Week 3: Development Phase II

Unit 1: Editor – Custom Development

Editor – Custom Development

Agenda

Introduction

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

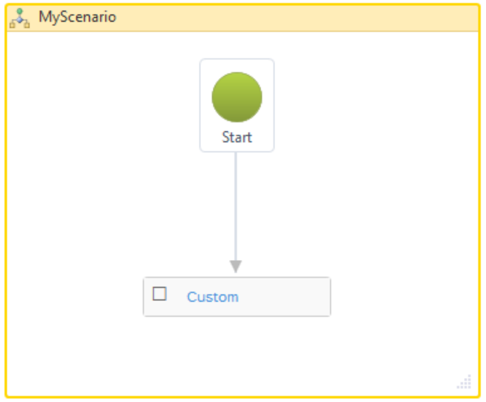
Summary

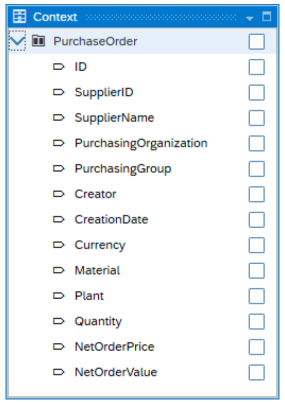
Editor – Custom Development

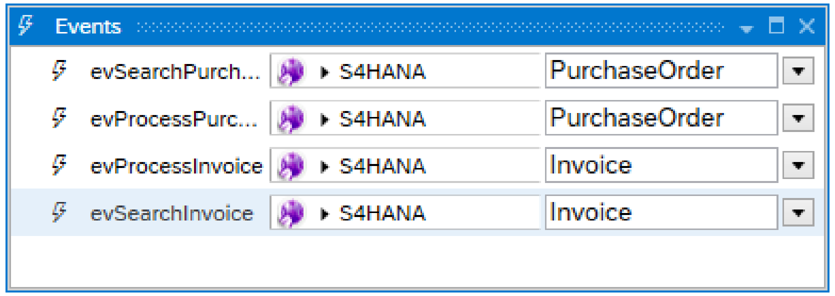
Introduction

Code generation: workflow designer → editor

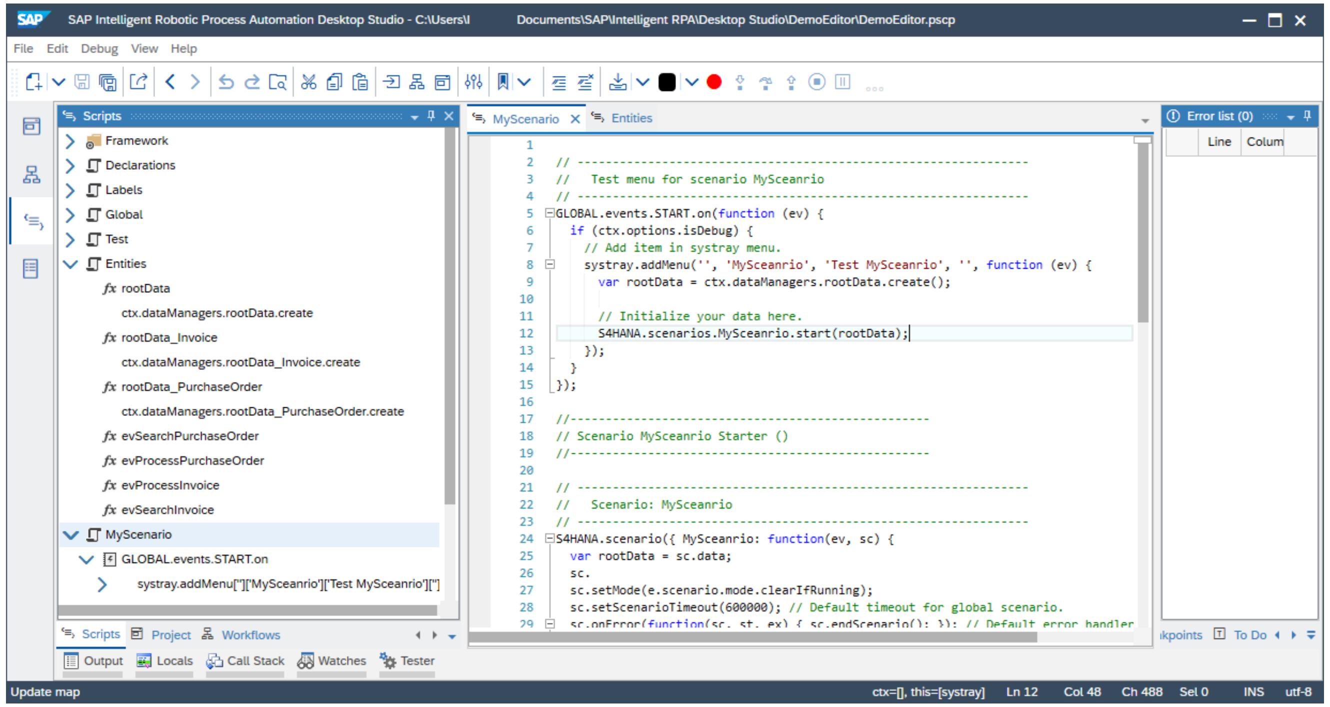
- **Context** → JS data structures in **`entities.js`**
- **Events** → events declarations in **`entities.js`**
- **Workflows** → Scenario, steps, and starter declaration for each workflow in **`nameofworkflow.js`**







