*Hi my dear friends !!!*

What is SAPUI5 ??

As a UI5 developer you must and should understand these concepts very well.

**What is SAPUI5 ?**

* SAPUI5 is SAP’s Web User Interface (UI) technology. It is also known as SAP UI Development Toolkit for HTML5.
* SAPUI5 is a collection of libraries that developers can use to build desktop and mobile applications that run in a browser.

**Advantages of  SAPUI5**

It supports for all types of platforms from Mobile, tablet to big Desktop.

* Easy to integrate with SAP ABAP Application Servers like ECC, SCM, SRM, CRM.
* Easy to access data from the server in the form of JSON or XML.
* Provides the developer s with a rich set of Controls to develop responsive web sites. It provides features to develop Rich Internet Applications (RIA).
* SAPUI5 can work together with Open-Ajax and Standard JavaScript Libraries.
* It supports CSS3 for custom themes, look and feel of web UI.
* SAPUI5 uses jQuery Library (which is open source) as a foundation.

**SAPUI5  Runtime**

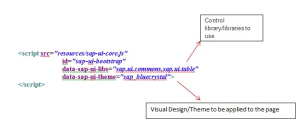
The SAPUI5 runtime is a client-side HTML5 rendering library with a rich set of standard and extension controls.

When we develop SAPUI5 applications with SAPUI5 application development tools, the code completion and application preview features are based on SAPUI5 runtime libraries installed in our local Eclipse. However, after we have deployed the application to the ABAP server and execute it there, it will use the SAPUI5 runtime libraries installed on the ABAP server.

To work with SAPUI5 we can include a set of JavaScript libraries into the HTML page (usually index.html page). After that we can use all controls provided by these libraries to construct our application.

**Bootstrap (Initialize and Loading SAPUI5)**

For loading SAPUI5, its bootstrap needs to be included with a <script> .

[](https://sapui5blogs.files.wordpress.com/2015/09/boot.png)

**Meta Tags**

There are two meta tags at the beginning of the <head> tag of theindex.html file.

1. i) The first meta tag is used to ensure that Internet Explorer 8+ uses its most standard- compliant rendering mode.

**<meta http-equiv=*“X-UA-Compatible”*content=*“IE=edge”*>**

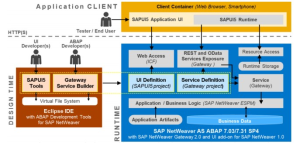
1. ii) The second meta tag is used to let all browsers treat the file as UTF-8 encoded (assuming that you use this encoding when editing/saving the file).

**<meta http-equiv=*‘Content-Type’*content=*‘text/html;charset=UTF-8’*/>**

**SAPUI5 Application Architecture**

The Complete application development architecture can be broadly subdivided into two types:

1. Design Time
2. Run Time

[](https://sapui5blogs.files.wordpress.com/2015/09/architecture.png)

**Design Time**

* The application is developed using an Eclipse IDE with ABAP Development Tools for Net-Weaver. (We can also develop sapui5 applications with other IDE or even with Notepad)
* The Eclipse IDE will provide the development environment for the SAPUI5 application.

**Plugins:**UI5 Application Development Plugin, UI5 Control Development Plugin, UI5

Repository Team Prov IDE Plugin.

* **SAP NetWeaver Gateway Service Builder**(transaction SEGW)) is a design-time environment. It provides developers an easy-to-use set of tools for creating services. The Code-based OData Channel consumes it and supports developers throughout the development life cycle of a service.

Service Builder provides an OData-compliant modeling environment for creation and maintenance of OData services.

All data access logic and data manipulation logic will be implemented in different methods

(Read,Query,Create,Update,Delete etc…) in Gateway Service builder.

**Run Time**

In order to fetch the data from SAP System and publishing these data to the outside world (users and devices, outside of the SAP Scope), the sapui5 application needs to call a service in Netweaver Gateway through a URI at runtime.

The Service URI would be accessible via the SAP NetWeaver Gateway which will act as an

**interface**between the SAP System and the user application.

Netweaver Gateway is a , license free interface based on the Atom Publishing format. The SAP NetWeaver Gateway interface has been implemented using the Open Data Protocol (OData).

**ICF (Internet Communication Framework)**

To execute the services available in Service builder of Net-weaver Gateway system, when the URL is called, the services must be activated in *SICF (Sap Internet Communication Framework).*

1. Code**:**SICF

In Gateway

Activate the ICF service in one of the following ways:

1. Using the menu option

Service/Host     Activate

2.Using the context menu and choosing Activate Service.

**Gateway Runtime Artifacts**

A Gateway Service is implemented by creating two ABAP classes – a Model Provider class and a Runtime Data Provider class. Once configured to function together, these two classes form a **Gateway Service.**

1. model provider class (MPC)
2. data provider class(DPC)

**REST and OData**

The OData Protocol is an application-level protocol for interacting with data via RESTful web services.

