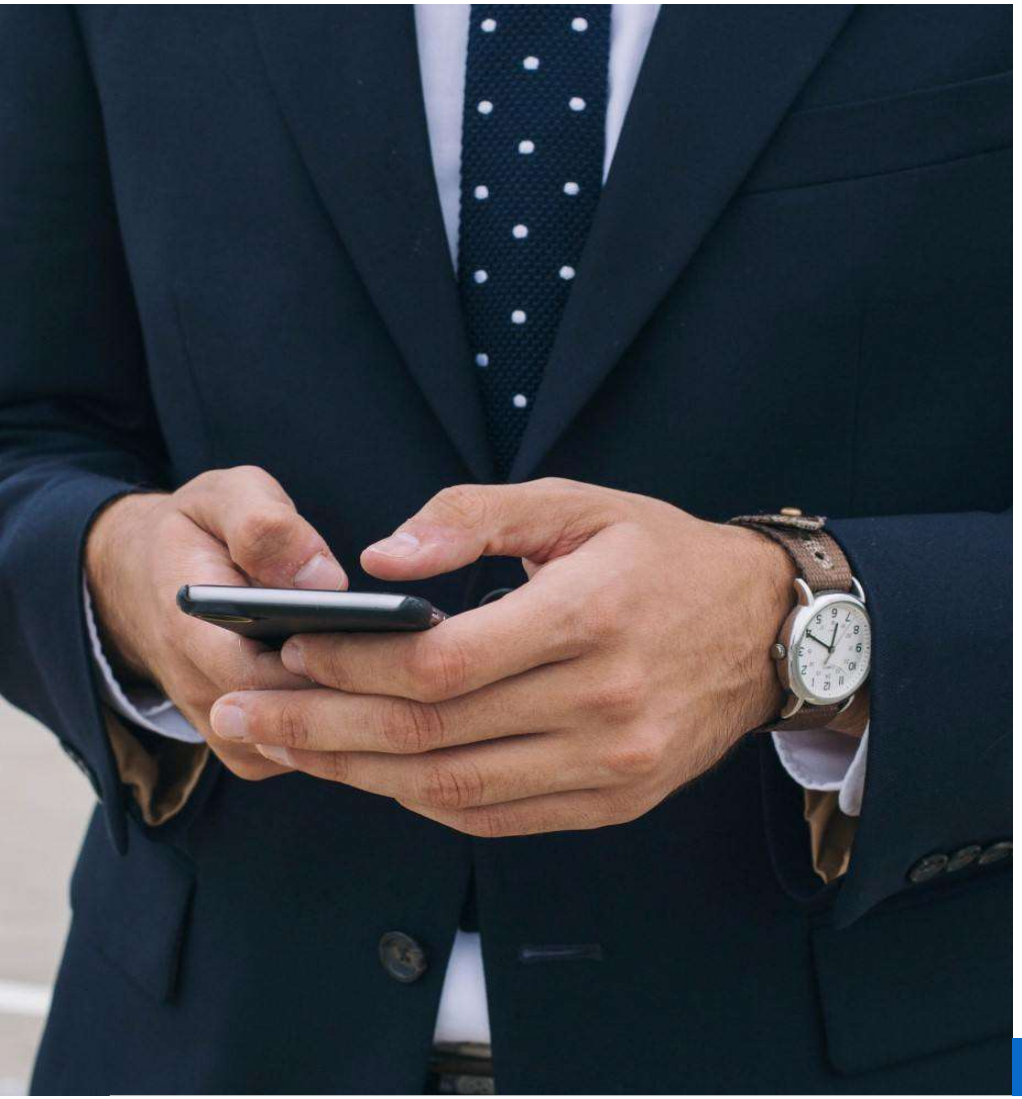


# ServiceNow
# NOW Experience Component Development

Tutorial # 1

Setup Experience Component Development Environment

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Tutorial Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 1: Setup Experience Component Dev Environment

**We will install three pieces of software 1) node.js & npm 2) Microsoft IDE Visual Studio Code 3) NOW Command level Interface now-cli**

### Step 1: Install Node.js & Node Package Manager in one shot

- Go to https://nodejs.org/en/download
- Download the software and install it
- Go to Command prompt and check for node.js

```
C:>node --version
```

- Go to Command prompt and check for Node Package Manager

```
C:>npm --version
```

# Tutorial 1: Setup Experience Component Dev Environment

### Step2 : Install Visual Studio Code ( IDE )

- Go to https://code.visualstudio.com/docs/setup/windows
- Download the software and install it
- Invoke Visual Studio code from Windows menu
- Make the default Shell as **bash**
  - Open **Visual Studio Code**
  - Open the command palette using **View -> Command Palette** or use short cut **Ctrl + Shift +** P .
  - Type **Default Shell**
  - Select Command **Terminal : Select Default Shell**
  - Select **Git Bash** from the options.
  - Open Terminal within Visual Studio code using **View -> Terminal**
  - The terminal window will show **bash** as the default shell.

# Tutorial 1: Setup Experience Component Dev Environment

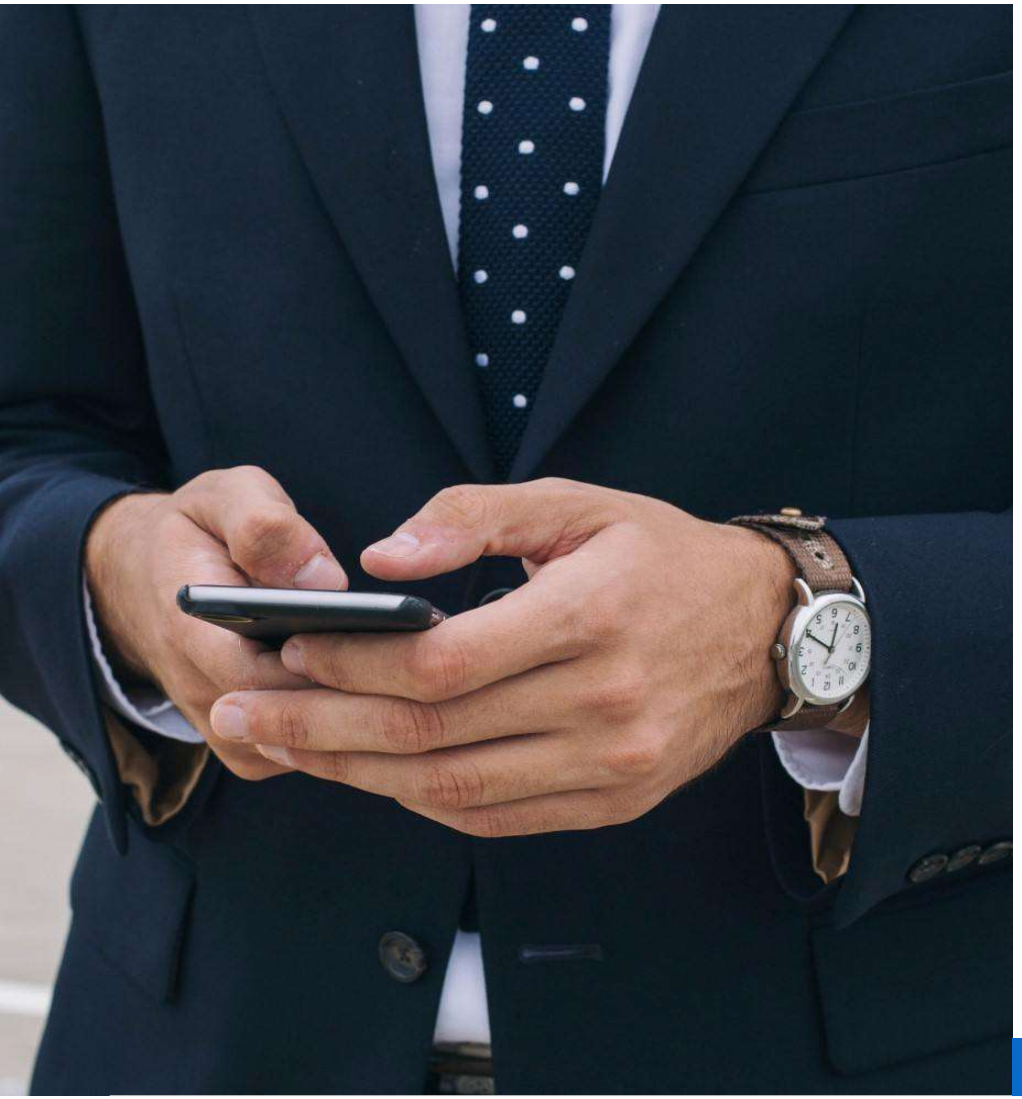
### Step 3: Install now-cli

- now-cli is “ServiceNow Command Level Interface (CLI)”
- It is available for installation using node package manager (npm) which we have already installed
- Invoke Windows command level interface cmd

```
C:>npm install --global @servicenow/cli@paris
```

- Check that now-cli is installed properly

```
C:>now-cli --version
```



# ServiceNow
# NOW Experience Component Development Series

Tutorial # 2

Create your first “Hello World” component

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Tutorial Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 2: Create “Hello World” NOW Experience Component

**Create an empty folder for the project -> Use Visual Studio IDE and open the empty folder -> In Terminal window of IDE Login to SN -> now-cli project -> npm install -> Add your code -> Save All -> npm install -> now-cli develop -> now-cli deploy -> Open Agent Workspace definition -> Open UI Builder -> Add component to the landing page**

## Step 1: Create an empty folder for your project and open the folder in Microsoft Visual Studio IDE

- Create a new directory or folder under one of your directories. Let us say we create a new empty folder named helloworld in a directory named g:\uex. So our folder is g:\uex\helloworld.
- Invoke Visual Studio code IDE and open the above created empty folder named helloworld within the IDE.
- View -> Terminal to open the Terminal window (this will have Bash Shell)
- g:\uex\helloworld will also be current directory of the Terminal window within the Visual Studio code.

## Step 2: Log into ServiceNow instance from within the Terminal window of Microsoft Visual Studio code

- In the terminal window use the following now-cli’s **login** command to log into your servicenow instance

```
g:\uex\helloworld>now-cli login --host https://ven03813.service-now.com --method basic --username admin --password Secret!
```

# Tutorial 2: Create “Hello World” NOW Experience Component

### Step 3: Create a NOW Project using now-cli from Terminal window within the Visual Studio code

- Create a ServiceNow UEX project for our new component named **rg1a-helloworld** ( **all letters should be small case and there should be at least one dash**) using ServiceNow provided now-cli which we had installed in tutorial #1.

```
g:\uex\helloworld>now-cli project --name rg1a-helloworld  --description ‘UEX rg1a-helloworld’
```

- This command will create the project named rg1-helloworld within our folder helloworld. You will see this as a set of directories and files. This will serve as the **Skeleton Code** and we will build our code using this **Skeleton**.