evSearchPurch...	S4HANA	PurchaseOrder
evProcessPurc...	S4HANA	PurchaseOrder
evProcessInvoice	S4HANA	Invoice
evSearchInvoice	S4HANA	Invoice

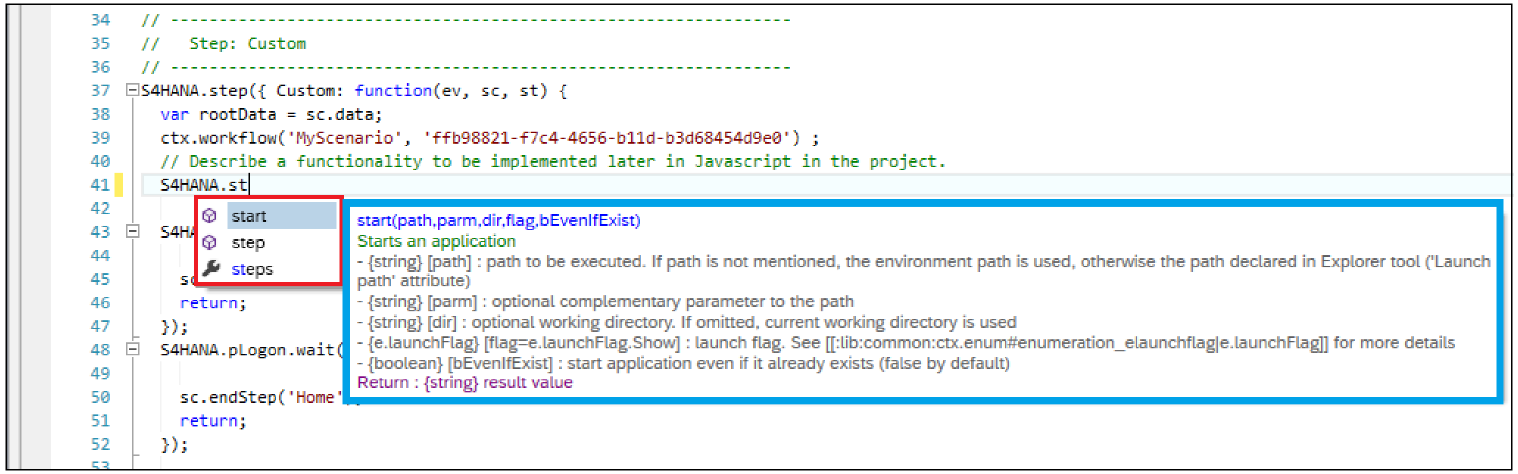


```
// -------------------------------------------------------------------
//   Test menu for scenario MySceanrio
// -------------------------------------------------------------------
GLOBAL.events.START.on(function (ev) {
  if (ctx.options.isDebug) {
    // Add item in systray menu.
    systray.addMenu('', 'MySceanrio', 'Test MySceanrio', '', function (ev) {
      var rootData = ctx.dataManagers.rootData.create();

      // Initialize your data here.
      S4HANA.scenarios.MySceanrio.start(rootData);
    });
  }
});

//-------------------------------------------------
// Scenario MySceanrio Starter ()
//-------------------------------------------------

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

Editor perspective

Coding assistants:

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



```
// ---------------------------------------------------------------
//   Step: Custom
// ---------------------------------------------------------------
S4HANA.step({ Custom: function(ev, sc, st) {
  var rootData = sc.data;
  ctx.workflow('MyScenario', 'ffb98821-f7c4-4656-b11d-b3d68454d9e0') ;
  // Describe a functionality to be implemented later in Javascript in the project.
  S4HANA.st

