

12-02 Activities view – Interactions – Moving rows via keyboard

This example shows how moving a row using the keys changes the position of the associated subtree.

In example **01-02 ActivitiesView | Dragging rows vertically**, all the processing of the row changes is done internally which makes it very easy to use.

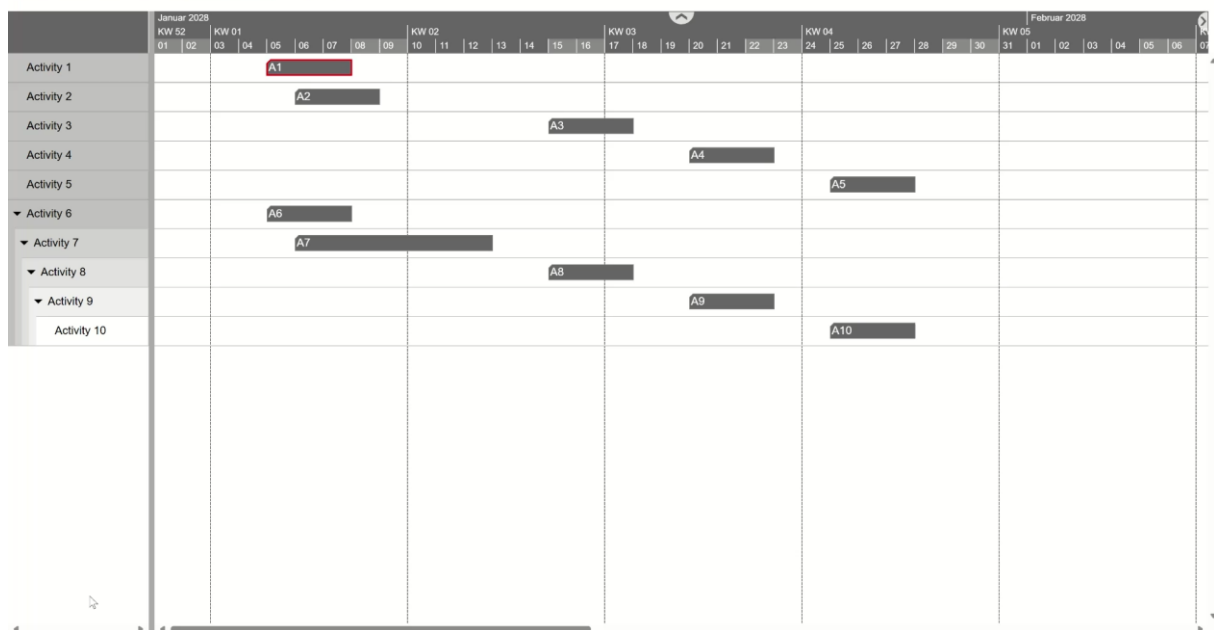
In cases where behavioural adjustments are required, they must be made explicit. This can be done in several ways.

In this example, we assume that we always want our array of activities to be in sync with visualisation.

We have defined four support functions for this purpose:

1. **moveRow()** moves the selected subtree to the desired position in the array.
2. **rowOrderSort()** ensures that the order of the activities in the array always matches the order in the visualisation.
3. **calculateLevels()** determines the hierarchy level of all activities.
4. **isChildParentOf()** checks whether a child belongs to a specific parent.

01-03 ActivitiesView - Move rows vertically using keys



Click on the bar for the row you want to move, or use the Tab key to move forward or Shift+Tab to move backward. A red frame will appear around the selected bar.

The **Up** and **Down** keys allow you to move sub-trees within the same parent tree.

You can use the **Shift+Up** and **Shift+Down** keys to step through all hierarchy levels. This means that the subtree is always hooked onto the corresponding level.

The **Alt+Up** and **Alt+Down** keys allow you to move subtrees at the same level, regardless of which parent they belong to.

The + and - keys allow you to change the hierarchy level of a subtree by increasing or decreasing it.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>01-03 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/12-02_ActivitiesView-Interactions-Moving_rows_via_keyboard.pdf"
        target="_blank">12-02 Activities view - Interactions - Moving rows via keyboard</a>
    </h3>
  </div>
  <div id="VSWidget"></div>
  <script>
    Miscellaneous.ready(() => {
      const { BarDragModes, ObjectType, RowDesigns } = netronic.nVSW;

      document.addEventListener("keydown", onKeydown);