### Step 4: Install project dependencies – or node packages needed by our newly created project

- Node package manager (npm) will copy required packages (files) into the directory named g:\uex\helloworld\node_modules from the internet. This will be based on whatever is specified using import directives in the .js files. In this case the previous step had created some startup .js files for you.

```
g:\uex\helloworld>npm install
```

# Tutorial 2: Create “Hello World” NOW Experience Component

## Step 5: Use this Skeleton code to make a UEX component that will say “Hello World!....”

- The primary source code file from where the component start is a javascript (.js) file named g:\uex\helloworld\src\xxx-rg1a-helloworld\index.js
- Open this index.js file within the Visual source code. Whatever the function named **view** returns, that is what your component returns as output. So make the view to return a string ‘**Hello World! This is my first ServiceNow component**’
- Note: Do a **“Save All”** as otherwise the >now-cli develop command will get stuck midway and not display the output

```
const view = (state, {updateState}) => {
    return (
        <div>’Hello World! This is my first ServiceNow component’</div>
    );
};
```

## Step 6: Install project dependencies – or node packages needed by our newly created project

- In case you add some import directives within your .js files of the project, the Node package manager (npm) will copy those additional packages into the folder named g:\uex\helloworld\node_modules from the internet. Though, we have not done it in our rg1-helloworld component but as a matter of practice I use ‘**npm install**’ in this step just to be sure that I have all the required packages

```
g:\uex\helloworld >npm install
```

# Tutorial 2: Create “Hello World” NOW Experience Component

**Step 7: Invoke now-cli to develop the component based on our code and also open the component in browser**

- Make sure that you have done “Save All” on the project
- Now we are ready to **compile** all the project files **into a NOW component** and **open** the component within our default browser using just one command as follows

```
g:\uex\helloworld >now-cli develop --open
```

- It will take a few minutes for now-cli to build the component – you will see that in the terminal window. And, finally the default browser will open with a tab saying

**‘Hello World! This is my first ServiceNow component’**

You have your first bare bone Hello World component ready. Let us now take it into our ServiceNow Workspace

# Tutorial 2: Create “Hello World” NOW Experience Component

**Step 8: We will now DEPLOY the newly built component into our ServiceNow instance**

- We will make use of >now-cli command level interface with deploy command to achieve it
- In our project folder we have a file named **now-ui.json** which gives directives to deploy the component into ServiceNow
- It will have a directive with a component name. In our case it will be ‘**xxx-rg1a-helloworld**’
- There is another important file named now-cli.json that contains your SN instance details which are required only if you are not logged in into the instance. We are logged in already in the session so we leave it as it is
- We will make use of **>now-cli** with **deploy** command to deploy component in our logged-in SN instance. We will use **force** option so that it **redeploys** if it is already deployed

```
g:\uex\helloworld>now-cli deploy --force
```

# Tutorial 2: Create “Hello World” NOW Experience Component

**Step 9: Use the deployed component on one of the pages of the workspace within ServiceNow**

- Navigator -> workspace -> Select All Workspaces under Administration -> Select Agent Workspace -> Open it in Edit mode by clicking on the ‘click here’

This record is in the **Agent Workspace** application, but **Global** is the current application. To edit this record click here.

- Before invoking UI Builder, go to the Related Tab and look at Landing Pages Tab. Find which page has the lowest Order value as that page displays on opening the Workspace. Therefore, we will place our new component on this page.
- Press “Open UI Builder” to go to the builder where you can drag and drop components
- Select the landing page. Otherwise, you can create a new landing page from here and set it’s Order field to be lowest so that it is on top of all landing pages for this Agent Workspace.
- Go to a Tab named + Add Component Tab in UI Builder.
- You will see your component xxx-rg1a-helloworld. There is also a component named **Container** to contain other components
- First drag and drop from left pane to right, the **container** component. The container will drop properly only when you see a **Green vertical or horizontal line**. If dropped at a wrong place, you can Trash the component and redrop.

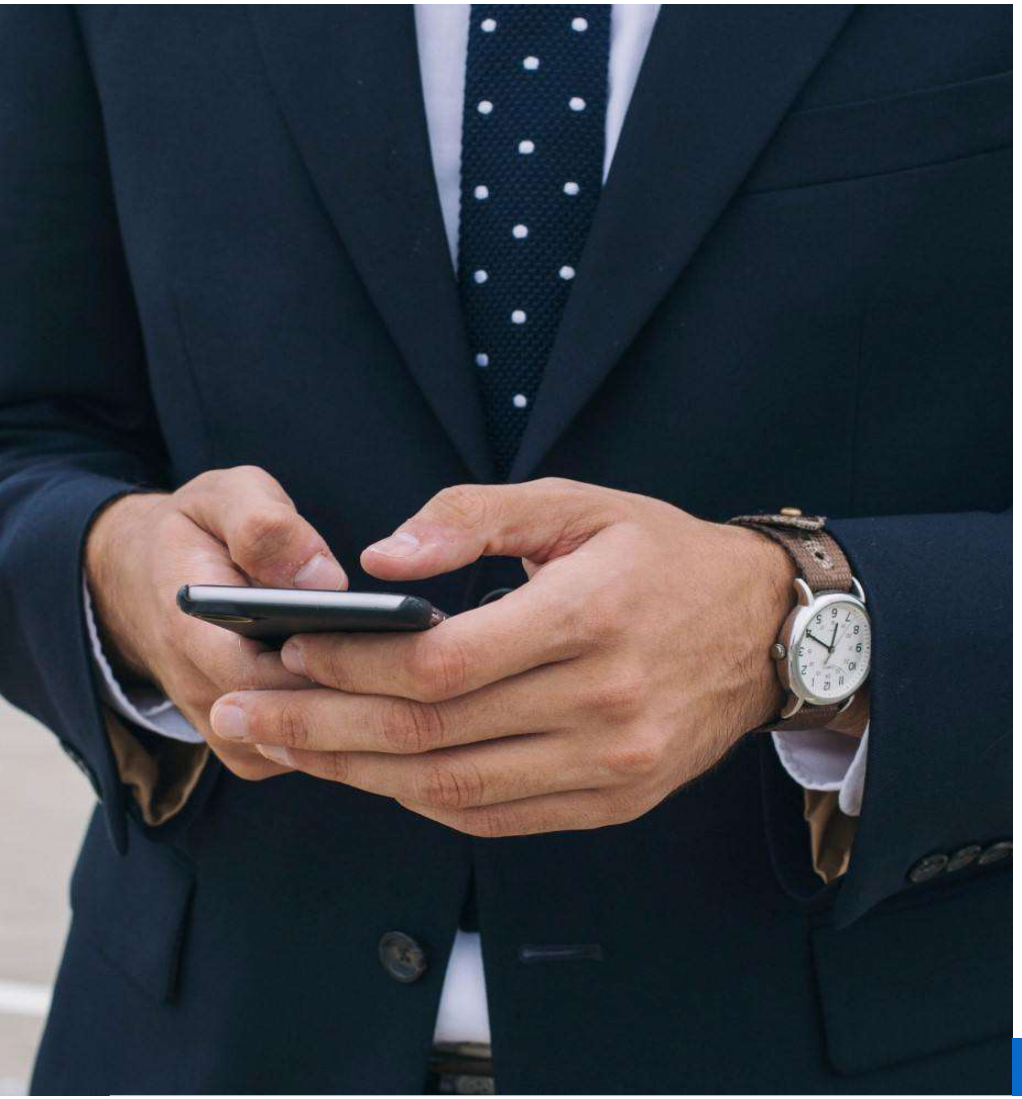
# Tutorial 2: Create “Hello World” NOW Experience Component

## Step 9 Continued: Use the deployed component on one of the pages of the workspace

- Now select your Hello World Component and drag and drop it onto the **Container** that we had placed. In order to sit properly in that container, while dropping the component, the container background should become Green.
- Once you are happy with your placement, do a Save on top right. Make sure that the page is Active.
- Go back to the Agent Workspace definition page
- Use Related Links section to Open the Agent Workspace
- You will see your component with all Excellence on the Agent Workspace landing page

## Important Note regarding Step 7, now-cli develop --open

- In a couple of cases, I noted that now-cli develop --open when given in Visual Studio IDE terminal session gives some unknown Error. I corrected this by creating a new directory and repeating the Steps 2 to Step 7 using Windows DOS prompt window opened by cmd. It seems that error might be because some process holds up some file. This happened only a couple of times and I wanted to make sure that you have this information.



# ServiceNow
# NOW Experience Component Development Series

## Tutorial # 3

ES6 features of JavaScript useful in NOW Component development

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Tutorial Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 1. Arrow Function:

- Arrow functions allow us to write shorter function syntax:
- ()=> instead of function()

```
console.log('Section 1: Arrow Function');

hello1a = function () {
    return "hello World 1a";
}
hello1b = () => {
    return "hello World 1b";
}
console.log(hello1a(), hello1b());
// hello World 1a hello World 1b
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 2. Const

- Const is a block scoped declaration that creates constants. Its value is set at the time of declaration. After the initial value is set, it cannot be changed.

```
console.log('Section 2: const');
{
  const a2=2;
  console.log(a2); //2
  a2 = 3; //type error
}
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 3. Spread Operator … in front of an array

- Spreads values from an array into individual values

```
console.log('Section 3: ... in front of an array - Spread Operator');
var a3 = [2,3,4];
var b3 = [1,...a3,5];
console.log(b3); // [ 1, 2, 3, 4, 5 ]
function foo3(x,y,z){
    console.log(x,y,z);
}
foo3(...[1,2,3]); //1 2 3
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 4. Gather Operator … in front of an array

- The same operator … in front of an array can gather values

```
console.log('Section 4: ... in front of an array - Gather Operator');
function foo4(...args) {
  //  console.log(args);
  return args;
}
console.log(foo4(1,2,3,4,5)); // [ 1, 2, 3, 4, 5 ]
//
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 5. Default parameter values

- Function parameters can now have default values. The default values can be simple values, any valid expression or even a function call. If these are expressions, these are lazily evaluated – that means they are evaluated only when needed.

```
console.log('Section 5: Default Parameter value');
function foo5(x=11, y=31) {
    return x+y;
}
console.log(foo5()); //42
console.log(foo5(5,6)); //11
console.log(foo5(5)); //36
console.log(foo5(5,undefined)); //36 as undefined means y=default 31 is used
console.log(foo5(5,null)); //5 as null coerced to 0
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 6. Default parameter values as expressions

- As mentioned earlier the default values can also be any valid expression or even a function call. If these are expressions, these are lazily evaluated – that means they are evaluated only when needed.