  S4H

    s
    return;
  });
  S4HANA.pLogon.wait(

    sc.endStep('Home'
    return;
  });
```

start(path,parm,dir,flag,bEvenIfExist)
Starts an application
- {string} [path] : path to be executed. If path is not mentioned, the environment path is used, otherwise the path declared in Explorer tool ('Launch path' attribute)
- {string} [parm] : optional complementary parameter to the path
- {string} [dir] : optional working directory. If omitted, current working directory is used
- {e.launchFlag} [flag=e.launchFlag.Show] : launch flag. See [[:lib:common:ctx.enum#enumeration_elaunchflag|e.launchFlag]] for more details
- {boolean} [bEvenIfExist] : start application even if it already exists (false by default)
Return : {string} result value

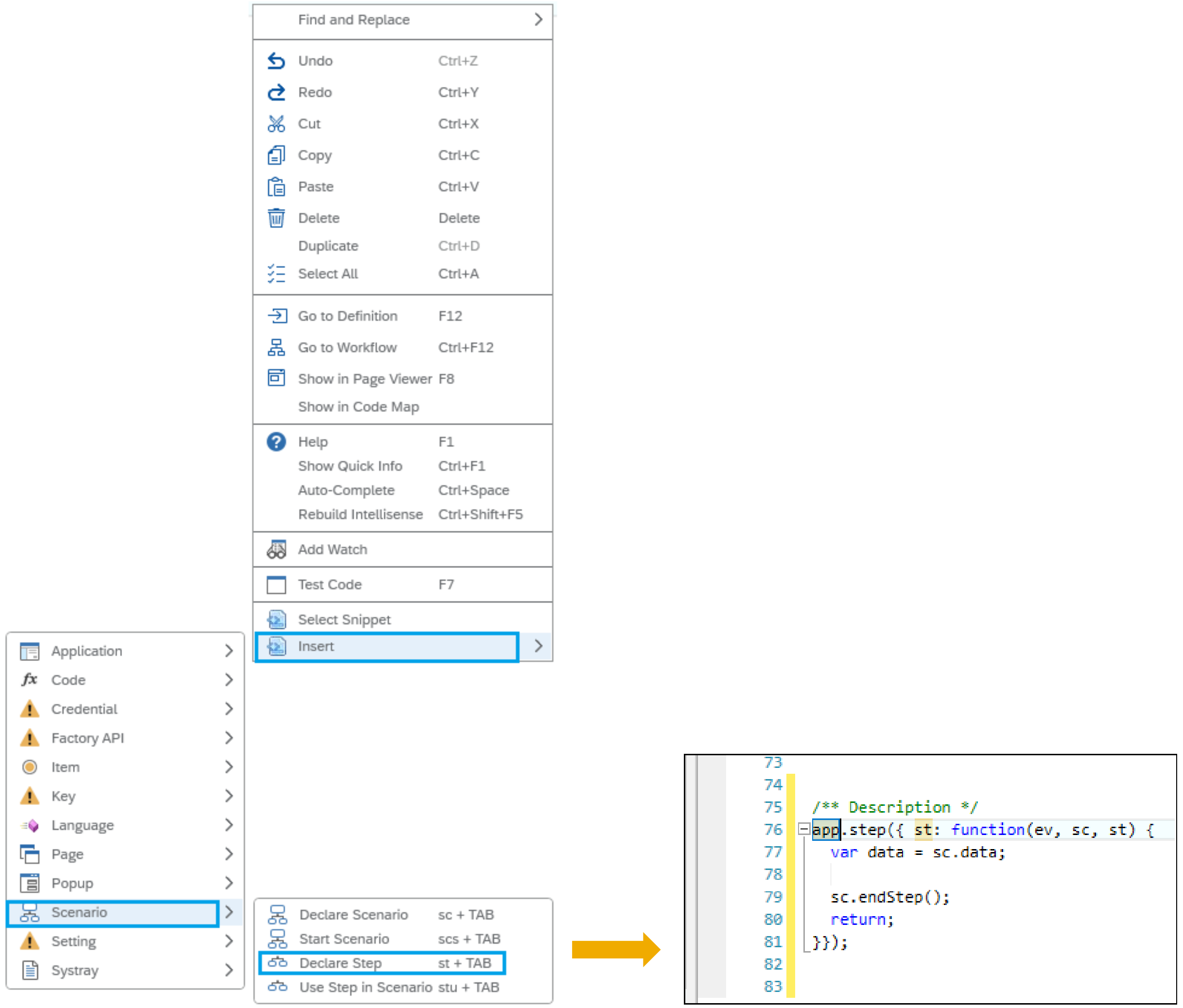
- **Object edition** (**F12**): Navigate to entity declaration (Explorer) from the selected entity in the editor
- **Code navigation** (**F12**): Navigate to the definition code of any selected function or module
- **Object view** (**F8**): Show the view of the selected entity in the page viewer
- **Online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