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

      const options = {
        licenseKey: window.VSW_LICENSE_KEY,

        start: timeAreaStart,
        end: timeAreaEnd,

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

        ignoreCalendarOnActivityBarInteractions: true,
        defaultActivityAllowedBarDragModes: BarDragModes.None,
        defaultActivityBarSelectable: false,
        defaultActivityRowSelectable: false,
        multipleSelectionEnabled: false,

        onClicked: (args) => {
          if (args.objectType === ObjectType.Activity)
            markActivity(args.object.ID);
        },

        onCollapseStateChanged: function(args) {
          args.object.CollapseState = args.newCollapseState;
        }
      };

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

      let activities = createActivities();
      rowOrderSort();
      vsWidget.addActivities(activities);

      let currentActivityID = activities[0].ID;
      markActivity(currentActivityID);
    });
  </script>
</body>
</html>
```

```

vsWidget.render();

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",
      TableText: "Activity 6",
      BarText: "A6",
      Start: new Date(2028, 0, 5),
      my_WorkingDays: 3,
      ParentID: "",
    },
    {
      ID: "Act7",
      TableText: "Activity 7",
      BarText: "A7",
      Start: new Date(2028, 0, 6),
      my_WorkingDays: 7,
      ParentID: "Act6",
    },
    {
      ID: "Act8",
      TableText: "Activity 8",
      BarText: "A8",
      Start: new Date(2028, 0, 15),
      my_WorkingDays: 3,
      ParentID: "Act7",
    },
    {
      ID: "Act9",
      TableText: "Activity 9",
      BarText: "A9",
      Start: new Date(2028, 0, 20),
    }
  ]
}

```

```

        my_WorkingDays: 3,
        ParentID: "Act8",
    },
    {
        ID: "Act10",
        TableText: "Activity 10",
        BarText: "A10",
        Start: new Date(2028, 0, 25),
        my_WorkingDays: 3,
        ParentID: "Act9",
    },
];

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

return activities;
}

function moveRow(fromRowIndex, toRowIndex) {
    if (fromRowIndex < 0 || fromRowIndex >= activities.length || fromRowIndex === toRowIndex)
        return;

    if (toRowIndex < 1) {
        const tempObject = activities[fromRowIndex];
        activities.splice(fromRowIndex, 1);
        activities.unshift(tempObject);
    }
    else if (toRowIndex >= activities.length) {
        const tempObject = activities[fromRowIndex];
        activities.splice(fromRowIndex, 1);
        activities.push(tempObject);
    }
    else {
        const element = activities.splice(fromRowIndex, 1)[0];
        activities.splice(toRowIndex, 0, element);
    }
}

function rowOrderSort() {
    let parents = activities.filter((item) => !item.ParentID);
    let children = activities.filter((item) => !!item.ParentID);

    while (children.length > 0) {
        for (let i = 0; i < children.length; i++) {
            let current = children[i];
            let indexParent = parents.findIndex((item) => item.ID === current.ParentID);

            if (indexParent > -1) {
                for (let j = indexParent + 1; j <= parents.length; j++) {
                    if (j === parents.length) {
                        parents.push(current);
                        children.splice(i, 1);
                        break;
                    }
                    else if (parents[j].ParentID !== current.ParentID) {
                        parents.splice(j, 0, current);
                        children.splice(i, 1);
                        break;
                    }
                }
            }
            break;
        }
    }

    for (let i = 0; i < activities.length; i++)
        activities[i] = parents[i];
}

function onKeydown(event) {
    let fromIndex = activities.findIndex((item) => item.ID === currentActivityID);

```

```

if (fromIndex < 0) return;

switch (event.key) {
  case "Tab": {
    event.preventDefault();

    document.getElementById("VSWidget").focus();

    let newActivityID = currentActivityID;
    if (event.shiftKey)
      newActivityID = `Act${modulo10N(+currentActivityID.slice(3) - 1, 10)}`;
    else
      newActivityID = `Act${modulo10N(+currentActivityID.slice(3) + 1, 10)}`;

    markActivity(newActivityID);
    break;
  }

  case "+": {
    event.preventDefault();

    const currentParentID = activities[fromIndex].ParentID;
    for (let i = fromIndex - 1; i >= 0; i--) {
      if (activities[i].ParentID === currentParentID) {
        activities[fromIndex].ParentID = activities[i].ID;
        vsWidget.updateActivities([activities[fromIndex]]);
        vsWidget.render();
        break;
      }
    }
    break;
  }

  case "-": {
    event.preventDefault();

    if (activities[fromIndex].ParentID) {
      const parentActivity = activities.find((item) => item.ID ===
        activities[fromIndex].ParentID);
      activities[fromIndex].ParentID = parentActivity.ParentID;
      vsWidget.removeActivities(activities);
      vsWidget.addActivities(activities);
      vsWidget.render();
    }
    break;
  }