The OData protocol supports the description of data models and the editing and querying of data according to those models. It provides facilities for:

Metadata: a machine-readable description of the data model exposed by a particular data provider.

Data: sets of data entities and the relationships between them.

Querying: requesting that the service perform a set of filtering and other transformations to its data then return the results.

Editing: creating, editing, and deleting data. Operations: invoking custom logic.

The OData Protocol is different from other REST-based web service approaches because it describes both the data and the data model.

Representational State Transfer (REST) is a software architecture which is used for creating web services.

**SAPUI5 and MVC Architecture**

SAPUI5 is a client UI technology based on JavaScript, CSS and HTML5. SAPUI5 applications run in a browser.

Servers come into play for deploying your applications, storing the SAPUI5 libraries and connecting to a database. Depending on the environment in which SAPUI5 is used, the libraries or your applications are stored on an SAP NetWeaver Application Server or an SAP HANA Cloud Platform.

The preferred way to access business data for your application is using the OData model through a SAP NetWeaver Gateway.

When users access an SAPUI5 application from their device, a request is sent to the respective server to load the application into the browser. The view accesses the relevant libraries. Usually the model is also instantiated and business data is fetched from the database.

Model View Controller

The Model View Controller (MVC) concept is used in SAPUI5 to separate the representation of information from the user interaction. This separation facilitates development and the changing of parts independently.

* The view is responsible for defining and rendering the UI.
* The model manages the application data.
* The controller reacts to view events and user interaction by modifying the view and model.

The purpose of data binding in the UI is to separate the definition of the user interface (view), the data visualized by the application (model), and the code for the business logic for processing the data (controller). The separation has the following advantages: It provides better readability, maintainability, and extensibility and it allows you to change the view without touching the underlying business logic and to define several views of the same data.

Views and controllers often form a 1:1 relationship, but it is also possible to have controllers without a UI, these controllers are called application controllers.

It is also possible to create views without controllers.

View and controller represent reusable units, and distributed development is highly supported.

* Models

A model in the Model View Controller concept holds the data and provides methods to retrieve the data from the database and to set and update data.

* views

The view in the Model View Controller concept is responsible for defining and renderign the UI. SAPUI5 supports predefined view types.

* Controller

SAPUI5 uses the controller to separate the view logic from the model logic.

* Data Binding

SAPUI5 uses data binding to bind two data sources or information sources together to keep them in synch: All changes in one data source are also reflected in the other data source.

SAPUI5 uses the model view controller (MVC) concept to achieve the following objectives:

* Support development in distributed teams with different source locations
* Propose file structure, naming, and usage patterns
* Add capability of UI declaration (in comparison to a programmatic construction).

——————————————————————————————————————–

**Basics of SAPUI5 Models**

SAPUI5 Model instances hold the data and provide the methods required to set the data or to retrieve data from a server.

SAPUI5 supports the following types of models:

1. JSON Model
2. OData Model
3. XML Model
4. Resource model

The JSON model, XML model, and the resource model are client-side models, meaning that the model data is loaded completely and is available on the client. The OData model is a server-side

model and only loads the data requested by the user interface from the server. Any changes in data binding or list operations require a new request to the server.

You can also define nested models, for example, a JSON model defined for the application and an OData model for a table control contained in the application.

**JSON Model**

JSON Stands for JavaScript Object Notation which is a commonly used data interchange format. The JSON model is a client-side model because dataset will be completely available on the client

side.

It is intended for small datasets. It supports two-way binding.

Syntax of creating a JSON data model:-

**var jsonModel = newsap.ui.model.json.JSONModel();**

For setting the data to the model we use the method    setData().

If you use a JSON model for data binding, sorting and filtering is implemented in JavaScript because the data is available on the client.

**OData Model**

Odata model is used for binding the data that come from Odata services.

The OData model is a server-side model because the dataset is only available on the server and the client only knows the currently visible rows and fields.

Syntax of creating an OData model.

**var oModel = newsap.ui.model.odata.ODataModel(“<serviceL>”);**

The service URL is the first parameter of the constructor.

One OData model instance can only cover one OData service. If you need access to multiple services, you have to create multiple instances of the OData model.

Sorting and filtering is done on the server.

SAPUI5 supports the following important operations and features for the OData model:

* write
* create
* remove
* update
* read
* XSRF Token
* refresh

The default binding mode issap.ui.model.BindingMode.OneWay.

You can also set the binding mode tosap.ui.model.BindingMode.TwoWay(Recent Changes).

**Create operation**

The create function triggers a POST request to the OData service created. Example

var oEntry = {};  //Object declaration

oEntry.Name = “Rinku”;  //Assigning data to the object’s Name property

oEntry.Age = “25”;

oModel.create(‘/Employee’, oEntry, null, function(){

alert(“Create successful”);

},function(){

alert(“Create failed”);});

After a successful request to create and update the bindings in the model, a refresh is triggered automatically.

