Tutorial 05 - Synchronizing React Charts with Data in a Group

In the previous tutorials we showed you how to use `<SciChartReact/>` and `<SciChartGroup/>` to dynamically add and remove charts to a group in React.

As a basis for this tutorial, use Tutorial 04 - Adding & Removing Charts. We're going to build directly on top of this code, so go ahead and copy to a new project folder.

Also, for the project setup with webpack and npm, check out Tutorial 01 - Understanding the scichart-react boilerplate which explains how to start from scratch!

Refactoring the App.jsx Code

Starting with the App.jsx code from Tutorial 04, let's begin by refactoring the code. We want to extract the chart initialization into a separate code file, as well as create a data manager to provide data to the chart on initialization.

Creating a DataManager Class

Add a new file to `src/RandomWalkGenerator.js` and paste in the following code. This is a JavaScript version of the RandomWalkGenerator.ts from Github which simply generates a randomised waveform based on some parameters.

RANDOMWALKGENERATOR.JS

```
export class RandomWalkGenerator {
  constructor(bias = 0.01) {
    this.bias = bias;
    this.reset();
  }

  Seed(seed) {
    this._seed = seed % 2147483647;
    if (this._seed <= 0) this._seed += 2147483646;
    return this;
  }

  reset() {
    this.i = 0;
    this.last = 0;
  }

  getRandomWalkSeries(count) {
    const xValues = [];
    const yValues = [];
    const random = () =>
      this._seed === undefined
        ? Math.random()
        : (this.nextSeeded() - 1) / 2147483646;

    for (let i = 0; i < count; i++) {
      const next = this.last + (random() - 0.5 + this.bias);
      xValues.push(this.i++);
      yValues.push(next);
      this.last = next;
    }

    return { xValues, yValues };
  }

  nextSeeded() {
    return (this._seed = (this._seed * 16807) % 2147483647);
  }
}
```

Also we're going to add a `DataManager` class, which will manage and fetch data by ID for charts. This can be extended later when you're working on your projects & apps to fetch data remotely, and it returns a promise to allow for asynchronous requests to your server or database for fetching and caching of data. This simulates a 100ms delay to account for requests and can be extended as you wish.

DATAMANAGER.JS

```
import { RandomWalkGenerator } from "./RandomWalkGenerator";

export class DataManager {
  constructor() {
    this.generators = new Map();
    this.data = new Map();
  }

  async fetchData(id, count = 100) {
    // Create new generator if doesn't exist
    if (!this.generators.has(id)) {
      const generator = new RandomWalkGenerator();
      this.generators.set(id, generator);
    }

    // Get or create data
    if (!this.data.has(id)) {
      const generator = this.generators.get(id);
      const data = generator.getRandomWalkSeries(count);
      this.data.set(id, data);
    }

    // Simulate async operation
    return new Promise((resolve) => {
      setTimeout(() => {
        resolve(this.data.get(id));
      }, 100);
    });
  }

  reset(id) {
    if (this.generators.has(id)) {
      const generator = this.generators.get(id);
      generator.reset();
      this.data.delete(id);
    }
  }

  resetAll() {
    this.generators.forEach((generator) => generator.reset());
    this.data.clear();
  }
}
```

Refactoring App.jsx to load data from DataManager

First we're going to refactor App.jsx to extract the chart initialization logic into a new file: initChart.js. We'll also add a `setData` function and some chart modifiers to provide zooming, panning and rollover tooltip behaviours. Go ahead and remove `simpleChart` fuhction from App.jsx and add a new file called `initChart.js`.