  case "ArrowUp": {
    event.preventDefault();

    if (event.altKey) {
      const levels = calculateLevels(activities);
      const currentLevel = levels[fromIndex];
      for (let toIndex = fromIndex - 1; toIndex > -1; toIndex--) {
        if (levels[toIndex] === currentLevel) {
          activities[fromIndex].ParentID = activities[toIndex].ParentID;
          if (toIndex - fromIndex > 1)
            toIndex--;
          moveRow(fromIndex, toIndex);
          break;
        }
      }
    }
    else if (event.shiftKey) {
      let toIndex = fromIndex - 1;
      if (fromIndex > 0 && activities[fromIndex].ParentID ===
        activities[toIndex].ID)
        toIndex--;
      activities[fromIndex].ParentID = toIndex < 0 ? "" : activities[toIndex].ID;
      moveRow(fromIndex, toIndex);
    }
    else {
      for (let toIndex = fromIndex - 1; toIndex >= 0; toIndex--) {
        if (activities[toIndex].ParentID === activities[fromIndex].ParentID) {
          moveRow(fromIndex, toIndex);
          break;
        }
      }
    }
  }
}

```

```

    }
  }
}

rowOrderSort();
vsWidget.removeActivities(activities);
vsWidget.addActivities(activities);
vsWidget.render();
break;
}

case "ArrowDown": {
  event.preventDefault();

  if (event.altKey) {
    const levels = calculateLevels(activities);
    const currentLevel = levels[fromIndex];
    for (let toIndex = fromIndex + 1; toIndex < activities.length; toIndex++) {
      if (levels[toIndex] === currentLevel && !isDescendantOfAncestor(activities[toIndex] -
1],
                                activities[fromIndex])) {
        activities[fromIndex].ParentID = activities[toIndex].ParentID;
        if (toIndex - fromIndex > 1)
          toIndex--;
        moveRow(fromIndex, toIndex);
        break;
      }
      if (toIndex === activities.length - 1 && currentLevel === 0 && levels[toIndex] > 0) {
        moveRow(fromIndex, toIndex);
        break;
      }
    }
  }
  else if (event.shiftKey) {
    let loopWithoutBreak = true;
    for (let toIndex = fromIndex + 1; toIndex < activities.length; toIndex++) {
      if (activities[fromIndex].ParentID === activities[toIndex].ParentID ||
          !activities[toIndex].ParentID) {
        activities[fromIndex].ParentID = toIndex >= activities.length ? "" :
          activities[toIndex].ID;

        moveRow(fromIndex, toIndex);
        loopWithoutBreak = false;
        break;
      }
    }
    if (loopWithoutBreak)
      activities[fromIndex].ParentID = "";
  }
  else {
    for (let toIndex = fromIndex + 1; toIndex < activities.length; toIndex++) {
      if (activities[toIndex].ParentID === activities[fromIndex].ParentID) {
        moveRow(fromIndex, toIndex);
        break;
      }
    }
  }

  rowOrderSort();
  vsWidget.removeActivities(activities);
  vsWidget.addActivities(activities);
  vsWidget.render();
}
}

function calculateLevels() {
  const levels = new Array(activities.length).fill(0);
  for (let i = 0; i < activities.length; i++) {
    let current = activities[i];
    while (current.ParentID) {
      levels[i]++;
      current = activities.find((activity) => activity.ID === current.ParentID);
    }
  }
}

```

```

    return levels;
}

function isDescendantOfAncestor(descendant, ancestor) {
    let current = descendant;
    while (current.ParentID) {
        if (current.ParentID === ancestor.ID)
            return true;

        current = activities.find((activity) => activity.ID === current.ParentID);
    }

    return false;
}

function markActivity(newActivityID) {
    let activity = activities.find((activity) => activity.ID === currentActivityID);
    if (activity) {
        delete activity.BorderColor;
        delete activity.BorderWidth;
        currentActivityID = "";
        vsWidget.updateActivities([activity]);
        vsWidget.render();
    }

    activity = activities.find((activity) => activity.ID === newActivityID);
    if (activity) {
        currentActivityID = activity.ID;
        activity.BorderColor = "red";
        activity.BorderWidth = 2;
        currentActivityID = newActivityID;
        vsWidget.updateActivities([activity]);
        vsWidget.render();
    }
}

function modulo10toN(value, n) {
    return (((value - 1) % n) + n) % n + 1;
}
});
</script>
</body>
</html>

```