```
/** Description */
app.step({ st: function(ev, sc, st) {
  var data = sc.data;

  sc.endStep();
  return;
}});
```

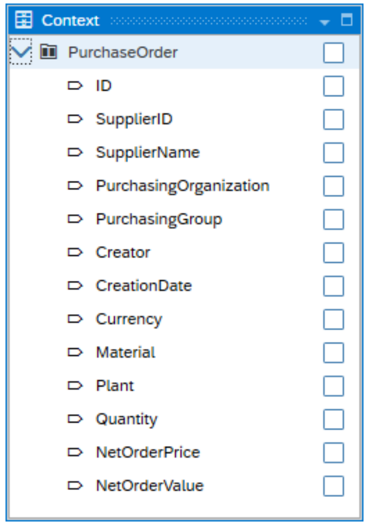
Data context

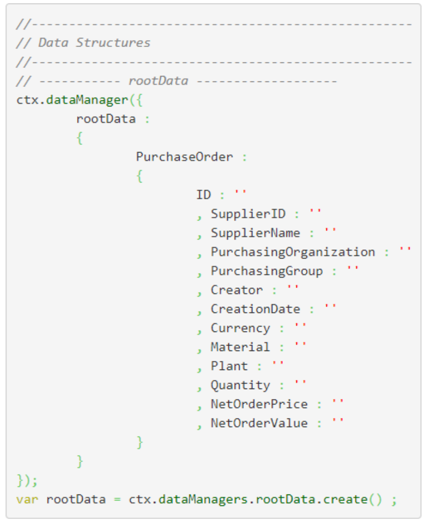
Definition:

JavaScript data structures with subfolders and items

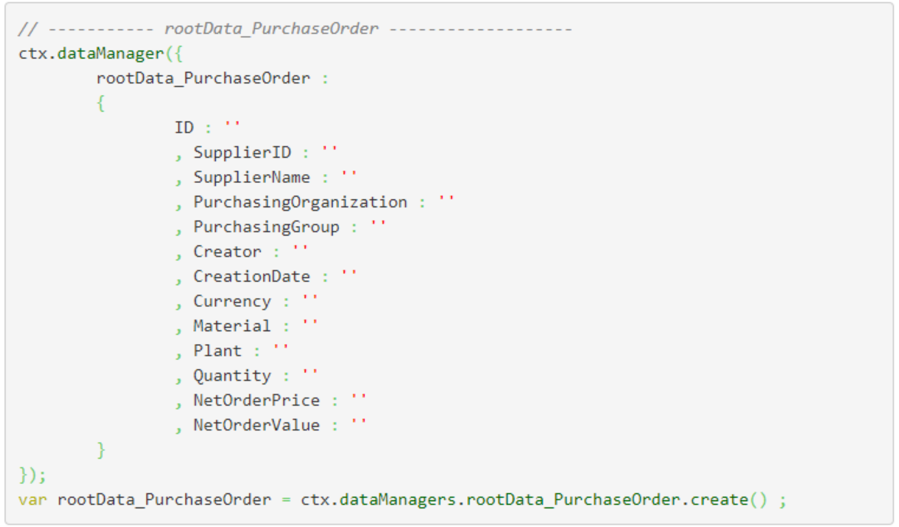
Implementation:

- 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() ;
```