```
console.log('Section 6: Default Parameter value as expression');
function bar6(val6){
    console.log('bar6 called');
    return y6 + val6;
}
function foo6 (x6 = y6+3, z6 = bar6(x6)){
    console.log(x6, z6);
}
var y6 = 5;
foo6(); // bar6 called
        // 8, 13 as 5 is added by foo6 and bar6
foo6(10); // bar6 called
         // 10, 15 as x6 is 10 and therefore z6 to foo is 10+5
foo6(undefined,10); // 8, 10 as x6 default is y6+3 = 8 and z6 default value is 10
//
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 7. Destructuring from an array into simple variables – when you see [ ] on the left and array on the right

- The individual elements of array are destructured directly into individual variables

```
function foo7() {
    return [1,2,3];
}
var [a,b,c] = foo7();
console.log(a,b,c); // 1 2 3 simple variables
//
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 8. Destructuring from an object into simple variables – when you see { } on the left and object on the right

- The individual elements of the object are destructured directly into individual variables. The **variable name** should match the **property name**

```
console.log('Section 8: { } on left of assignment, Destructuring from object into variables');
//
const d8 = {
    a8: 9,
    b8: 12
};
const {a8, b8} = d8; // Destructing from object to variables
console.log (a8, b8); // 9 12
//
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 9. Property names can be computed using [ ]

- Using [ ] around an expression that returns a string you can compute property names

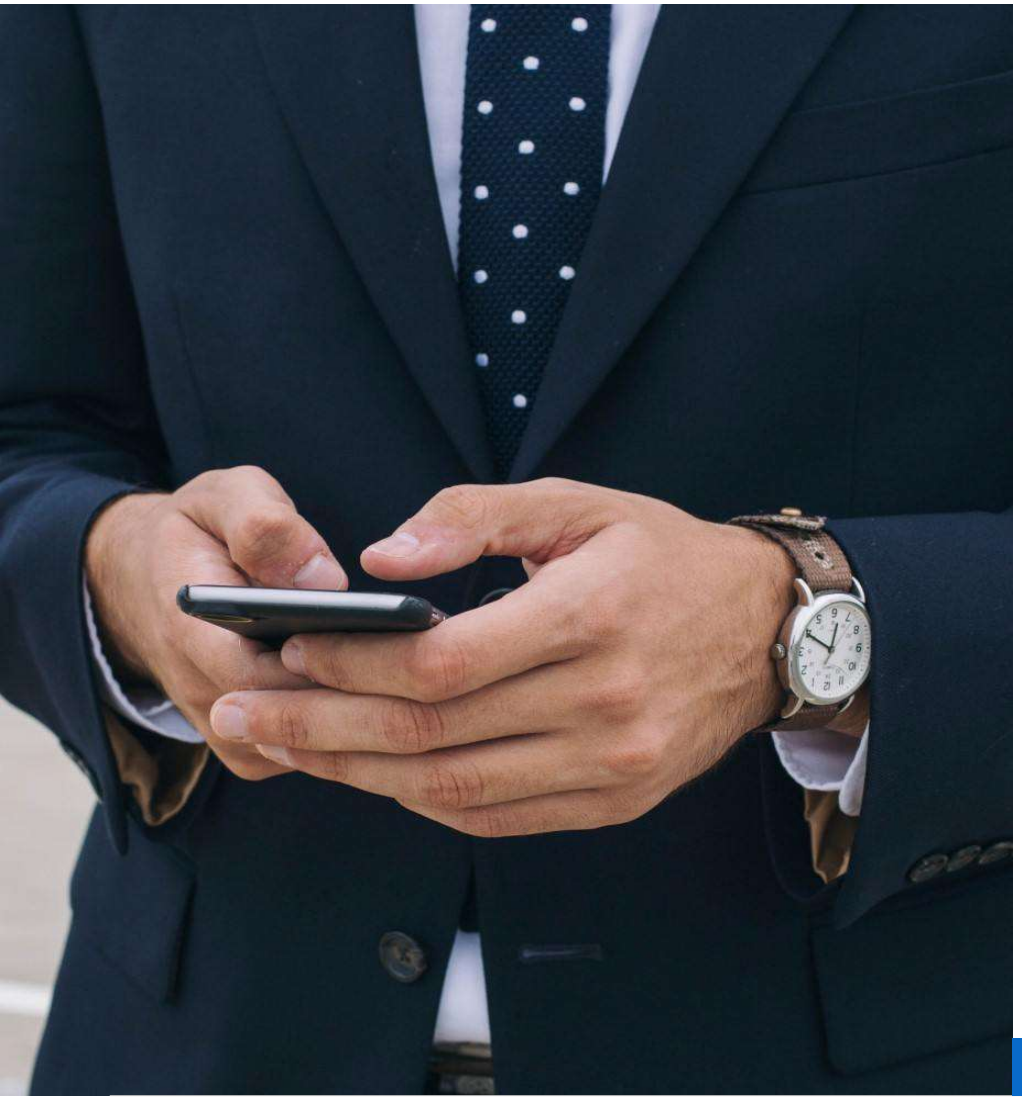
```
console.log('Section 9: [ ] around an expression, Property name can be computed');
//
var a9 = 'x1name';
var b9 = 'x2name';
cobj9 = { [a9]: 15, [b9]: 16};
console.log (cobj9); // { x1name: 15, x2name: 16 }
//
```

# Tutorial 3: ES6 features of JavaScript useful in Component development

## 10. Map function for Array and Objects – was available before ES6

- Map function is used extensively in component development. It map array elements values calling a mapping function for each element

```
console.log('Section 10: map function for an array - Not specific to ES6');
//
const array1 = [1,4,9,16];
// Pass a function to map
const mappedArray1 = array1.map(x=>x*2);
console.log(mappedArray1); // [2,8,18,32]
```



# ServiceNow
# NOW Experience Component Development Series

Tutorial # 4

JSX used in NOW Component Development

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Learning Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 4: JSX used in NOW Component Development

## 1. What is JSX

- JSX stands for JavaScript XML
- JSX is an XML/HTML like syntax that extends ECMAScript so that XML/HTML like text can co-exist with JavaScript/React code.
- The syntax is intended to be used by preprocessors (i.e. transpilers like Babel) to transform HTML like text found in JavaScript files into standard JavaScript objects that JavaScript engine will parse in such a way that subsequently they will be passed onto be a part of the DOM
- Using JSX you will write HTML in JavaScript and within this HTML you can embed JavaScript
- Similar to **Servlets** where we could write HTML inside Java and we had **JSP** where we could write Java inside HTML

# Tutorial 4: JSX used in NOW Component Development

### 2. Single line HTML Element inside JavaScript

- The HTML element can directly be written in JavaScript

```
const view = (state, {updateState}) => {
    return (
        <div>Hello World! This is my first component</div>
    );
};
```

- The Babel transpiler will know by angular bracket <> that this is XML and how to treat it

# Tutorial 4: JSX used in NOW Component Development

### 3. Multi-line HTML inside JavaScript

- If you need to embed multi-line HTML inside JavaScript, the HTML should be wrapped in a top level element like <ul></ul> or 
- And, we will need to wrap that into a set of parenthese

```
const view = (state, {updateState}) => {
    return (
        <ul>
            <li>Apples</li>
            <li>Banana</li>
            <li>Cherries</li>
        </ul>
    );
};
```

# Tutorial 4: JSX used in NOW Component Development

**4. Conditional rendering using if or inline ternary operator**

```
const view = (state, {updateState}) => {
    if (state.firstName) {
        return <h1>Hello we have the first name</h1>;
      }
      else {
        return <h1>Hello Guest</h1>;
      }
};
```

# Tutorial 4: JSX used in NOW Component Development

### 5. JavaScript inside HTML

- You can write JavaScript Expressions (variables, or property, or any other valid JS expression) within JSX using { }
- **Example 1**

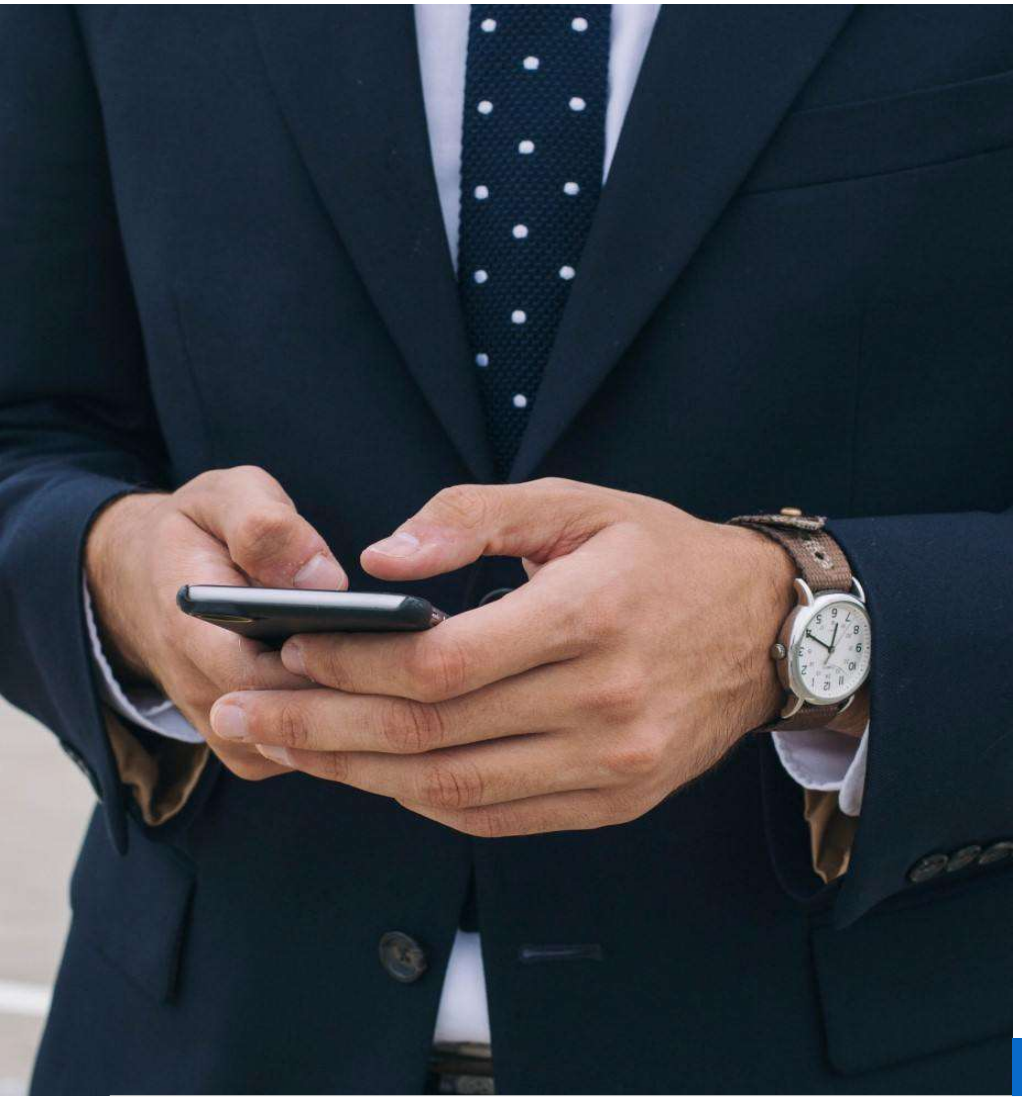
```
const view = (state, {updateState}) => {
    return <div>React is { 5 + 10 } times better with JSX</div>;
};
```

# Tutorial 4: JSX used in NOW Component Development

**5. JavaScript inside HTML**

- **Example 2**

```
const view = (state, {updateState}) => {
    return (
        <div>
            <h2>Click Counter</h2>
            <span>
                <button type="button"
                    on-click={
                        () =>  updateState({tally: (state.tally + 1)})
                    }>Increment
                </button>
            </span>
            <div>Value: {state.tally}</div>
      </div> )
};
```