INITCHART.JS

```
import {
  SciChartSurface,
  NumericAxis,
  SplineMountainRenderableSeries,
  ZoomPanModifier,
  MouseWheelZoomModifier,
  ZoomExtentsModifier,
  RolloverModifier,
  XyDataSeries,
  EllipsePointMarker,
  SciChartJsNavyTheme,
  EAutoRange,
  NumberRange,
  FadeAnimation,
} from "scichart";

export const initChart = async (divElement, chartId, chartGroupId) => {
  const { sciChartSurface, wasmContext } = await SciChartSurface.create(
    divElement,
    {
      theme: new SciChartJsNavyTheme(),
      canvasBorder: {
        borderLeft: 1,
        borderTop: 1,
        borderRight: 1,
        borderBottom: 1,
        color: "#eee",
      },
    }
  );

  sciChartSurface.xAxes.add(
    new NumericAxis(wasmContext, {
      axisTitle: `Chart ${chartId}`,
      axisTitleStyle: { fontSize: 12 },
    })
  );
  sciChartSurface.yAxes.add(
    new NumericAxis(wasmContext, {
      axisTitle: "Y Axis",
      axisTitleStyle: { fontSize: 12 },
      autoRange: EAutoRange.Always,
      growBy: new NumberRange(0, 0.1),
    })
  );

  // #region AddModifiers
  // Add modifiers for zoom, pan and tooltip behaviour
  sciChartSurface.chartModifiers.add(
    new ZoomPanModifier(),
    new MouseWheelZoomModifier(),
    new ZoomExtentsModifier(),
    new RolloverModifier({
      modifierGroup: chartGroupId,
      rolloverLineStroke: "LightSteelBlue",
      snapToDataPoint: true,
    })
  );
  // #endregion

  // Add a setData function. This will accept xValues and yValues and update the chart
  // creating a new series each time.
  const setData = (xValues, yValues) => {
    // Clear existing series, deleting memory by passing callDeleteOnChildren: true
    sciChartSurface.renderableSeries.clear(true);

    // Create new series
    const mountainSeries = new SplineMountainRenderableSeries(wasmContext, {
      dataSeries: new XyDataSeries(wasmContext, {
        xValues,
        yValues,
      }),
      pointMarker: new EllipsePointMarker(wasmContext, {
        fill: "SteelBlue",
        stroke: "White",
      }),
      animation: new FadeAnimation({ duration: 500 }),
    });

    // Setup series rollovermodifier properties
    mountainSeries.rolloverModifierProps.tooltipTextColor = "#fff";
    mountainSeries.rolloverModifierProps.tooltipColor = "SteelBlue";
    mountainSeries.rolloverModifierProps.tooltipLabelX = "X";
    mountainSeries.rolloverModifierProps.tooltipLabelY = "Y";

    sciChartSurface.renderableSeries.add(mountainSeries);
    sciChartSurface.zoomExtents();
  };

  // return values from initChart() go into initResult in <SciChartReact onInit={initResult
  return { sciChartSurface, setData };
};
```

Note that here we've extended the previous chart initialization function with some extra behaviours:

- We now pass in `chartId` as well as `chartGroupId` to the function.
- We create a chart with a border, so we can differentiate between different charts in the group.
- We remove the `chartTitle`, instead setting the title on the xAxis to save space.
- We set `yAxis.autoRange = EAutoRange.Always`. This will be important later.
- We add some chart modifiers to provide zooming, panning and tooltip behaviour. `modifierGroup` is passed to the `RolloverModifier` to share mouse events across charts in

the same group.
- We create and return a `setData` function, which adds a series to the chart

Next, update the App.jsx to use `DataManager`. Using `<SciChartReact onInit/>` we can fetch data asynchronously then call `setData` to add a series to the chart.

> Note that `onInit` cannot be an async function, so we use `fetch().then()` in order to set the data on the chart.

APP.JSX

```
import React, { useState, useRef } from "react";
import "./styles.css";
import { SciChartGroup, SciChartReact } from "scichart-react";
import { initChart } from "./initChart";
import { DataManager } from "./DataManager";
function App() {
  const [charts, setCharts] = useState([0, 1]); // Initialize with 2 charts
  // Create a DataManager class to proxy fetching data
  const [dataManager] = useState(() => new DataManager());
  const addChart = () => {
    setCharts([...charts, charts.length]);
  };
  const removeChart = () => {
    if (charts.length > 0) {
      setCharts(charts.slice(0, -1));
    }
  };
  return (
    <div className="App">
      <header className="App-header">
        <h1>&lt;SciChartReact/&gt; chart groups</h1>
      </header>
      <div
        style={{
          display: "flex",
          justifyContent: "left",
          backgroundColor: "lightgrey",
          padding: "10px",
        }}
      >
        <button onClick={addChart} style={{ margin: "0 10px" }}>
          Add Chart
        </button>
        <button onClick={removeChart} style={{ margin: "0 10px" }}>
          Remove Chart
        </button>
      </div>
      <div style={{ height: "600px" }}>
        <SciChartGroup>
```

```
          {charts.map((chartId) => (
            <SciChartReact
              key={chartId}
              initChart={(div) => initChart(div, chartId, "chartGroupId")}
              // After initialization, fetch data and call setData on the chart (see initCha
              // onInit cannot be an async function, so use dataManager.fetchData().then() t
              onInit={(initResult) => {
                dataManager.fetchData(chartId).then((data) => {
                  initResult.setData(data.xValues, data.yValues);
                });
              }}
              style={{ height: `${100 / charts.length}%` }}
            />
          ))}
        </SciChartGroup>
      </div>
    </div>
  );
}
export default App;
```

Synchronizing Zoom, Pan and Tooltip Operations Across Charts

The next step in the tutorial is to synchronize Zoom, Pan and Tooltip operations across charts. To do this we're going to use the `AxisSynchronizer` class found in the SciChart.js demo - Sync Multi Chart example, as well as the *modifierGroup* property found on *ChartModifierBase*.

