = 0; i < noOfCurves; i++) {
          capacityCurves.push({
            ID: `C${i}`,
            Type: 0,
            FillColor: "white",
          });
        }
      }
    });
  </script>

```

```

        StrokeColor: "darkgrey",
        CurvePointEntries: [],
        ScaleMaximumValue: scaleMaximumValues[i % scaleMaximumValues.length],
    });
}

const loadCurves = [];
for (let i = 0; i < noOfCurves; i++) {
    loadCurves.push({
        ID: `L${i}`,
        Type: 0,
        FillColor: fillColors[i % fillColors.length],
        StrokeColor: fillColors[i % fillColors.length],
        CurvePointEntries: [],
        OverloadColor: "red",
    });
}

// special case: L10
const loadCurve10 = {
    ID: "L10",
    Type: 3,
    FillColor: "grey",
    StrokeColor: "transparent",
    CurveIDs: loadCurves.slice(1).map(curve => curve.ID).reverse(),
    OverloadColor: "salmon",
};

const randFromArray = xorshift32Array(12345);

for (let date = new Date(timeAreaStart); date <= timeAreaEnd; date.setDate(date.getDate() + 1)) {
    const currentDate = date.toISOString().slice(0, 10);
    const weekday = date.getDay() + 1;

    let sumCapacity = 0;
    let sumLoad = 0;

    for (let i = 0; i < 3; i++) {
        const capacity = weekday > 5 ? 0 : randFromArray(i==2 ? [4, 4, 4, 3] : [8, 8, 8, 7]);
        const load = weekday > 5 ? 0 : randFromArray(i==2 ? [2, 3, 4] : [4, 5, 6, 7, 8]);

        capacityCurves[i+1].CurvePointEntries.push({ PointInTime: currentDate, Value: capacity });
        loadCurves[i+1].CurvePointEntries.push({ PointInTime: currentDate, Value: load });

        sumCapacity += capacity;
        sumLoad += load;
    }
    capacityCurves[0].CurvePointEntries.push({ PointInTime: currentDate, Value: sumCapacity});
    loadCurves[0].CurvePointEntries.push({ PointInTime: currentDate, Value: sumLoad});
}
return [...capacityCurves, ...loadCurves, loadCurve10];
}

function createResources() {
    const resources = [];

    for (let i = 0; i < 4; i++) {
        const resource = {
            ID: `Res${i}`,
            TableText: `Resource ${i}`,
            CapacityCurveID: `C${i}`,
            LoadCurveID: `L${i}`,
            LoadCurvePaneHeight: [160, 80, 80, 48][i],
        };
        if (i !== 0)
            resource.ParentID = resources[0].ID;
        else
            resource.TableText = "Resource Group";

        resources.push(resource);
    }

    return resources;
}

```

```

function checkStatus() {
  const stackedFlag = document.getElementById("StackLoadCurves").checked;
  const resizeFlag = document.getElementById("CurvePanelsResizable").checked;
  const gridLinesFlag = document.getElementById("ShowGridLines").checked;
  const overloadAsPattern = document.getElementById("ShowOverloadAsPattern").checked;

  resources[0].LoadCurveID = stackedFlag ? "L10" : "L0";

  vsWidget.option("curvePanelsResizable", resizeFlag);
  vsWidget.option("linesShownInLoadCurvePanels", gridLinesFlag);
  vsWidget.option("patternShownOnOverloadCurves", overloadAsPattern);

  vsWidget.updateResources([resources[0]]);
  vsWidget.render();
}

function createTooltipTemplates() {
  const tooltipTemplates = [
    {
      ID: "TooltipForResource",
      //IsInteractive: true,
      HTMLFormat: `
        <div style="max-height: 120px; overflow:auto;">
          <table class="table-container">
            <tr>
              <td class="tooltip-header" colspan="2">Resource</td>
            </tr>
            <tr>
              <td class="table-rowHeader">ID</td>
              <td class="table-data">{{ID}}</td>
            </tr>
            <tr>
              <td class="table-rowHeader">Name</td>
              <td class="table-data">{{TableText}}</td>
            </tr>
          </table>
        </div>`
    },
    {
      ID: "TooltipForCurve",
      IsInteractive: true,
      HTMLFormat: `
        <table>
          <tr>
            <td><b>Resource</b></td>
          </tr>
          <tr>
            <td>Name:</td>
            <td>{{TableText}}</td>
          </tr>
          <tr>
            <td>Date:</td>
            <td>{{#Date}}</td>
          </tr>
          <tr>
            <td>Capacity:</td>
            <td>{{#Capacity}}</td>
          </tr>
          <tr>
            <td>Load:</td>
            <td>{{#Load}}</td>
          </tr>
        </table>`
    }
  ];

  return tooltipTemplates;
}

// PseudoRandomNumberGenerator for creating curve values
function xorshift32Array(seed) {
  let x = seed >>> 0;
  return function(arr) {

```

```

        const len = arr.length;
        if (len === 0)
            return undefined; // falls leeres Array
        x ^= x << 13;
        x ^= x >>> 17;
        x ^= x << 5;
        const rand = (x >>> 0) / 4294967296;
        const idx = Math.floor(rand * len);
        return arr[idx];
    };
}

function handleOnCurvePaneResized(args) {
    const resource = args.object;
    resource.LoadCurvePaneHeight = args.newHeight;
    vsWidget.updateResources([resource]);
    vsWidget.render();
}
});
</script>
</body>
</html>

```