# ServiceNow
# NOW Experience Component Development Series

Tutorial # 5

ServiceNow Component Source Code Directory structure

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

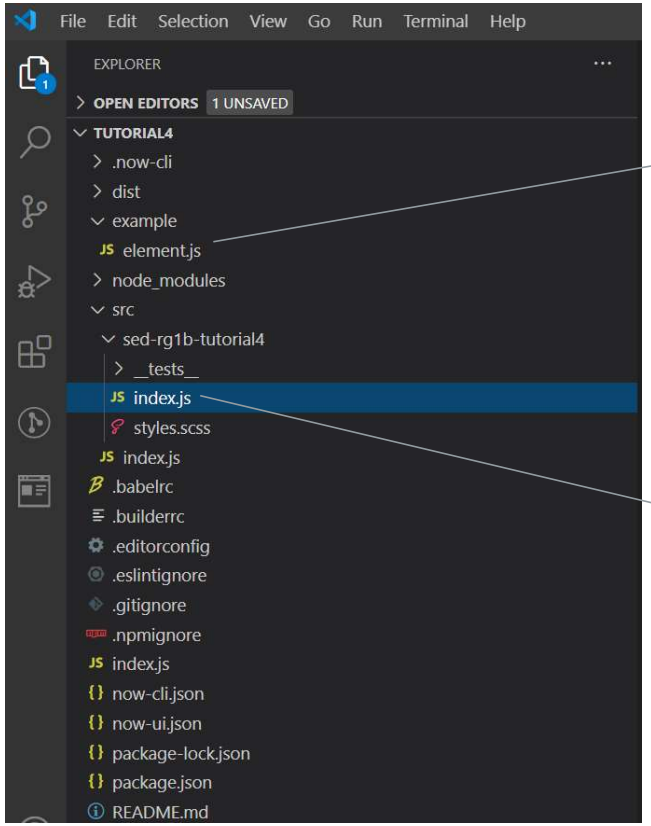
ServiceNow Technology Tutorial Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 5: ServiceNow Component Source Code Directory structure

## 1. Directory structure if there is only one component (no inner)  like in Hello World example




example > element.js > ...

```
import '../src/sed-rg1b-tutorial4';

const el = document.createElement('DIV');
document.body.appendChild(el);

el.innerHTML = `
<sed-rg1b-tutorial4></sed-rg1b-tutorial4>
`;
```

**element.js** – the starting file. Here, we have createElement and import statement

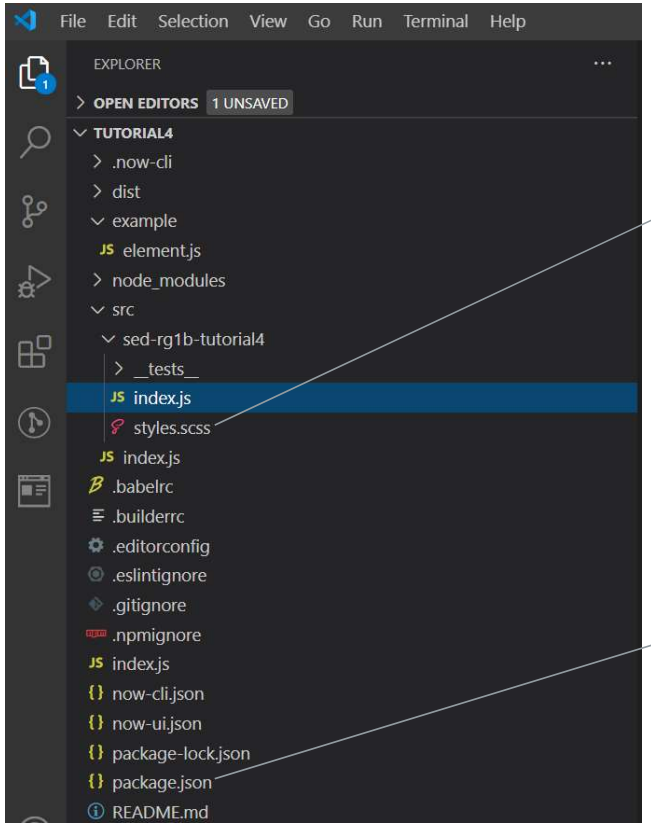
src > sed-rg1b-tutorial4 > index.js > ...

```
import {createCustomElement} from '@servicenow/ui-core';
import snabbdom from '@servicenow/ui-renderer-snabbdom';
import styles from './styles.scss';

const view = (state, {updateState}) => {
        return (
            <div>
                <h2>Click Counter</h2>
                <span>
                    <button type="button"
                        on-click={
                            () =>  updateState({tally: (state.tally + 1)})
                        }>Increment
                    </button>
                </span>
                <div>Value: {state.tally}</div>
            </div> )
    };

createCustomElement('sed-rg1b-tutorial4', {
    renderer: {type: snabbdom},
    initialState: {firstName: 'Rakesh', tally: 0},
    view,
    styles
});
```

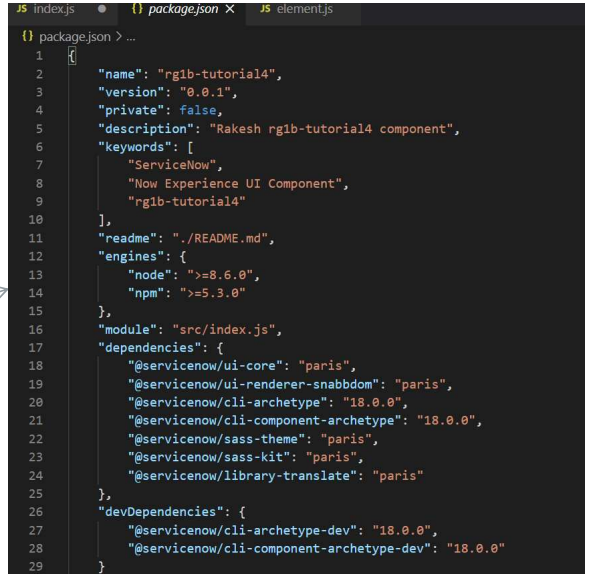
**index.js** – the file where the component is defined

# Tutorial 5: ServiceNow Component Source Code Directory structure

## 1. Directory structure if there is only one component (no inner)  like in Hello World example



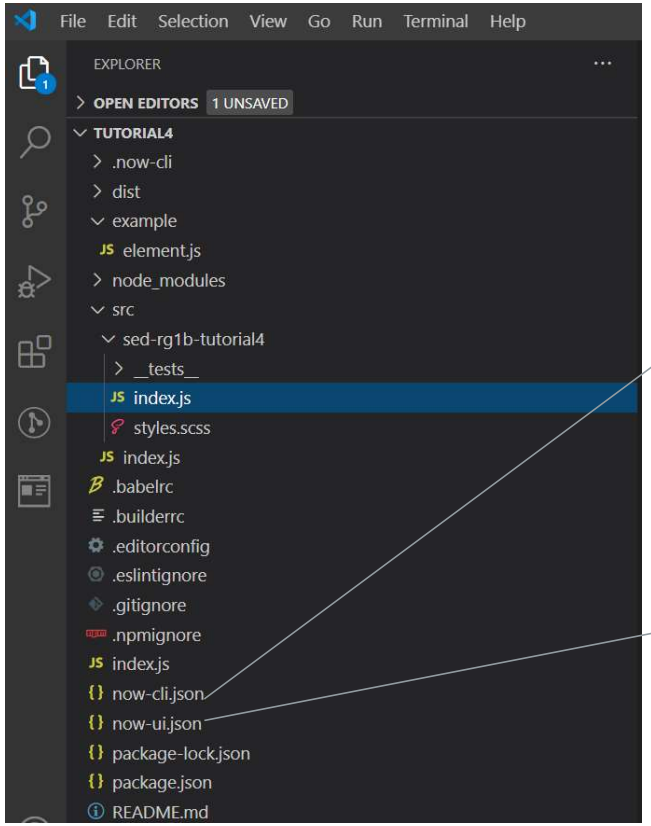

**css file**

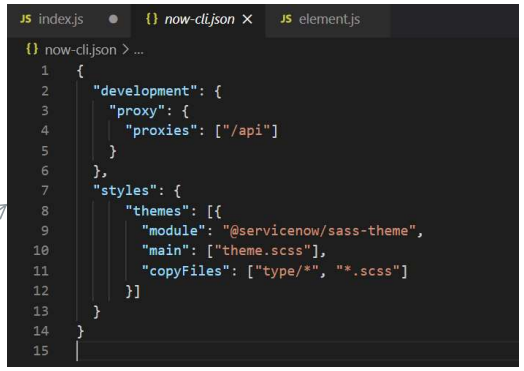




**package.json** file
Manually add names of all @import packages into dependencies as
'npm install' looks at this file to place packages in node_module

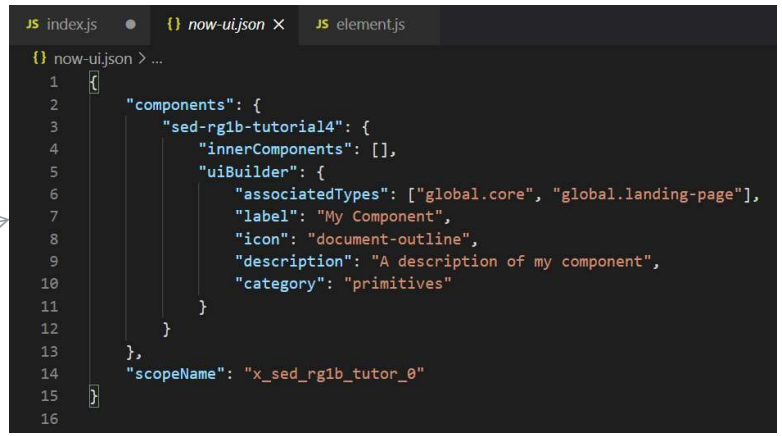
# Tutorial 5: ServiceNow Component Source Code Directory structure