You'll notice in the `initChart()` function `modifierGroup` is set here:

INITCHART.JS

```
// Add modifiers for zoom, pan and tooltip behaviour
sciChartSurface.chartModifiers.add(
  new ZoomPanModifier(),
  new MouseWheelZoomModifier(),
  new ZoomExtentsModifier(),
  new RolloverModifier({
    modifierGroup: chartGroupId,
    rolloverLineStroke: "LightSteelBlue",
    snapToDataPoint: true,
  })
);
```

This ensures that mouse events from one ChartModifier are passed to others in the same group. This will partly tooltips across charts but will not synchronize everything - such as axis sizes or *axis.visibleRange*. To do this, we need some further logic, found in

`AxisSynchronizer`.

Go ahead and add a new class file to the project, AxisSynchronizer.js. Add the following code:

AXISSYNCHRONIZER.JS

```
import { EventHandler } from "scichart";

/** A helper class for synchronizing an arbitrary number of axes */
export class AxisSynchroniser {
  constructor(initialRange, axes) {
    this.visibleRange = initialRange;
    this.axes = [];
    this.visibleRangeChanged = new EventHandler();

    this.publishChange = this.publishChange.bind(this);
    if (axes) {
      axes.forEach((a) => this.addAxis(a));
    }
  }

  publishChange(data) {
    this.visibleRange = data.visibleRange;
    this.axes.forEach((a) => (a.visibleRange = this.visibleRange));
    this.visibleRangeChanged.raiseEvent(data);
  }

  addAxis(axis) {
    if (!this.axes.includes(axis)) {
      this.axes.push(axis);
      axis.visibleRange = this.visibleRange;
      axis.visibleRangeChanged.subscribe(this.publishChange);
    }
  }

  removeAxis(axis) {
    const index = this.axes.findIndex((a) => a === axis);
    if (index >= 0) {
      this.axes.splice(index, 1);
      axis.visibleRangeChanged.unsubscribe(this.publishChange);
    }
  }
}
```

Next, we ned to update App.jsx where the `<SciChartReact/>` component is declared to use `onInit` and `onDelete` to add/remove the chart xAxis from the `AxisSynchronizer`. This will ensure that all visibleRanges on the axis are synchronized and that all charts zoom and pan together in the group.

APP.JSX - USING AXISSYNCHRONIZER

```
<SciChartReact
  key={chartId}
  initChart={(div) => initChart(div, chartId, "chartGroupId")}
  // After initialization, fetch data and call setData on the chart (see initChart.js)
  // onInit cannot be an async function, so use dataManager.fetchData().then() to update the
  onInit={(initResult) => {
    dataManager.fetchData(chartId).then((data) => {
      initResult.setData(data.xValues, data.yValues);
    });

    // After init, add the chart to axis synchronizer
    axisSynchronizer.addAxis(
      initResult.sciChartSurface.xAxes.get(0)
    );
  }}
  onDelete={(initResult) => {
    // After delete, remove the chart from axis synchronizer
    axisSynchronizer.removeAxis(
      initResult.sciChartSurface.xAxes.get(0)
    );
  }}
  style={{ height: `${100 / charts.length}%` }}
/>
```

Where `axisSynchronizer` is declared at the top of the `App()` React Component as follows:

APP.JSX - DECLARING AXISSYNCHRONIZER

```
import { AxisSynchroniser } from "./AxisSynchronizer";

function App() {
  const [charts, setCharts] = useState([0, 1]); // Initialize with 2 charts
  const [axisSynchronizer, setAxisSynchronizer] = useState(
    new AxisSynchroniser()
  );
```

Here's the completed App.jsx output and initChart function.

