



Week 3: Development Phase II

## Unit 1: Editor – Custom Development

# Editor – Custom Development

## Agenda

### Introduction

- Editor perspective
- Data context
- Functional events
- Scenarios
- Steps
- Demo

### Summary



# Editor – Custom Development

## Introduction

### Code generation: workflow designer → editor

- **Context** → JS data structures in `entities.js`
- **Events** → events declarations in `entities.js`
- **Workflows** → Scenario, steps, and starter declaration for each workflow in `nameofworkflow.js`

The diagram illustrates the process of code generation from a workflow designer to a code editor. On the left, the 'MyScenario' workflow designer shows a 'Start' event leading to a 'Custom' step. Below it, the 'Context' panel lists various data fields (ID, SupplierID, etc.) and the 'Events' panel lists event declarations (evSearchPurch..., evProcessPurch..., etc.). A large orange arrow points from the workflow designer to the right, where the SAP Intelligent Robotic Process Automation Desktop Studio is shown. The studio interface displays the 'Scripts' panel with a tree view of the project structure, including 'Framework', 'Declarations', 'Global', 'Test', 'Entities', and 'MyScenario'. The 'Entities' panel shows the generated JavaScript code for the scenario, including event declarations and data structures.

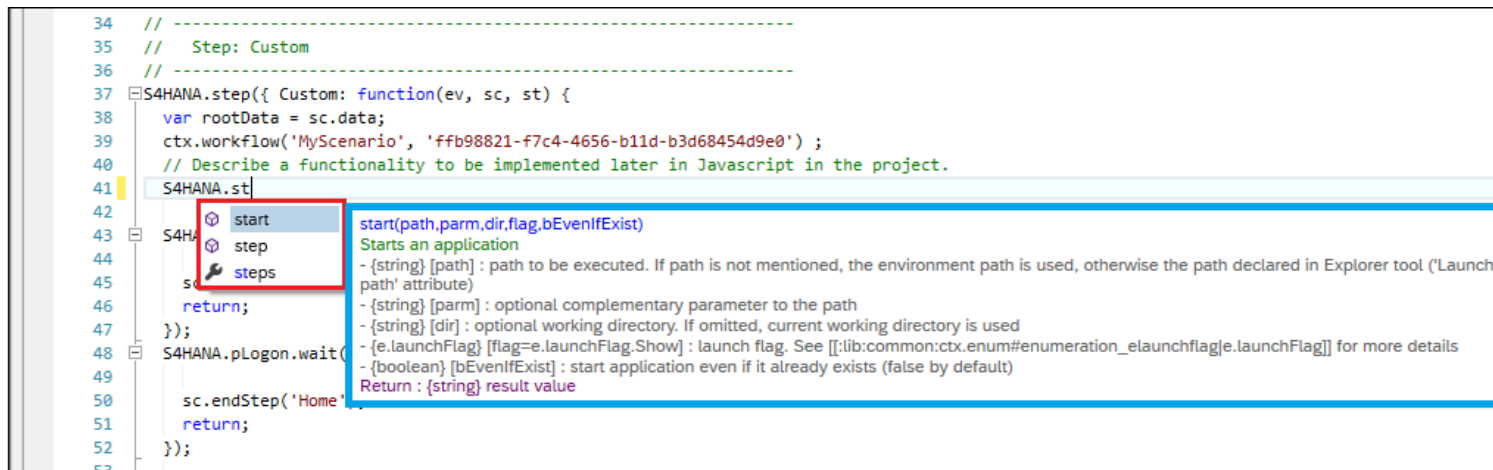
```
1 // -----  
2 // Test menu for scenario MyScenario  
3 // -----  
4 //  
5 GLOBAL.events.START.on(function (ev) {  
6   if (ctx.options.isDebugEnabled) {  
7     // Add item in systay menu.  
8     systay.addMenu('MyScenario', 'Test MyScenario', function (ev) {  
9       var rootData = ctx.dataManagers.rootData.create();  
10      // Initialize your data here.  
11      S4HANA.scenarios.MyScenario.start(rootData);  
12    });  
13  }  
14 });  
15  
16 //-----  
17 // Scenario MyScenario Starter ()  
18 //-----  
19 //  
20 // -----  
21 // Scenario: MyScenario  
22 // -----  
23 //  
24 S4HANA.scenario({ MyScenario: function(ev, sc) {  
25   var rootData = sc.data;  
26   sc.  
27   sc.setMode(e.scenario.mode.clearIfRunning);  
28   sc.setScenarioTimeout(600000); // Default timeout for global scenario.  
29   sc.onError(function(sc, st, ex) { sc.endScenario(); }); // Default error handler
```