## 1. Directory structure if there is only one component (no inner) like in Hello World example



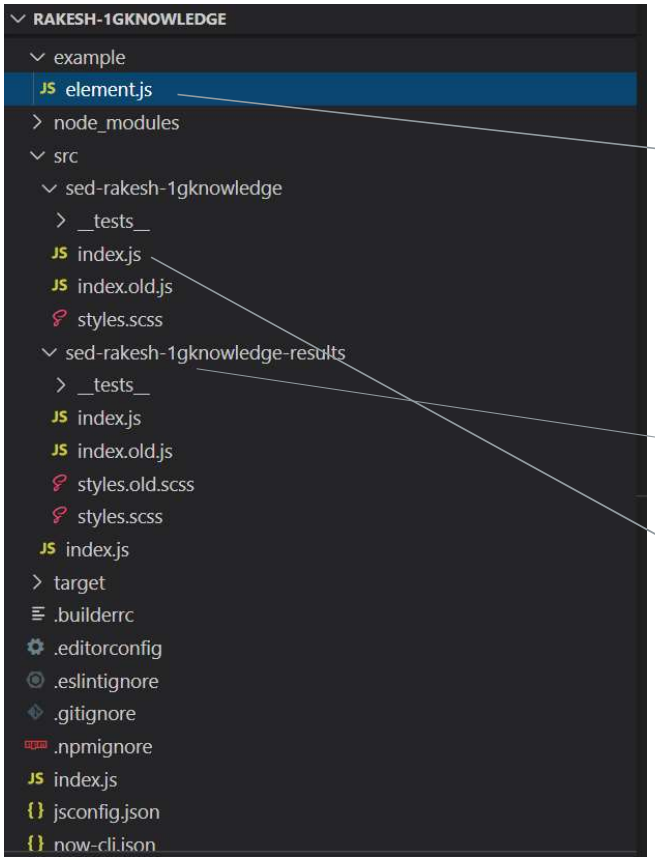

**now-cli.json**
To specify proxy for the ServiceNow instance if you are not using now-cli login command




**now-ui.json**
Used by now-cli deploy command – component name, label, description, scope name

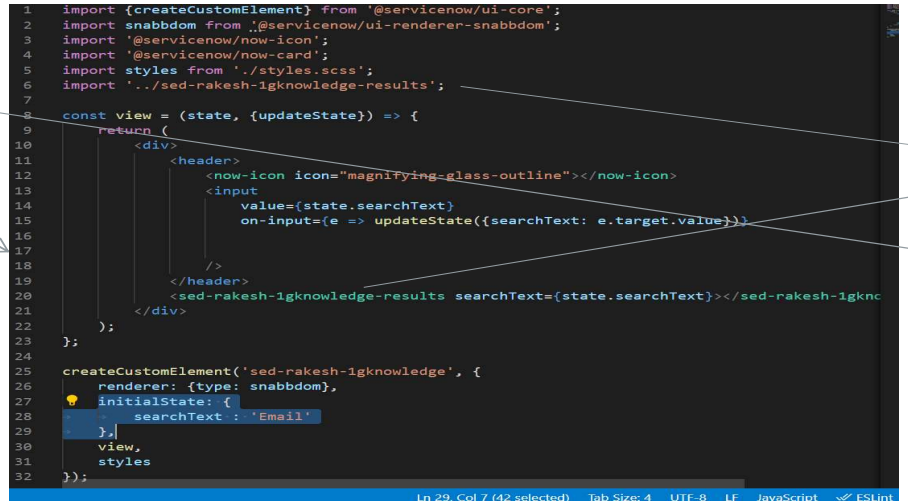
# Tutorial 5: ServiceNow Component Source Code Directory structure

## 2. Directory structure if there is an main and a secondary component – Knowledge Example





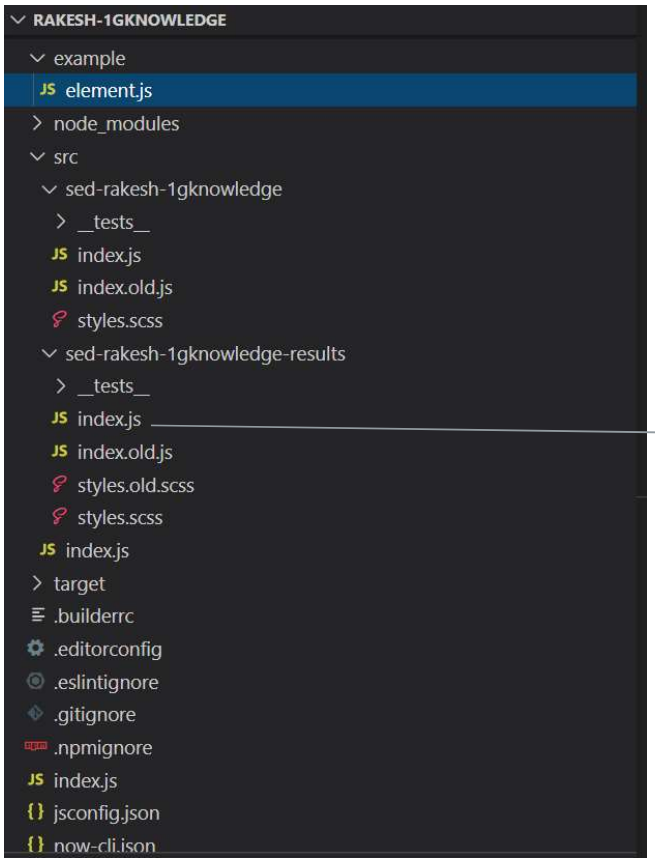
**Element.js for the main component**
createElement needed only for the main or outer component

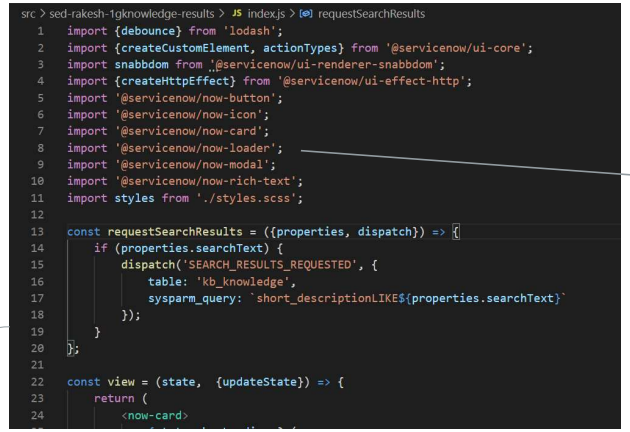



**Index.js for the main component**
There is an inner component whose code is in a separate directory tree – we have import statement here for that.

# Tutorial 5: ServiceNow Component Source Code Directory structure

## 2. Directory structure if there is an main and a secondary component – Knowledge Example



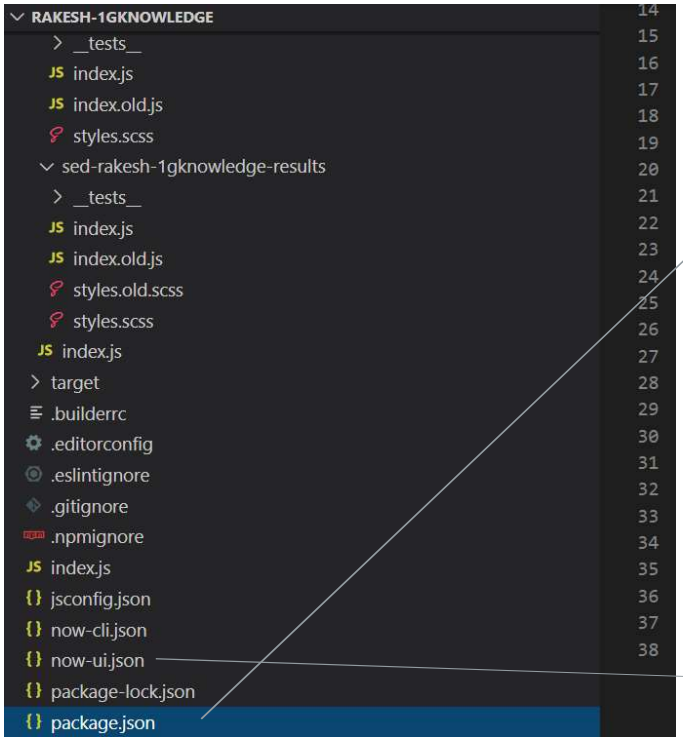


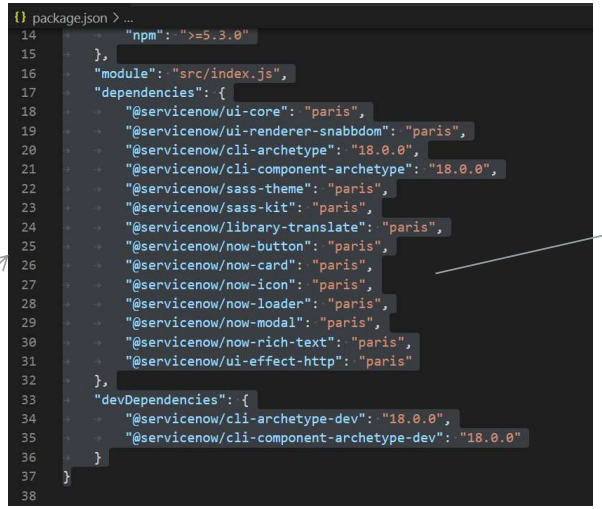
**Index.js of secondary component**
These are inner components. All these packages need to go into package.json file as shown on next slide. This is used by npm install to put packages in node_module directory

**Index.js of the secondary component (continued)**
createCustomeElement statement for the secondary component: sed-rakesh-1gknowledge-results

# Tutorial 5: ServiceNow Component Source Code Directory structure

## 2. Directory structure if there is an main and a secondary component – Knowledge Example

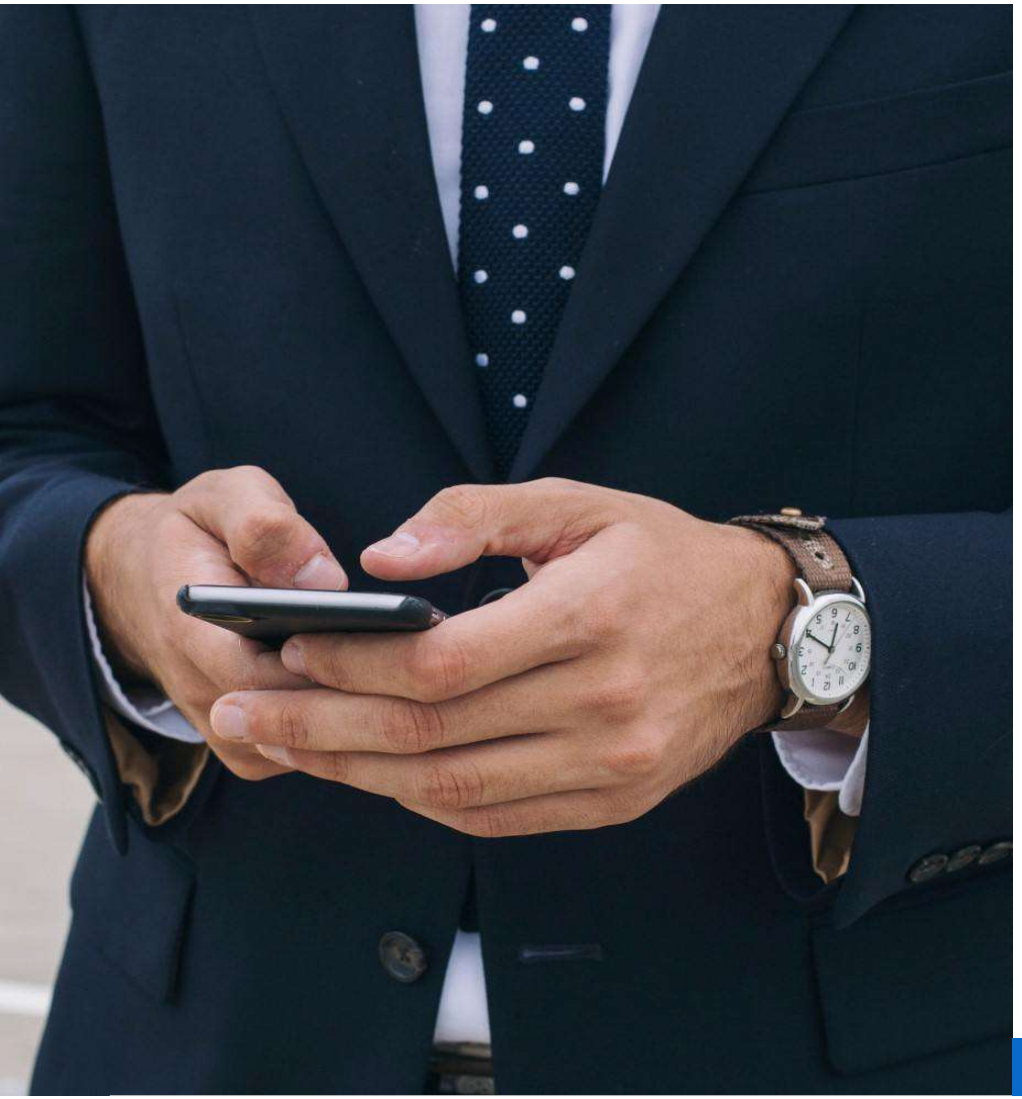




**package.json**
All @import packages (main component and secondary component) need to go into package.json file we discussed earlier. This is used by npm install to put packages in node_module directory