Editor – Custom Development

Functional events

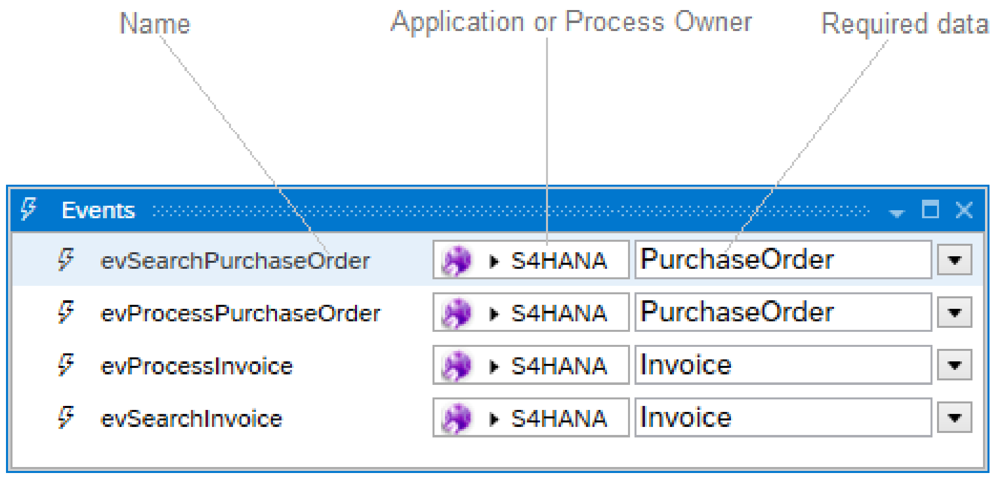
Definition:

A functional event:

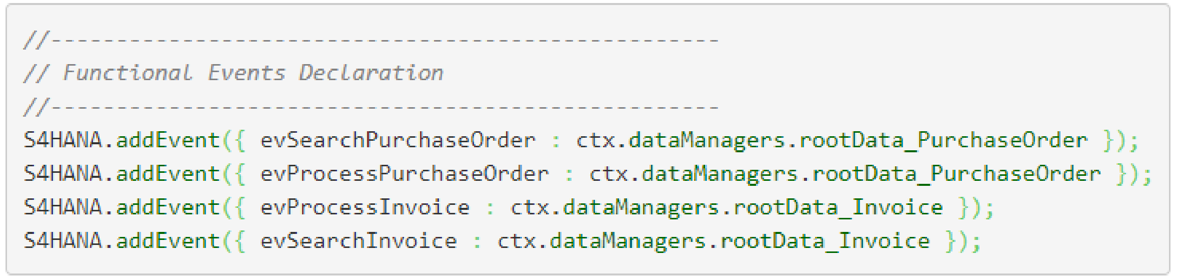
- A function called asynchronously
- Has a name
- Belongs to an entity: a process or an application
- Can have input data: a string or a JavaScript object

Its mechanism implies:

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



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 });
```

Editor – Custom Development

Functional events

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) {
        ...
});
```

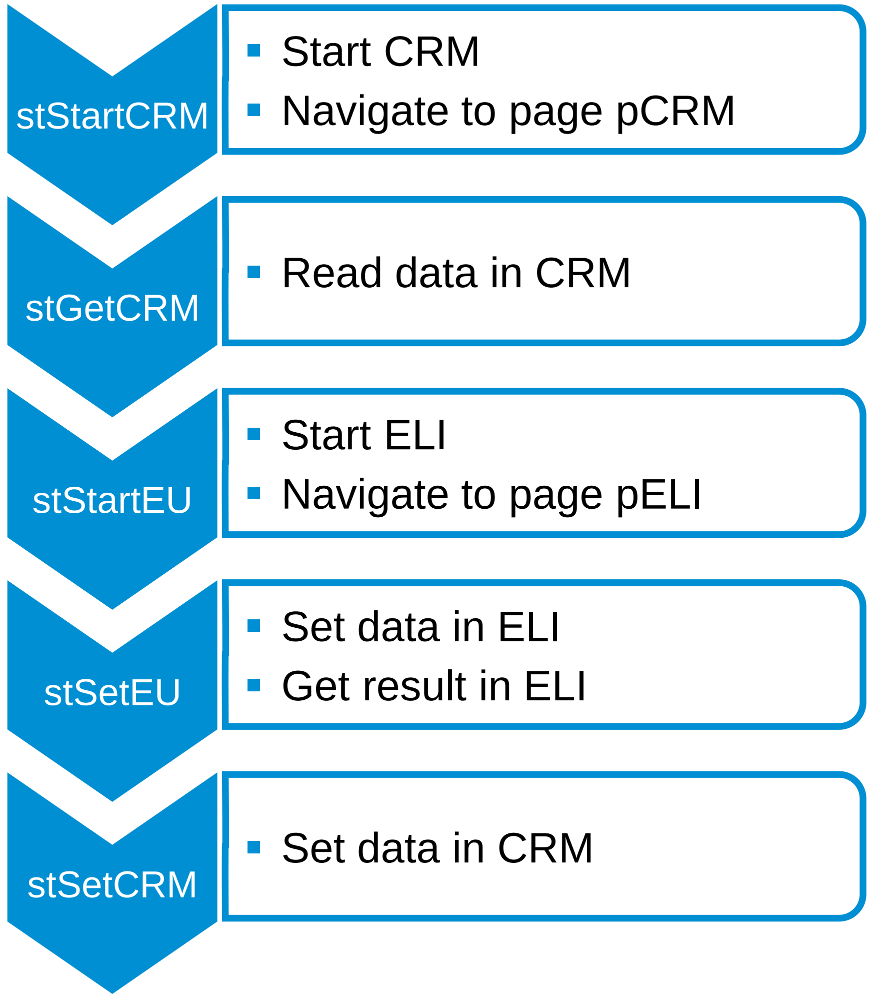
```
//-------------------------------------------------
// persistent listening of 'LOAD' event of the page 'pSearchPurchaseOrder'
//-------------------------------------------------
S4HANA.pSearchPurchaseOrder.events.LOAD.on(function(ev) {

  // notify an event to a 'S4HANA' application
  GLOBAL.notify(S4HANA.events.evSearchPurchaseOrder, 'PurchaseOrder');

});
```