## Editor – Custom Development

### Editor perspective

#### Coding assistants:

- **Code auto-completion:** Display all available properties and methods
- **Tooltips (or QuickInfo):** Display information for each method and parameter



- **Object edition (F12):** Navigate to entity declaration (Explorer) from the selected entity in the editor
- **Code navigation (F12):** Navigate to the definition code of any selected function or module
- **Object view (F8):** Show the view of the selected entity in the page viewer
- **Online help (F1):** Navigate to the help documentation from the selected item

## Editor – Custom Development

### Editor perspective

#### Code snippets:

##### ■ From context menu

- Position the cursor in the editor
- Right-click and chose **insert** or insert snippet
- Chose the module from the second menu
- Select the snippet you want
- The code is generated where the cursor is positioned

##### ■ From snippet alias

- Type the snippet alias + 'TAB' key
  - Ex. st + 'TAB'
  - The code is generated where the cursor is positioned

The diagram illustrates the process of inserting a code snippet in the SAP Editor perspective. It shows three main components: a 'Find and Replace' menu, a context menu, and a code editor.

**Find and Replace Menu:** This menu is open, showing various editing actions. The 'Insert' option is highlighted at the bottom.

**Context Menu:** This menu is open, showing a list of modules. The 'Scenario' module is highlighted. A sub-menu is also open, showing the 'Declare Step' option, which is highlighted.

**Code Editor:** The code editor shows the following code snippet being inserted:

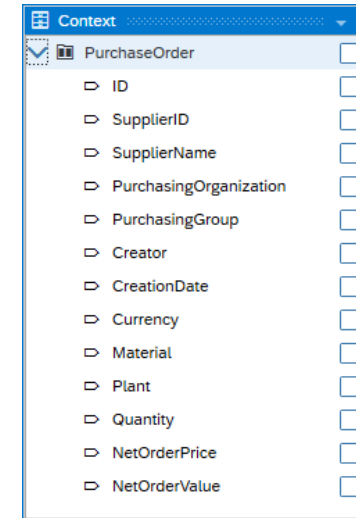
```
73  
74  
75 /** Description */  
76 app.step({ st: function(ev, sc, st) {  
77     var data = sc.data;  
78     sc.endStep();  
79     return;  
80 }});  
81  
82  
83
```

### Definition:

JavaScript data structures with subfolders and items

### Implementation:

- Implementation class:
  - `ctx.dataManger`
  - With a model for each subfolder
- Instantiation method:
  - `ctx.dataMangers.[sf].create()`
  - Example `sf = rootData_PurchaseOrder`



```
//----- Data Structures -----  
//----- rootData -----  
ctx.dataManager({  
  rootData :  
  {  
    PurchaseOrder :  
    {  
      ID : ''  
      , SupplierID : ''  
      , SupplierName : ''  
      , PurchasingOrganization : ''  
      , PurchasingGroup : ''  
      , Creator : ''  
      , CreationDate : ''  
      , Currency : ''  
      , Material : ''  
      , Plant : ''  
      , Quantity : ''  
      , NetOrderPrice : ''  
      , NetOrderValue : ''  
    }  
  }  
});  
var rootData = ctx.dataManagers.rootData.create();
```

```
//----- rootData_PurchaseOrder -----  
ctx.dataManager({  
  rootData_PurchaseOrder :  
  {  
    ID : ''  
    , SupplierID : ''  
    , SupplierName : ''  
    , PurchasingOrganization : ''  
    , PurchasingGroup : ''  
    , Creator : ''  
    , CreationDate : ''  
    , Currency : ''  
    , Material : ''  
    , Plant : ''  
    , Quantity : ''  
    , NetOrderPrice : ''  
    , NetOrderValue : ''  
  }  
});  
var rootData_PurchaseOrder = ctx.dataManagers.rootData_PurchaseOrder.create();
```