**now-ui** Will have all the inner components that we are using in our two components (these are ootb from ServiceNow)



# ServiceNow
# NOW Experience Component Development Series

Tutorial # 6

Develop a Master Detail Component – Search Knowledgebase

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Tutorial Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

**Create an empty folder for the project -> Use Visual Studio IDE and open the empty folder -> In Terminal window of IDE Login to SN -> now-cli project -> npm install -> Add your code -> Save All -> npm install -> now-cli develop --open and finally now-cli deploy -> Open Agent Workspace definition -> Open UI Builder -> Add component to the landing page**

## Step 1: Create an empty folder for your project and open the folder in Microsoft Visual Studio IDE

- Create a new directory or folder under one of your directories. Let us say we create a new empty folder named knowledge in a directory named g:\uex. So our folder is g:\uex\knowledge.

## Step 2: Log into ServiceNow instance from within the Terminal window of Microsoft Visual Studio code

- In the terminal window use the following now-cli's **login** command to log into your servicenow instance

```
g:\uex\knowledge>now-cli login --host https://ven03813.service-now.com --method basic --username admin --password Secret!
```

## Step 3: Create a NOW Project using now-cli from Terminal window within the Visual Studio code

```
g:\uex\knowledge>now-cli project --name rg1c-knowledge  --description 'UEX rg1c-knowledge'
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

**Step 4: Install project dependencies – or node packages needed by our newly created project**

```
g:\uex\knowledge >npm install
```

**Step 5: Use this Skeleton code to make a UEX component that will say "Hello World!...."**

```
const view = (state, {updateState}) => {
    return (
        <div>'Hello World! This is my first ServiceNow component'</div>
    );
};
```

**Step 6: Install project dependencies – or node packages needed by our newly created project**

```
g:\uex\knowledge >npm install
```

**Step 7: Invoke now-cli to develop the component based on our code and also open the component in browser**

```
g:\uex\knowledge >now-cli develop --open
```

- Important note: At two instances this command failed for me giving an error message. In that case I deleted the directory knowledge and re-performed all the above steps directly under DOS command window instead of terminal session within IDE and it worked. Only the step 5 was performed within the IDE as I had to change the code.

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## Step 8: Opened g:\uex\knowledge folder within the IDE and performed the following changes

- Created a directory named sed-rg1c-knowledge-results under the directory src to have the code of our secondary component. Also created two empty file named index.js and styles.scss.

## Step 9: Make changes to the code

- We will change index.js and styles.scss of our primary component sed-rg1c-knowledge
- We will change index.js and styles.scss of our secondary component sed-rg1c-knowledge-results
- We will make changes to package.json to add the packages that we are importing in our code so that npm install can install those packages
- We will make changes to now-ui.json to specify all the ServiceNow provided inner components that we are using

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

**Step 10: Important: The name of the primary component (in my code it is sed-rg1c-knowledge) to be same in element.js file, index.js of sed-rg1c-knowledge and now-ui.json files.**

- If you are doing hands-on by downloading whole of the code (with directories and sub-directories) than you are ok. However, if you create your own skeleton directory structure based on the detailed steps Step 1 to Step 9 including copying my code into your directory structure, make sure that the name of the primary component sed-rg1c-knowledge is same in the following three files. Though the name of the directories can stay as generated as those are taken care of by the import statements

- File element.js in the parent directory where we have createElement command

```
import '../src/sed-rg1c-knowledge';

const el = document.createElement('DIV');
document.body.appendChild(el);

el.innerHTML = `
<sed-rg1c-knowledge></sed-rg1c-knowledge>
`;
```

- File index.js of sed-rg1c-knowledge

```
createCustomElement('sed-rg1c-knowledge', {
    renderer: {type: snabbdom},
    initialState: {
        searchWords : 'Email'
    },
    view,
    styles
});
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

- File now-ui.json

```
{
    "components": {
        "sed-rg1c-knowledge": {
            "innerComponents": [
                "now-icon",
                "now-card",
                "now-button",
                "now-modal",
                "now-loader",
                "now-rich-text",
                "now-heading"
            ],
            "uiBuilder": {
```

------ continued ----

**Step 11: Ensure that import statements like this within the .js files are matching with your directory names**

```
import '../src/sed-rg1c-knowledge';
import '../sed-rg1c-knowledge-results';
```

In file element.js

In index.js of main component

**Step 12: Give npm install and now-cli develop --open commands within the terminal session of IDE**

- Make sure that you were already logged into your instance using now-cli login from within the terminal session of the IDE

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 1: element.js

```
import '../src/sed-rg1c-knowledge';

const el = document.createElement('DIV');
document.body.appendChild(el);

el.innerHTML = `
<sed-rg1c-knowledge></sed-rg1c-knowledge>
`;
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 2: index.js of main component - `sed-rg1c-knowledge`

```
import {createCustomElement} from '@servicenow/ui-core';
import snabbdom from '@servicenow/ui-renderer-snabbdom';
import '@servicenow/now-icon';
import '@servicenow/now-card';
import styles from './styles.scss';
import '../sed-rg1c-knowledge-results';
const view = (state, {updateState}) => {
    return (
        <div>
            <header>
                <now-icon icon="magnifying-glass-outline"></now-icon>
                <input
                    value={state.searchWords}
                    on-input={e => updateState({searchWords: e.target.value})}


                />
            </header>
            <sed-rg1c-knowledge-results searchText={state.searchWords}>
            </sed-rg1c-knowledge-results>
        </div>
    );
};
createCustomElement('sed-rg1c-knowledge', {
    renderer: {type: snabbdom},
    initialState: {
        searchWords : 'Email'
    },
    view,
    styles
});
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 3: styles.scss of main component - `sed-rg1c-knowledge`

```
@import '@servicenow/sass-kit/host';

:host{
    display: block;
    max-width: 30rem;
    margin: $now-global-space--xl auto;
header {
    display: flex;
    align-items: center;
    width: 100%;
    padding: 0 $now-global-space--md;
    border: 1px solid ($now-color--divider-tertiary);
    border-bottom: none;
    }

input {
    width: 100%;
    margin:0;
    padding: $now-global-space--md 0;
    border:none;
    outline:none;
    font-size: inherit;
    }
}
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 4: index.js of secondary component - `sed-rg1c-knowledge-results`

```
import {debounce} from 'lodash';
import {createCustomElement, actionTypes} from '@servicenow/ui-core';
import snabbdom from '@servicenow/ui-renderer-snabbdom';
import {createHttpEffect} from '@servicenow/ui-effect-http';
import '@servicenow/now-button';
import '@servicenow/now-icon';
import '@servicenow/now-card';
import '@servicenow/now-loader';
import '@servicenow/now-modal';
import '@servicenow/now-rich-text';
import styles from './styles.scss';

const requestSearchResults = ({properties, dispatch}) => {
    if (properties.searchText) {
        dispatch('SEARCH_RESULTS_REQUESTED', {
            table: 'kb_knowledge',
            sysparm_query: `short_descriptionLIKE${properties.searchText}`
        });
    }
};

const view = (state,  {updateState}) => {
    return (
        <now-card>
            {state.showLoading ? (
            <now-loader />
            ) : (
            <ul>
                {state.searchResults.length ? (
                    state.searchResults.map(result => (
                        <li>
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 4: index.js of secondary component - `sed-rg1c-knowledge-results (continued)`

```
                        <now-button-iconic
                            bare
                            icon="circle-info-outline"
                            size="md"
                            on-click={() =>
                                updateState({selectedResult: result}
                                )
                            }></now-button-iconic>
                        {result.short_description} </li>
                    ))
                ) : (
                    <li>No matches found </li>
                    )}
            </ul>
            )}
            {state.selectedResult ? (
            <now-modal
                opened={state.selectedResult}
                size="1g"
                footerActions={[
                    {
                        label: 'Done',
                        variant: 'secondary',
                        clickActionType: 'NOW_MODAL#OPENED_SET'
                    }
                ]}>
                    <now-rich-text html={state.selectedResult.text}>
                    </now-rich-text>
            </now-modal>
            ) : null }

        </now-card>
    );
};
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 4: index.js of secondary component - `sed-rg1c-knowledge-results (continued)`

```
    createCustomElement('sed-rg1c-knowledge-results', {
     renderer: {type: snabbdom},
     initialState: {
         showLoading: true,
         searchResults: [],
         selectedResult: null
     },
     properties: {
         searchText: {
             default: 'something'
         }
     },
     view,
     actionHandlers: {
         [actionTypes.COMPONENT_CONNECTED]:
         requestSearchResults,
         [actionTypes.COMPONENT_PROPERTY_CHANGED]:
         debounce(requestSearchResults, 250),
         SEARCH_RESULTS_REQUESTED: createHttpEffect('/api/now/table/:table', {
             pathParams: ['table'],
             queryParams: ['sysparm_query'],
             startActionType: 'SEARCH_RESULTS_STARTED',
             successActionType: 'SEARCH_RESULTS_FETCHED'
         }),
         SEARCH_RESULTS_STARTED: ({updateState}) =>
             updateState({showLoading: true}),
         SEARCH_RESULTS_FETCHED: ({action, updateState}) =>
             updateState({searchResults: action.payload.result, showLoading: false})
     },
     styles
});
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 5: styles.sccs of secondary component - `sed-rg1c-knowledge-results (continued)`

```
@import '@servicenow/sass-kit/host';
:host {
    display: block;
    ul {
        list-style: none;
        margin: 0;
        padding: 0;
        li {
            display: flex;
            align-items: center;
        }
        li + li {
            margin-top:
            $now-global-space--sm;
        }
    }
}
```

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 6: package.json component - `sed-rg1c-knowledge`