APP.JSX - FINAL

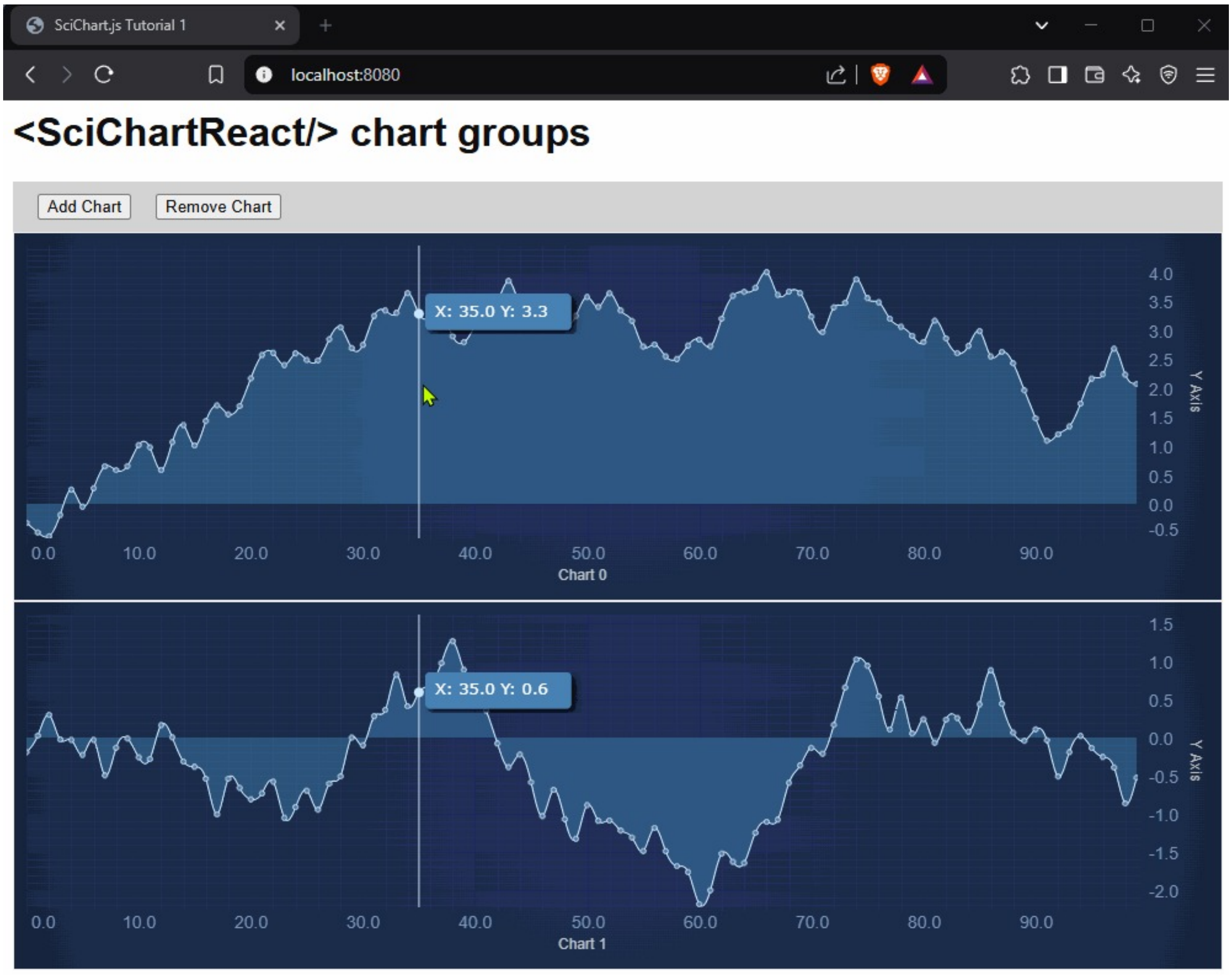
```
import React, { useState, useRef } from "react";
import "./styles.css";
import { SciChartGroup, SciChartReact } from "scichart-react";
import { initChart } from "./initChart";
import { DataManager } from "./DataManager";
// #region AxisSynchronizer
import { AxisSynchroniser } from "./AxisSynchronizer";
```

```
function App() {
  const [charts, setCharts] = useState([0, 1]); // Initialize with 2 charts
  const [axisSynchronizer, setAxisSynchronizer] = useState(
    new AxisSynchroniser()
  );
  // #endregion

  // Create a DataManager class to proxy fetching data
  const [dataManager] = useState(() => new DataManager());

  const addChart = () => {
    setCharts([...charts, charts.length]);
  };

  const removeChart = () => {
    if (charts.length > 0) {
      setCharts(charts.slice(0, -1));
    }
  };

  return (
    <div className="App">
      <header className="App-header">
        <h1>&lt;SciChartReact/&gt; chart groups</h1>
      </header>
      <div
        style={{
          display: "flex",
          justifyContent: "left",
          backgroundColor: "lightgrey",
          padding: "10px",
        }}
      >
        <button onClick={addChart} style={{ margin: "0 10px" }}>
          Add Chart
        </button>
        <button onClick={removeChart} style={{ margin: "0 10px" }}>
          Remove Chart
        </button>
      </div>
      <div style={{ height: "600px" }}>
        <SciChartGroup>
          {charts.map((chartId) => (
            // #region SciChartReact Component
            <SciChartReact
              key={chartId}
              initChart={(div) => initChart(div, chartId, "chartGroupId")}
              // After initialization, fetch data and call setData on the chart (see initCha
              // onInit cannot be an async function, so use dataManager.fetchData().then() t
              onInit={(initResult) => {
                dataManager.fetchData(chartId).then((data) => {
                  initResult.setData(data.xValues, data.yValues);
                });

                // After init, add the chart to axis synchronizer
                axisSynchronizer.addAxis(
                  initResult.sciChartSurface.xAxes.get(0)
                );
              }}
              onDelete={(initResult) => {
                // After delete, remove the chart from axis synchronizer
                axisSynchronizer.removeAxis(
                  initResult.sciChartSurface.xAxes.get(0)
                );
              }}
              style={{ height: `${100 / charts.length}%` }}
            />
            // #endregion
          ))}
        </SciChartGroup>
      </div>
    </div>
  );
}

export default App;
```