Editor – Custom Development
Scenarios

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

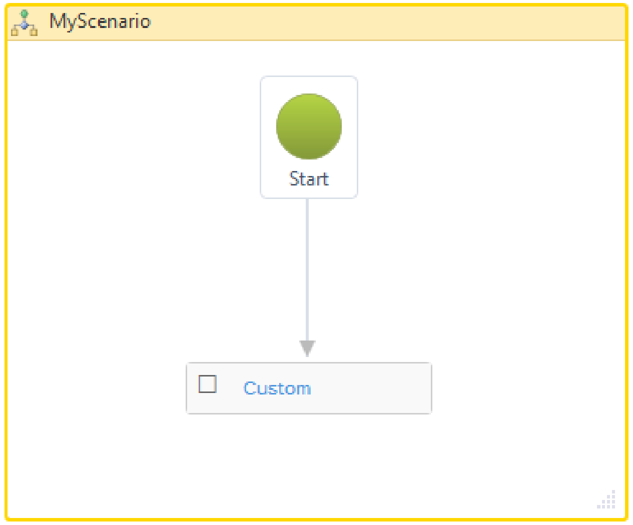


Editor – Custom Development

Scenarios

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: MySceanrio
// ----------------------------------------------------------------
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

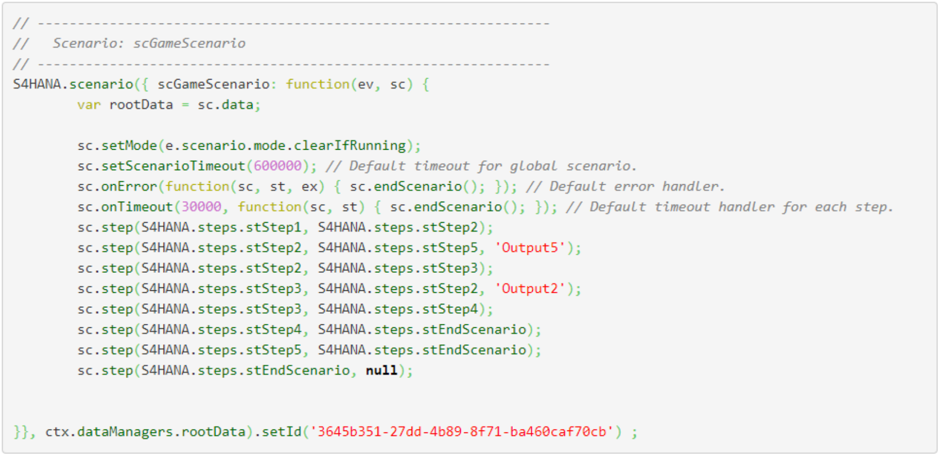
  Step3 → Step2 [Output2]