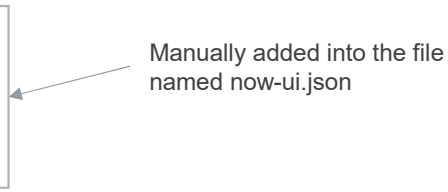
```
{
    "name": "rg1c-knowledge",
    "version": "0.0.1",
    "private": false,
    "description": "'UEX",
    "keywords": [
        "ServiceNow",
        "Now Experience UI Component",
        "rg1c-knowledge"
    ],
    "readme": "./README.md",
    "engines": {
        "node": ">=8.6.0",
        "npm": ">=5.3.0"
    },
    "module": "src/index.js",
    "dependencies": {
        "@servicenow/ui-core": "paris",
        "@servicenow/ui-renderer-snabbdom": "paris",
        "@servicenow/cli-archetype": "18.0.0",
        "@servicenow/cli-component-archetype": "18.0.0",
        "@servicenow/sass-theme": "paris",
        "@servicenow/sass-kit": "paris",
        "@servicenow/library-translate": "paris",
        "@servicenow/now-button": "paris",
        "@servicenow/now-card": "paris",
        "@servicenow/now-icon": "paris",
        "@servicenow/now-loader": "paris",
        "@servicenow/now-modal": "paris",
        "@servicenow/now-rich-text": "paris",
        "@servicenow/ui-effect-http": "paris"
    },
    "devDependencies": {
        "@servicenow/cli-archetype-dev": "18.0.0",
        "@servicenow/cli-component-archetype-dev": "18.0.0"
    }
}
```

Manually added into the file package.json

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 7: now-ui.json component - `sed-rg1c-knowledge`

```
{
    "components": {
        "sed-rg1c-knowledge": {
            "innerComponents": [
                "now-icon",
                "now-card",
                "now-button",
                "now-modal",
                "now-loader",
                "now-rich-text",
                "now-heading"
            ],
            "uiBuilder": {
                "associatedTypes": ["global.core", "global.landing-page"],
                "label": "My rg1c-knowledge Component",
                "icon": "document-outline",
                "description": "Description My rg1c-knowledge component",
                "category": "primitives"
            }
        }
    },
    "scopeName": "x_sed_rg1c_knowl_0"
}
```



# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 8. Logic

- **Start:**

DOM for displaying into the browser starts getting formed from the index.js of Primary component which in our case is sed-rg1c-knowledge. The view of this index.js returns a header element with a OOTB component `<now-icon icon="magnifying-glass-outline"></now-icon>` and `<sed-rg1c-knowledge-results searchText={state.searchWords}> </sed-rg1c-knowledge-results>`

Note that while invoking the secondary component `sed-rg1c-knowledge-results` we are setting a property named searchText of sed-rg1c-knowledge-results to state.searchWords of the component sed-rg1c-knowledge.

On invoking of the secondary component `sed-rg1c-knowledge-results` an Action named COMPONENT_CONNECTED takes place and therefore the following Action Handler of sed-rg1c-knowledge-results is invoked

```
[actionTypes.COMPONENT_CONNECTED]:
        requestSearchResults,
```

This handler calls requestSearchResults function which fills up the array named searchResults which is one of the State objects. The view of sed-rg1c-knowledge-results returns the content of this filled array within the DOM using Map function

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 8. Logic - continued

- **On user modifying the search words which are within component** sed-rg1c-knowledge

On User modifying the value inside the primary component that is displaying the magnifying glass, the state of sed-rg1c-knowledge is Updated using the logic:

```
<input
                    value={state.searchWords}
                    on-input={e => updateState({searchWords: e.target.value})}
                />
```

and therefore the view logic of sed-rg1c-knowledge is executed again with new value of state object named state.searchWords.

This time the sed-rg1c-knowledge-results is invoked with the changed value of the property named searchText. This invokes action handler `[actionTypes.COMPONENT_PROPERTY_CHANGED]` which calls function named `requestSearchResults again` to refresh the searched results array. The view displays the refreshed results

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 8. Logic - continued

- **On user clicking "circle-info-outline“ icon in front of any one of the results displayed in sed-rg1c-knowledge-results output**

On User clicking **circle-info-outline** icon in front any one of the results, the following logic is called:

```
on-click={() =>
                                updateState({selectedResult: result}
                                )
                            }
```

This logic makes the value of selectedResult from **null** to a **non-null** value which is the result row identifier.

Since the selectedResult has a non-null value, the following logic calls an out of the box component named now-modal which displays state.selectedResult.text

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## 8. Logic - continued

- **Miscellaneous**

Every time updateState function is called. One or more the state objects are updated by this function and the concerned component is re-invoked.

User actions create DOM events which in turn either call updateState or dispatches an Action. The updateState re-invokes the component while dispatch action invokes the related ActionHandler.

We use a function named `requestSearchResults` defined in component sed-rg1c-knowledge-results. This function dispatches an action named SEARCH_RESULTS_REQUESTED. This action's action handler takes help of a ServiceNow provided function named createHTTPEffect. This function uses supporting action handlers named SEARCH_RESULTS_STARTED and SEARCH_RESULTS_FETCHED to fill the array named searchResults (which is one of the objects of the state).

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## Components

**Component sed-rg1c-knowledge**

state: searchWords
properties:
ActionHandlers:

**Component sed-rg1c-knowledge-results**

state: `showLoading: true,`
`searchResults: [],`
`selectedResult: null`

properties: searchText

ActionHandlers:

`[actionTypes.COMPONENT_CONNECTED]:`

`[actionTypes.COMPONENT_PROPERTY_CHANGED]:`

`SEARCH_RESULTS_REQUESTED`

`SEARCH_RESULTS_STARTED`

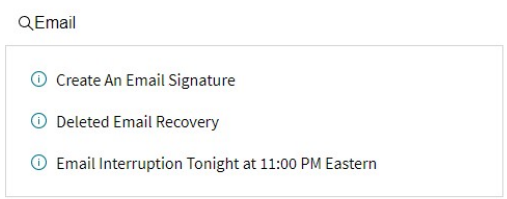
`SEARCH_RESULTS_FETCHED`

**now-modal**

properties: opened

Passes: `state.searchWords from sed-rg1c-knowledge component to property searchText of component sed-rg1c-knowledge-results`

## View




## Flow Logic

**A. Initial State**

1. sed-rg1c-knowledge has 'Email' based on value in state.searchWords from Initialstate
2. sed-rg1c-knowledge-results gets property searchText set as Email from index.js of sed-rg1c-knowledge. And, in sed-rg1c-knowledge-results using the ActionHandler `[actionTypes.COMPONENT_CONNECTED]it` fills state.searchResults array. The action handler calls the function requestSearchResults. This in turn takes help of action handler SEARCH_RESULTS_REQUESTED. The searched results are displayed in the return from the view of the sed-rg1c-knowledge-results component. SEARCH_RESULTS_STARTED and SEARCH_RESULTS_FETCHED support createHTTPEffect function that fetches the results.

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## Components

**Component sed-rg1c-knowledge**

state: searchWords
properties:
ActionHandlers:

**Component sed-rg1c-knowledge-results**

state: `showLoading: true,`
`searchResults: [],`
`selectedResult: null`

properties: searchText

ActionHandlers:

`[actionTypes.COMPONENT_CONNECTED]:`

`[actionTypes.COMPONENT_PROPERTY_CHANGED]:`

`SEARCH_RESULTS_REQUESTED`

`SEARCH_RESULTS_STARTED`

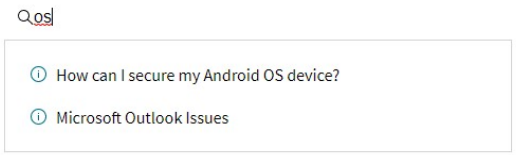
`SEARCH_RESULTS_FETCHED`

**now-modal**

properties: opened

Passes:
`state.searchWords from sed-rg1c-knowledge component to property searchText of component sed-rg1c-knowledge-results`

## View




## Flow Logic

**B. User changes the value of Search Words to os – this is an event in component sed-rg1c-knowledge**

1. On input sed-rg1c-knowledge calls logic

```
on-input={e => updateState({searchWords: e.target.value})}
```

`to update the searchWords to new value.`

2. The tag <sed-rg1c-knowledge-results> in index.js of primary component therefore updates property searchText to the new value of searchWords.

3. In sed-rg1c-knowledge-results using the ActionHandler `[actionTypes.COMPONENT_PROPERTY_CHANGED]` refills it's state.searchResults array. The action handler does this by calling the requestSearchResults function again. The searched results are displayed in the return from the view of the sed-rg1c-knowledge-results component. SEARCH_RESULTS_STARTED and SEARCH_RESULTS_FETCHED support createHTTPEffect function that fetches the results.

# Tutorial 6: Develop a Master Detail Component – Search Knowledgebase

## Components

### Component sed-rg1c-knowledge

state: searchWords
properties:
ActionHandlers:

### Component sed-rg1c-knowledge-results

state: `showLoading: true,`
`searchResults: [],`
`selectedResult: null`

properties: searchText

ActionHandlers:

`[actionTypes.COMPONENT_CONNECTED]:`

`[actionTypes.COMPONENT_PROPERTY_CHANGED]:`

`SEARCH_RESULTS_REQUESTED`

`SEARCH_RESULTS_STARTED`

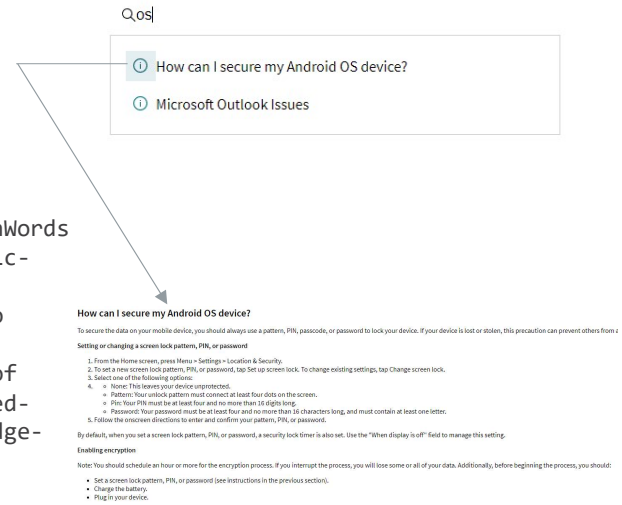
`SEARCH_RESULTS_FETCHED`

### now-modal

properties: opened

## View

Passes:
`state.searchWords from sed-rg1c-knowledge component to property searchText of component sed-rg1c-knowledge-results`