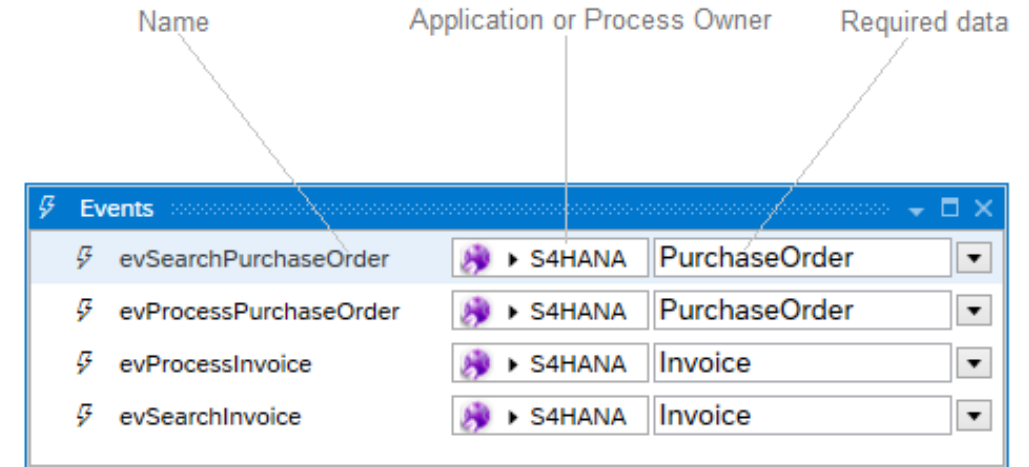
## Definition:

A functional event:

- A function called asynchronously
- Has a name
- Belongs to an entity: a process or an application
- Can have input data: a string or a JavaScript object

Its mechanism implies:

- A **receiver**: the owner entity receiving the notification
- A **sender**: the entity sending the notification



The screenshot shows a table titled 'Events' with three columns: 'Name', 'Application or Process Owner', and 'Required data'. The table contains four rows of data. Labels with arrows point from the column headers to the corresponding columns in the table.

Name	Application or Process Owner	Required data
evSearchPurchaseOrder	S4HANA	PurchaseOrder
evProcessPurchaseOrder	S4HANA	PurchaseOrder
evProcessInvoice	S4HANA	Invoice
evSearchInvoice	S4HANA	Invoice



```
//-----  
// Functional Events Declaration  
//-----  
S4HANA.addEvent({ evSearchPurchaseOrder : ctx.dataManagers.rootData_PurchaseOrder });  
S4HANA.addEvent({ evProcessPurchaseOrder : ctx.dataManagers.rootData_PurchaseOrder });  
S4HANA.addEvent({ evProcessInvoice : ctx.dataManagers.rootData_Invoice });  
S4HANA.addEvent({ evSearchInvoice : ctx.dataManagers.rootData_Invoice });
```

## Implementation:

- Implemented with the class:
  - `ctx.event`
- Declared using the method:
  - `ctx.application.addEvent`

## Notification:

- From any application or process:
  - `APPLI.notify`

## Delivery:

- Is always delivered asynchronously
- Can be delivered with a delay (in milliseconds)

```
//-----  
// Functional Events Declaration  
//-----  
S4HANA.addEvent({ evSearchPurchaseOrder : ctx.dataManagers.PurchaseOrder });  
  
// single listening of 'evSearchPurchaseOrder' event  
S4HANA.events.evSearchPurchaseOrder.once(function(ev) {  
    ...  
});
```