  Step3 → Step4

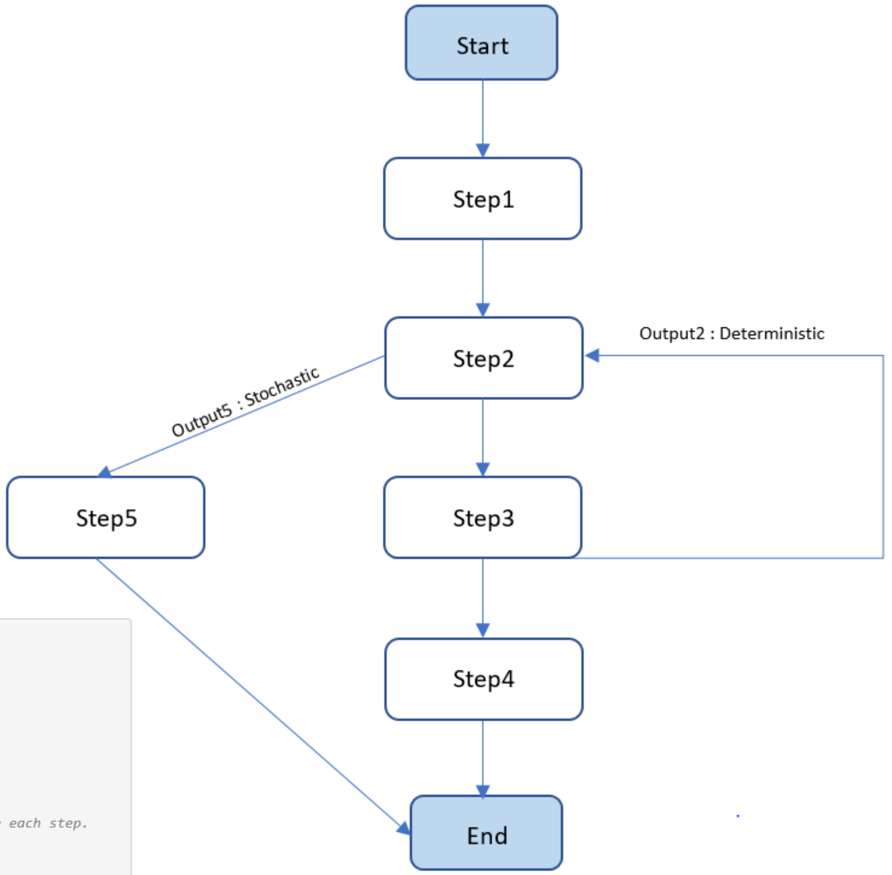
  Step4 → End Scenario

  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') ;
```



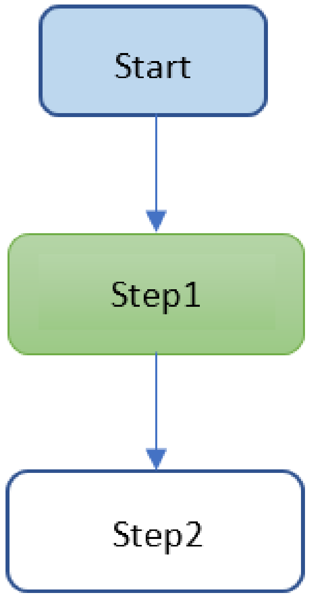
Editor – Custom Development

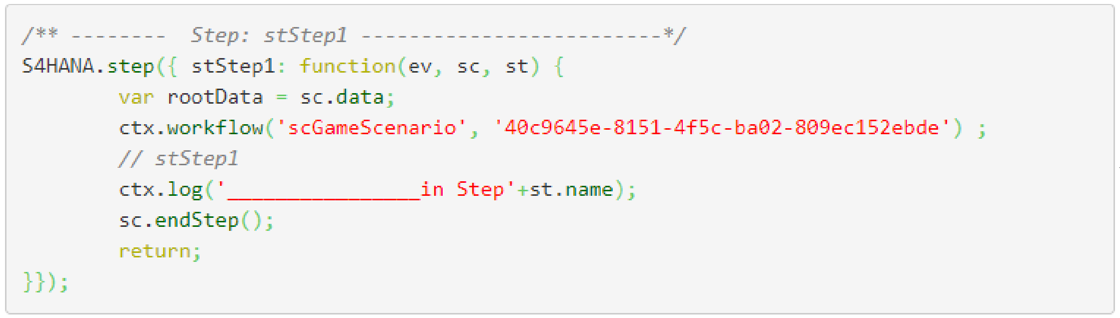
Steps

Definition:

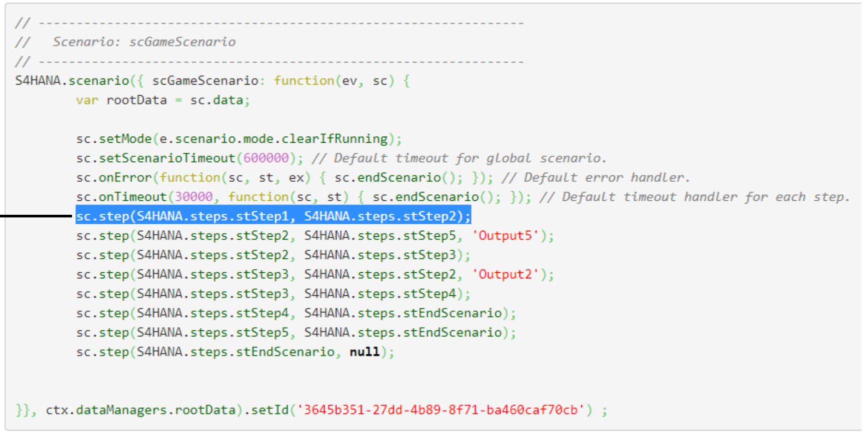
A step:

- 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;
}});
```