Passes:
`opened={state.selectedResult} to now-modal component`

## Flow Logic

**C. User clicks on `circle-info-outline` icon – this is a user action on sed-rg1c-knowledge-results component**

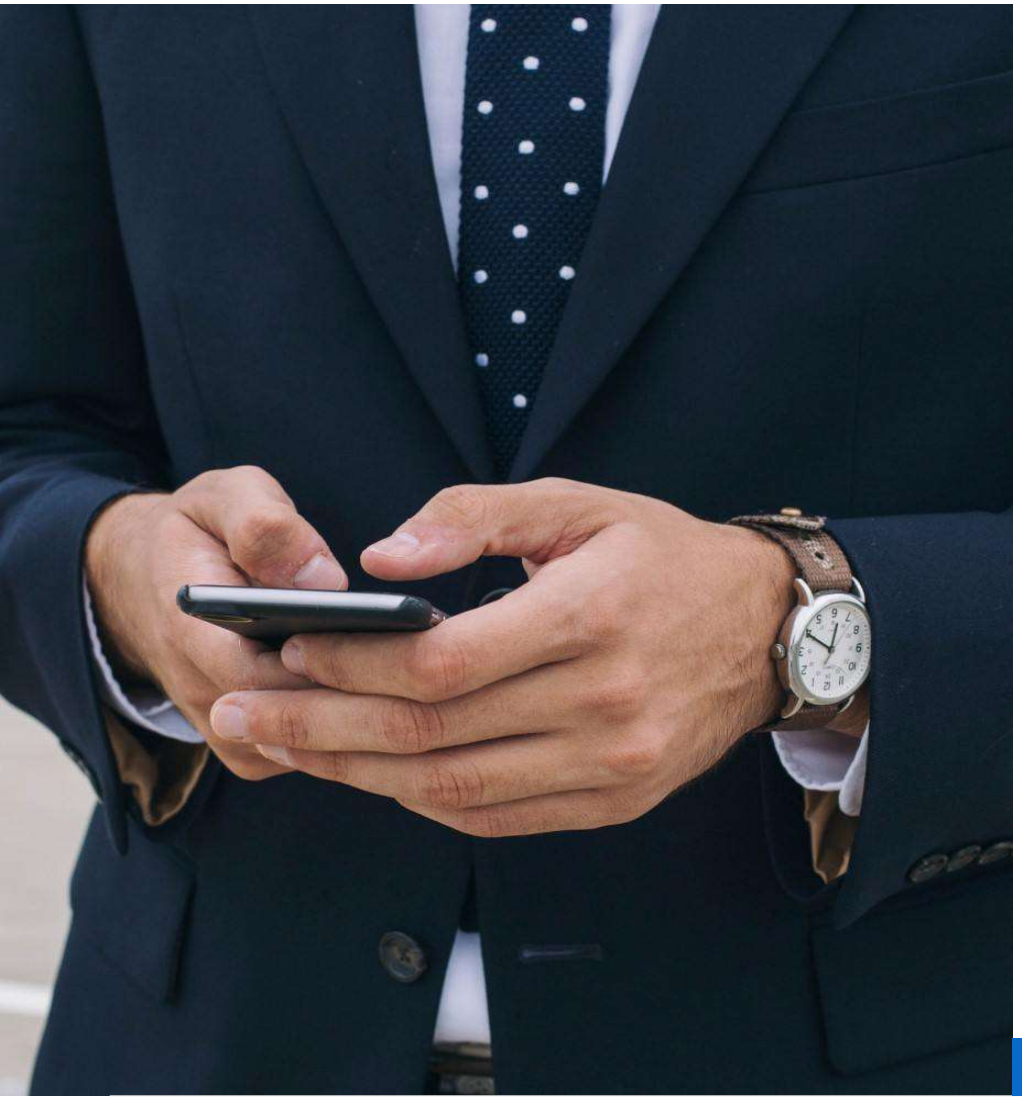
1. This invokes the following logic in index.js of sed-rg1c-knowledge-results:

```
on-click={() =>

  updateState({selectedResult:
 result}

  )
```

2. This makes the value of selectedResult from null to a non-null value equal to the identifier of the result row selected.

3. Therefore, the following logic gets executed to display the now-modal in the DOM

```
{state.selectedResult ? (
<now-modal
                  ……..

<now-rich-
text html={state.selectedResul
t.text}></now-rich-text>
</now-modal>
```



# ServiceNow
# NOW Experience Component Development Series

## Tutorial # 6 Add On

Master Detail Component (continued) – Quick explanation of the code

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Tutorial Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation

# Tutorial 7: Master Detail Knowledge component – Quick code explanation

## Start Action

- The action starts at element.js file which creates an element e1 with the e1.innerHTML as the tags which in our case are:

```
el.innerHTML = `
<sed-rg1c-knowledge></sed-rg1c-knowledge>
`
```

## Initial Display

- The control now goes to index.js file in sed-rg1c-knowledge directory the component `sed-rg1c-knowledge` is created. The createCustomElement statement initializes the objects stored in State. Like, we initialize state.searchWords = Email.
- The control goes to the return from the View. Here the initialized state objects are used to form the initial html.
- In our case the initial html has a text box in the header with first an icon of magnifying glass and then the text as initial value of state.searchWords which is email.
- This is followed with tags for results component <sed-rg1c-knowledge-results searchText={state.searchWords}>……
- This takes the control to the index.js of sed-rg1c-knowledge-results with value of the property named searchText set as state.searchWords which is Email in our case.
- In this file since we have defined Action handler for COMPONENT_CONNECTED, on component connection this handler is called, which in turn calls requestSearchResults func to fill the search results. These are displayed by Views return statement

# Tutorial 7: Master Detail Knowledge component – Quick code explanation

## On User inputing a new set of search words in primary component

- When user inputs new Search words, the following logic is triggered which updates the state.searchWords of sed-rg1c-knowledge component.

```
on-input={e => updateState({searchWords: e.target.value})}
```

- This updateState will make the sed-rg1c-knowledge component render again with the modified value of state.searchWords, for example state.searchWords = os.
- The rendering of sed-rg1c-knowledge component will re-execute return logic in View. This time with state.searchWords as os. This will render the input text box with searchWord os. While executing , <sed-rg1c-knowledge-results> tags it will go to index.js of sed-rg1c-knowledge-results with changed value of the property named searchText.
- In index.js of sed-rg1c-knowledge-results, we have an action handler defined for change in any property – COMPONENT_PROPERTY_CHANGED.
- This action handler calls requestSearchResults function again. This function in turns dispatches an action called SEARCH_RESULTS_REQUESTED. This action handler uses NOW provided API call named createHttpEffect to fill an array named state.SearchResults with the search results. createHTTPEffect is supported by supporting action handlers SEARCH_RESULTS_STARTED and SEARCH_RESULTS_FETCHED
- `state.searchResults.map` call in return of view displays the search results

# Tutorial 7: Master Detail Knowledge component – Quick code explanation

## On User selecting info icon in front of one of the results

- The index.js of sed-rg1c-knowledge-results component's following logic is triggered:

```
<now-button-iconic
    bare
    icon="circle-info-outline"
    size="md"
    on-click={() =>
        updateState({selectedResult: result}
        )
}></now-button-iconic>
```

- This updates the state of sed-rg1c-knowledge-results component making selectedResult from null to the selected result row.

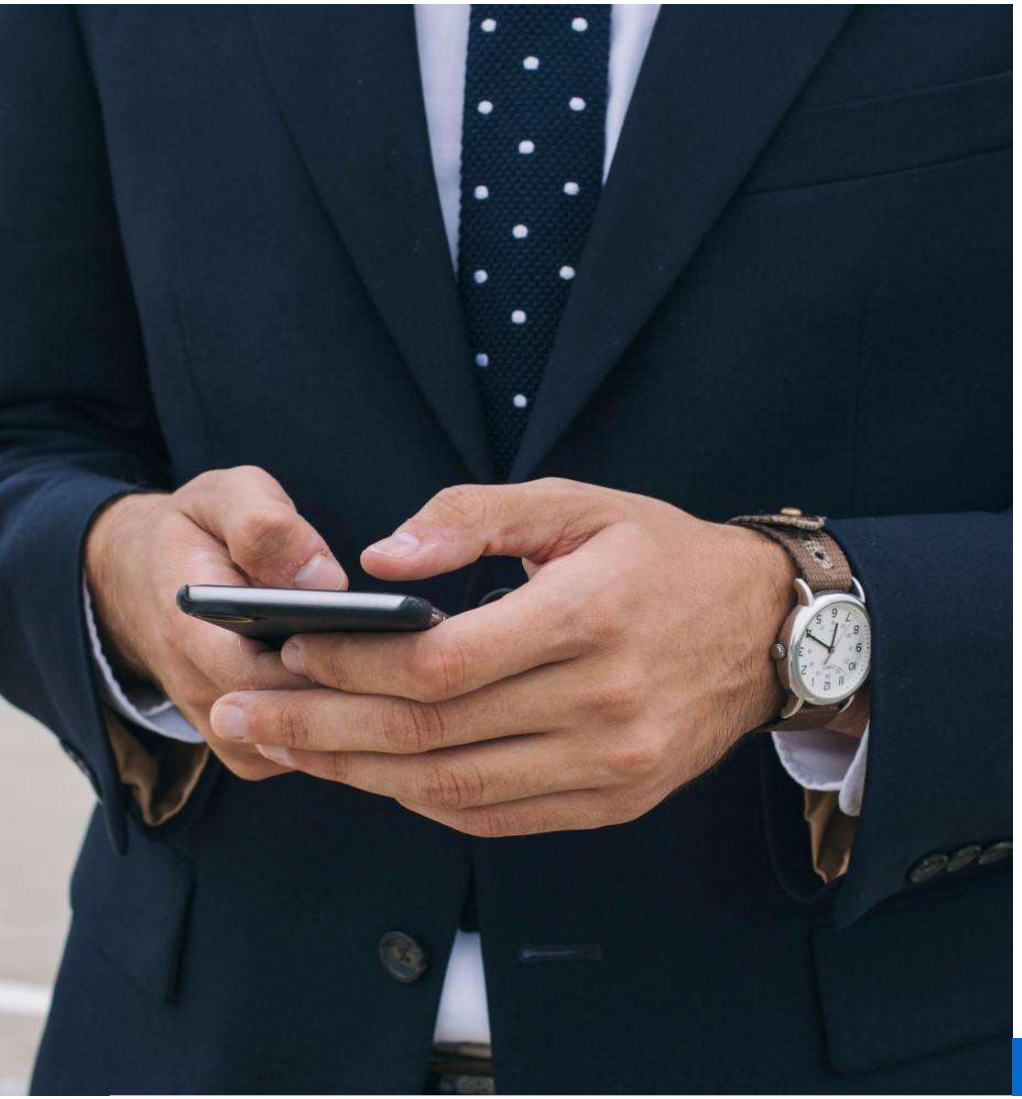
# Tutorial 7: Master Detail Knowledge component – Quick code explanation

## On User selecting info icon in front of one of the results

- This updateState will make sed-rg1c-knowledge-results render again. When it renders this time, within the return of View, the control goes to

```
{state.selectedResult ? (
            <now-modal
                ..
                    <now-rich-text html={state.selectedResult.text}>
                    </now-rich-text>
            </now-modal>
```

- This logic will open <now-model> component with the details of the selected result in state.selectedResult.text



# ServiceNow
# NOW Experience Component Development Learning Series

Thank you for attending

Learning reinforced through:

a. Power Point slides
b. Source code
c. Hands-on session

Slides & Code Download link at: www.servicedata.com/tutorials/

ServiceNow Technology Learning Series by ServiceData

Rakesh Goel, Founder & CTO

ServiceData
Intelligent Service Automation