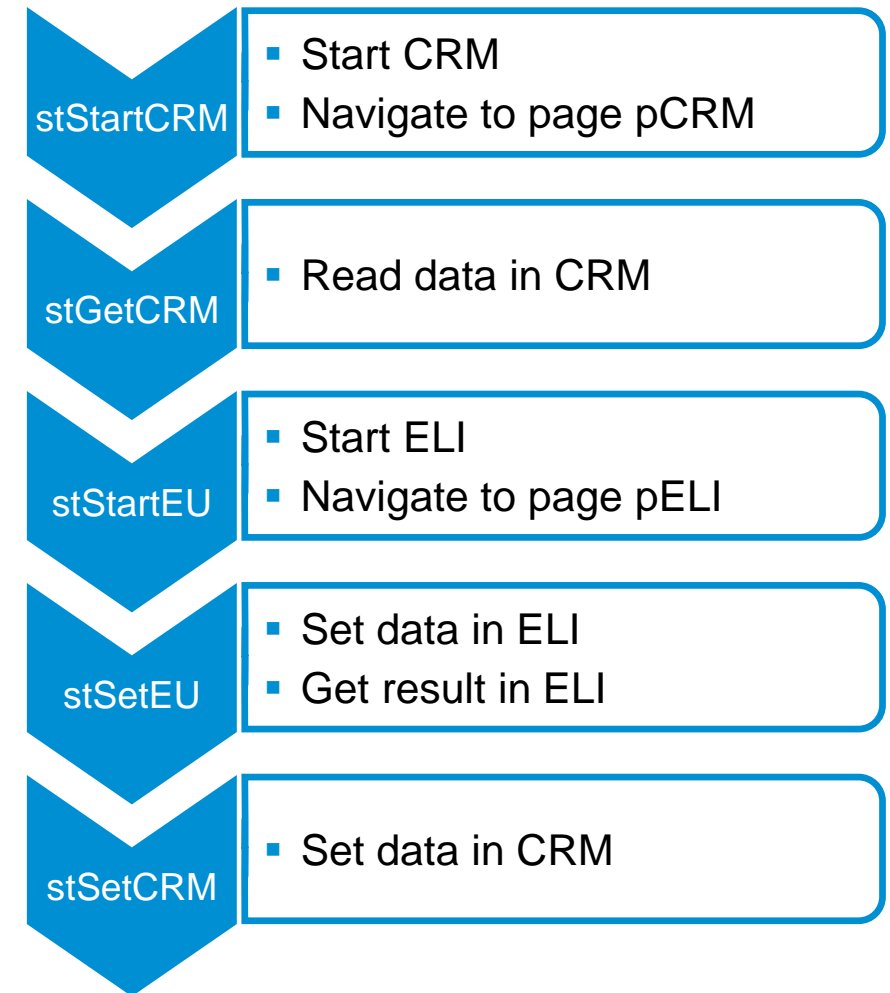
```
//-----  
// persistent listening of 'LOAD' event of the page 'pSearchPurchaseOrder'  
//-----  
S4HANA.pSearchPurchaseOrder.events.LOAD.on(function(ev) {  
  
    // notify an event to a 'S4HANA' application  
    GLOBAL.notify(S4HANA.events.evSearchPurchaseOrder, 'PurchaseOrder');  
  
});
```



**Definition:** a **scenario** is a set of **steps** running in a given **order** defined by a state machine.

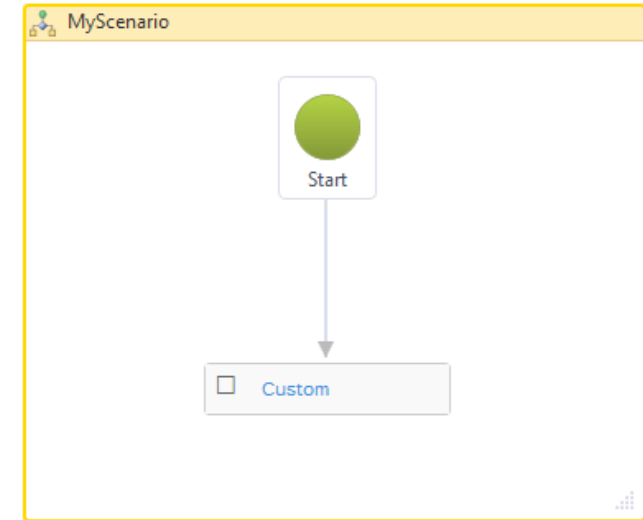
Design principles:

- Completion
  - Nominal cases first
  - Non-nominal and error cases
- Coherence
  - Avoid simultaneously active scenarios on the same applications
  - Avoid negative interference with the user
- Reusability
- Maintenance



A scenario has:

- A mode: starting conditions according to business rules
- A scenario timeout for the whole scenario
- A callback for errors/exception handling
- A step timeout for each step
- A callback for timeout handling
- A state machine of the steps used in the scenario
- A unique identifier: GUID generated by the workflow designer



```
// -----  
//   Scenario: MyScenario  
// -----  
S4HANA.scenario({ MyScenario: function(ev, sc) {  
    var rootData = sc.data;  
  
    sc.setMode(e.scenario.mode.clearIfRunning);  
    sc.setScenarioTimeout(600000); // Default timeout for global scenario.  
    sc.onError(function(sc, st, ex) { sc.endScenario(); }); // Default error handler.  
    sc.onTimeout(30000, function(sc, st) { sc.endScenario(); }); // Default timeout handler for each step.  
    sc.step(S4HANA.steps.Custom);  
}}, ctx.dataManagers.rootData).setId('189801de-667c-442b-ad2b-94a8f9a82a2f') ;
```

## Editor – Custom Development Scenarios

A set of steps running in a given order defined by a state machine.

### Steps and transitions (switch)

#### ■ Example:

##### Start

Step1 → Step2

Step2 → Step5 [Output5]

Step2 → Step3

Step3 → Step2 [Output2]

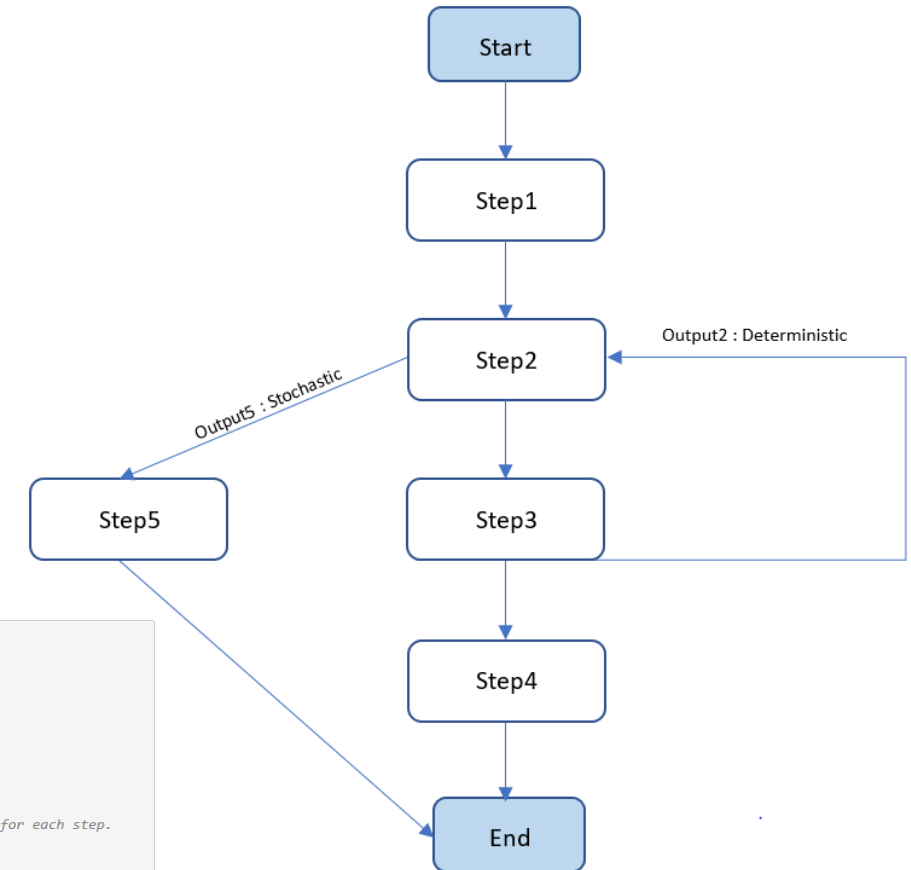
Step3 → Step4

Step4 → End Scenario

Step5 → End Scenario

##### End

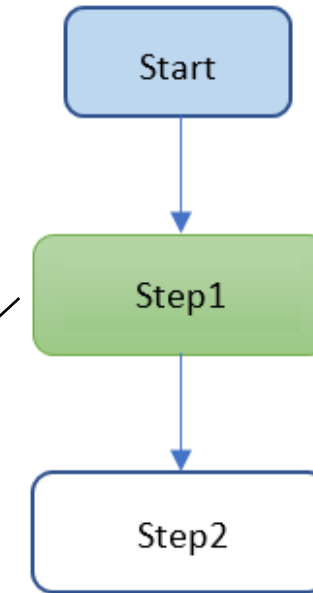
```
// -----  
// Scenario: scGameScenario  
// -----  
S4HANA.scenario({ scGameScenario: function(ev, sc) {  
    var rootData = sc.data;  
  
    sc.setMode(e.scenario.mode, clearIfRunning);  
    sc.setScenarioTimeout(600000); // Default timeout for global scenario.  
    sc.onError(function(sc, st, ex) { sc.endScenario(); }); // Default error handler.  
    sc.onTimeout(30000, function(sc, st) { sc.endScenario(); }); // Default timeout handler for each step.  
    sc.step(S4HANA.steps.stStep1, S4HANA.steps.stStep2);  
    sc.step(S4HANA.steps.stStep2, S4HANA.steps.stStep5, 'Output5');  
    sc.step(S4HANA.steps.stStep2, S4HANA.steps.stStep3);  