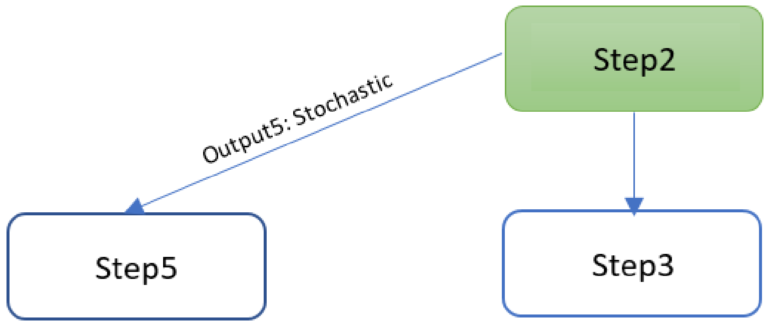


```
// ----------------------------------------------------------------
//   Scenario: scGameScenario
// ----------------------------------------------------------------
S4HANA.scenario({ scGameScenario: function(ev, sc) {
        var rootData = sc.data;

        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') ;
```

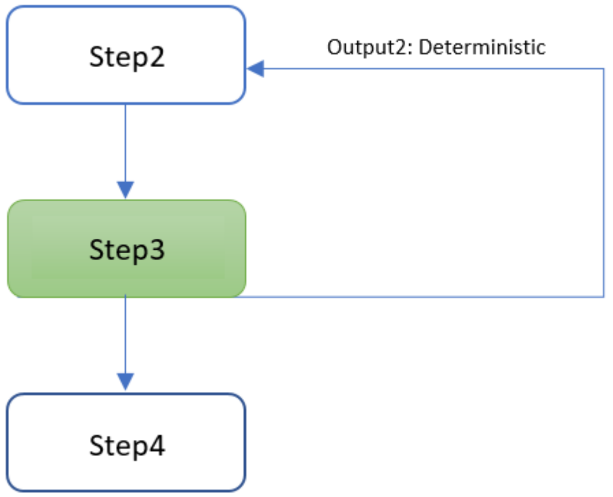
Editor – Custom Development

Steps

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;
}});
```

Editor – Custom Development

Steps

Additional patterns:

Wait Multiple:

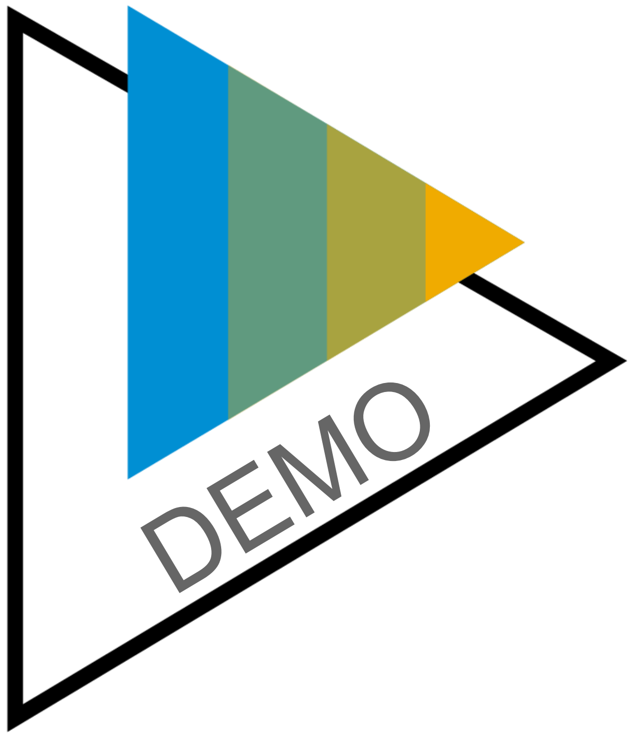
- Wait for pages
- Wait for buttons
- Wait for functional events
- Wait for timers and pollings
- etc…

Editor – Custom Development

Demo

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

THE BEST RUN