INITCHART.JS - FINAL

```
import {
  SciChartSurface,
  NumericAxis,
  SplineMountainRenderableSeries,
  ZoomPanModifier,
  MouseWheelZoomModifier,
  ZoomExtentsModifier,
  RolloverModifier,
  XyDataSeries,
  EllipsePointMarker,
  SciChartJsNavyTheme,
  EAutoRange,
  NumberRange,
  FadeAnimation,
} from "scichart";

export const initChart = async (divElement, chartId, chartGroupId) => {
  const { sciChartSurface, wasmContext } = await SciChartSurface.create(
    divElement,
    {
      theme: new SciChartJsNavyTheme(),
      canvasBorder: {
        borderLeft: 1,
        borderTop: 1,
        borderRight: 1,
        borderBottom: 1,
        color: "#eee",
      },
    }
  );

  sciChartSurface.xAxes.add(
    new NumericAxis(wasmContext, {
      axisTitle: `Chart ${chartId}`,
      axisTitleStyle: { fontSize: 12 },
    })
  );
  sciChartSurface.yAxes.add(
    new NumericAxis(wasmContext, {
      axisTitle: "Y Axis",
      axisTitleStyle: { fontSize: 12 },
      autoRange: EAutoRange.Always,
      growBy: new NumberRange(0, 0.1),
    })
  );

  // #region AddModifiers
  // Add modifiers for zoom, pan and tooltip behaviour
  sciChartSurface.chartModifiers.add(
    new ZoomPanModifier(),
    new MouseWheelZoomModifier(),
    new ZoomExtentsModifier(),
    new RolloverModifier({
      modifierGroup: chartGroupId,
      rolloverLineStroke: "LightSteelBlue",
      snapToDataPoint: true,
    })
  );
  // #endregion

  // Add a setData function. This will accept xValues and yValues and update the chart
  // creating a new series each time.
  const setData = (xValues, yValues) => {
    // Clear existing series, deleting memory by passing callDeleteOnChildren: true
    sciChartSurface.renderableSeries.clear(true);

    // Create new series
    const mountainSeries = new SplineMountainRenderableSeries(wasmContext, {
      dataSeries: new XyDataSeries(wasmContext, {
        xValues,
        yValues,
      }),
      pointMarker: new EllipsePointMarker(wasmContext, {
        fill: "SteelBlue",
        stroke: "White",
      }),
      animation: new FadeAnimation({ duration: 500 }),
    });

    // Setup series rollovermodifier properties
    mountainSeries.rolloverModifierProps.tooltipTextColor = "#fff";
    mountainSeries.rolloverModifierProps.tooltipColor = "SteelBlue";
    mountainSeries.rolloverModifierProps.tooltipLabelX = "X";
    mountainSeries.rolloverModifierProps.tooltipLabelY = "Y";

    sciChartSurface.renderableSeries.add(mountainSeries);
    sciChartSurface.zoomExtents();
  };

  // return values from initChart() go into initResult in <SciChartReact onInit={initResult
  return { sciChartSurface, setData };
};
```

And here's the final output of the tutorial:



Click the Add Chart, Remove Chart button. Now zoom on the chart by dragging the mouse. Use the mouse-wheel to zoom and double-click to reset zoom. All charts should be synchronized, tooltips should track the same x-Value across charts, and zooming/panning in the x-direction should match.

You can get the full source code for this tutorial over at scichart.js.examples on

Github under the folder Tutorials/React/Tutorial_05_Synchronizing_React_Charts