    sc.step(S4HANA.steps.stStep3, S4HANA.steps.stStep2, 'Output2');  
    sc.step(S4HANA.steps.stStep3, S4HANA.steps.stStep4);  
    sc.step(S4HANA.steps.stStep4, S4HANA.steps.stEndScenario);  
    sc.step(S4HANA.steps.stStep5, S4HANA.steps.stEndScenario);  
    sc.step(S4HANA.steps.stEndScenario, null);  
  
}}, ctx.dataManagers.rootData).setId('3645b351-27dd-4b89-8f71-ba460caf70cb') ;
```



## Definition:

### A step:

- is used in scenarios
- has a name and belongs to an entity: a process or an application
- has access to the scenario data: `sc.data`
- has a workflow GUID if generated with the workflow designer
- ends calling the function `sc.endStep()` with options for switch cases



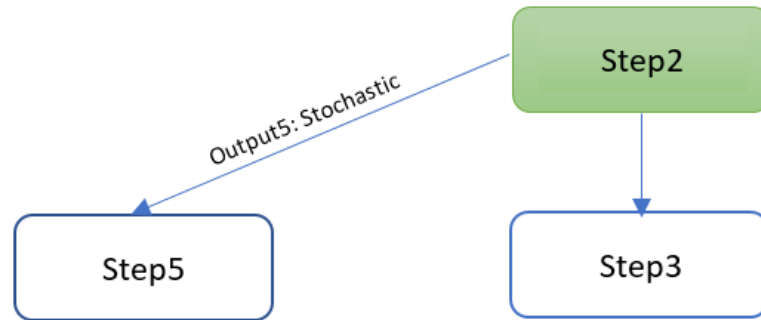
```
/** ----- Step: stStep1 ----- */
S4HANA.step({ stStep1: function(ev, sc, st) {
    var rootData = sc.data;
    ctx.workflow('scGameScenario', '40c9645e-8151-4f5c-ba02-809ec152ebde') ;
    // stStep1
    ctx.log('_____in Step'+st.name);
    sc.endStep();
    return;
}});
```

```
// -----
// Scenario: scGameScenario
// -----
S4HANA.scenario({ scGameScenario: function(ev, sc) {
    var rootData = sc.data;

    sc.setMode(e.scenario.mode.clearIfRunning);
    sc.setScenarioTimeout(600000); // Default timeout for global scenario.
    sc.onError(function(sc, st, ex) { sc.endScenario(); }); // Default error handler.
    sc.onTimeout(30000, function(sc, st) { sc.endScenario(); }); // Default timeout handler for each step.
    sc.step(S4HANA.steps.stStep1, S4HANA.steps.stStep2);
    sc.step(S4HANA.steps.stStep2, S4HANA.steps.stStep5, 'Output5');
    sc.step(S4HANA.steps.stStep2, S4HANA.steps.stStep3);
    sc.step(S4HANA.steps.stStep3, S4HANA.steps.stStep2, 'Output2');
    sc.step(S4HANA.steps.stStep3, S4HANA.steps.stStep4);
    sc.step(S4HANA.steps.stStep4, S4HANA.steps.stEndScenario);
    sc.step(S4HANA.steps.stStep5, S4HANA.steps.stEndScenario);
    sc.step(S4HANA.steps.stEndScenario, null);

}}, ctx.dataManagers.rootData).setId('3645b351-27dd-4b89-8f71-ba460caf70cb') ;
```

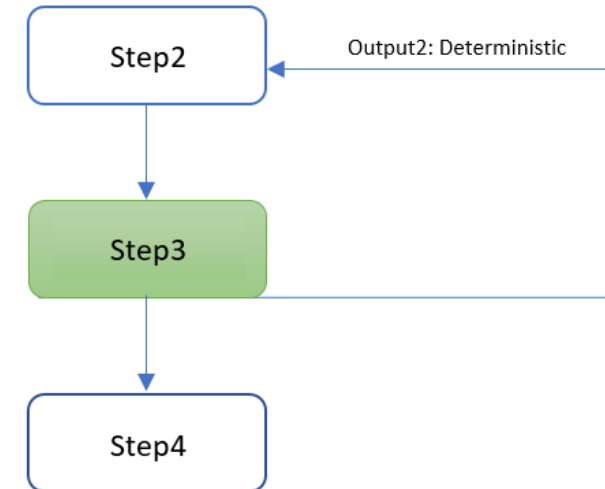
## Switch implementation



```
/** ----- Step: stStep2 ----- */
S4HANA.step({ stStep2: function(ev, sc, st) {

    /** here we are iun random process where
        we test a random generated value (0 <= rand <=10 )
    */

    // Get a random integer from 0 to 10
    var rand = Math.floor(Math.random() * 11);
    if(rand>9){
        // Step2 to Step5
        sc.endStep('Output5');
    } else {
        // Step2 to Step3
        sc.endStep();
    }
    return;
}});
```



```
/** ----- Step: stStep3 ----- */
S4HANA.step({ stStep3: function(ev, sc, st) {
    var data = sc.data;
    // In adeterministic process: we now how many iterations Left
    if(data.index<data.nbIterationStep2){
        ctx.log('€€€€ Still in game $$$ increment index €€€€ Step3 to Step2 ');
        data.index=data.index+1;
        sc.endStep('Output2');
    } else {
        ctx.log('#### Game Over #### Step3 to Step4');
        sc.endStep();
    }
    return;
}});
```

## Additional patterns:

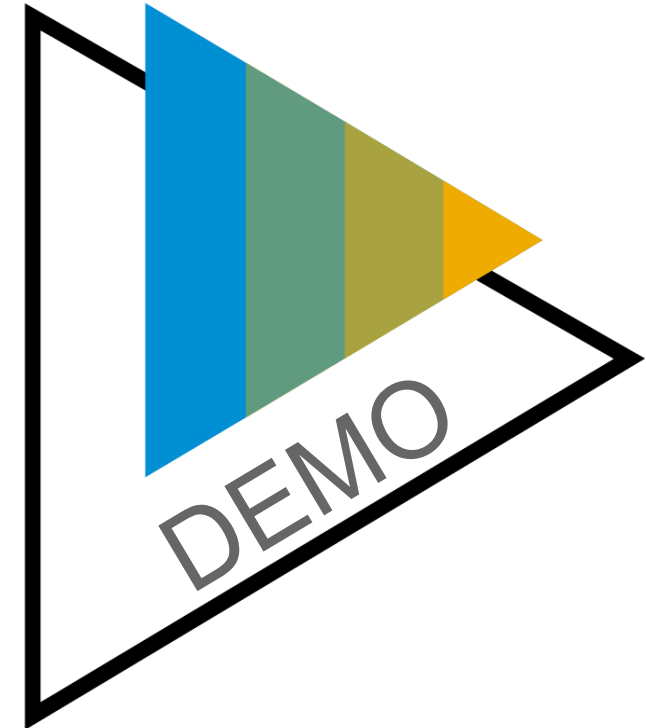
### Wait Multiple:

- Wait for pages
- Wait for buttons
- Wait for functional events
- Wait for timers and pollings
- etc...



**Demo view:**

- Workflow creation
  - Script generation
- Code customization
  - Initialize data
  - Create steps
  - Set the State machine of the scenario
  - Implement the steps
- Run
  - Show a run of the scenario





# Thank you.

**Contact information:**

**open@sap.com**

Follow all of SAP



[www.sap.com/contactsap](http://www.sap.com/contactsap)

© 2019 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platforms, directions, and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

See [www.sap.com/copyright](http://www.sap.com/copyright) for additional trademark information and notices.