**Update operation**

The update function triggers a PUT request to the OData service created. Example

var oEntry = {};

oEntry.Age = “26”;

oModel.update(‘/Employee(1)’, oEntry, null, function(){

alert(“Update successful”);

},function(){

alert(“Update failed”);});

After a successful request to update the bindings in the model, a refresh is triggered automatically.

**Delete Operation**

The remove function triggers a DELETE request to the OData service created. The application has to specify the path to the entry which should be deleted.

oModel.remove(‘/Employee(1)’, null, function(){

alert(“Delete successful”);

},function(){

alert(“Delete failed”);});

**Read Request**

The read function triggers a GET request to a specified path which should be retrieved from the

OData service which was specified during creation of the OData model.

oModel.read(‘/Employee(1)’, null, null, true, function(oData, oResponse){

alert(“Read successful: ” + JSON.stringify(oData));

},function(){

alert(“Read failed”);});

**Refresh**

The Refresh function triggers a refresh for each binding to check if a value has been updated, or not.

**XSRF**(Cross Site Request Forgery) Token

To address the challenge of Cross Site Request Forgery an OData service might require XSRF tokens for change requests by the client application for security reasons. In this case the client has to fetch a token from the server and send it with each change request to the server.

The OData model fetches the XSRF token when reading the metadata and then automatically sends it in each write request header. If the token is not valid any more a new token can be fetched by calling the refresh function on the OData model.

**XML Model**

In order to bind XML data to controls, XML model can be used.

It is a client-side model intended for small datasets, which are completely available on the client.

It supports **two way binding**.

Syntax  of Creating XML model instance is as below:

var oModel = newsap.ui.model.xml.XMLModel();

Sorting and filtering is implemented in JavaScript because all data is available on the client.

**Resource Model**

The resource model is used as a wrapper for resource bundles.  You can use the JavaScript modulejQuery.sap.resources to access localized texts.  The resource

bundle jQuery.sap.util.ResourceBundleprovides access to the localized texts that are contained in the resource bundle.

Example:

To bind texts of a control to language-dependent resource bundle properties.

1. With a bundleName ( the name of a resource bundle and equals a SAPUI5 module name within the require/declare concept ).
2. With a bundleUrl that points to the resource bundle

If you use the bundle name, the file must have the .properties suffix. If you do not specify a locale, the system uses the login language.

var oModel = newsap.ui.model.resource.ResourceModel({bundleName:”myBundle”,locale:”en”});

**Custom Model**

Custom models can be used if none of the models provided by SAPUI5 is suitable for the specific needs of an application.

———————————————————————————————————————-

**SAPUI5 Components**

Components are independent and reusable parts used in SAPUI5 applications. A component is simply a folder.

SAPUI5 provides the following two types of components:

* Faceless components (class:sap.ui.core.Component)

Faceless components do not have a user interface and are used, for example, for a service that delivers data from a back-end system.

* UI components (class:sap.ui.core.UIComponent)

UI components extend components and add rendering functionality to the component. They represent a screen area or element on the user interface, for example, a button or a shell, along with the respective settings and metadata.

The sap.ui.core.Component class is the base class for UI and faceless components and provides the metadata for both types of components.

**Different Design Patterns**

The below are the important design patterns:

**Master-Detail (MD)**

The Master-Detail (MD) design pattern is used to represent two levels of data hierarchy. Typically this will be a list of items, shown in the master, and information for a selected item shown in the detail.

This pattern is directly supported by **the sap.m.SplitApp**control, which itself inherits from the **sap.m.SplitContainer**control.

**Master-Master-Detail (MMD)**

This Master-Master-Detail (MMD) pattern is a variation on Master-Detail and is designed to allow an extra level of drill down through a three level data hierarchy.

**Full Screen**

Typically used in analytical applications and visualizations.

**Multi Flow**

This pattern is used when the app functionality is too complex for a single pattern. It consists of the Full Screen and Master-Detail patterns. It is a combination of known (and therefore expected) UI presentations.

————————————————————————————————————-

**Data Binding**

Association of data sources so that all changes in one data source are automatically reflected in the other data source is called data binding.

Usually, the UI uses data binding to bind UI controls to a data source that holds the application data, so that the controls are updated automatically whenever application data is changed.

**Achieving Data Binding**

* Create  a model instance.
* Create instance for your control
* Bind the properties or Lists to the model.

Models and data binding is used in SAPUI5 applications to connect snd synchronize two data or information sources.

**Binding Types**

**SAPUI5 supports three different binding types:**

1. a) Property binding,      b) Aggregation binding,      c) Element binding.

**a.) Property binding**

Once you have defined the property binding, the property is updated automatically every time the bound model property value is changed. Property binding allows properties of the control to get automatically initialized and updated from model data.

The following two options can be used to define a property binding on a control:

* In the settings object in the constructor of a control.
* Using the bindProperty method of a control.

Example-1

var oTextField = newsap.ui.commons.TextField({

value: “{/../..}”  //Give binding path

});

Example-2

var oTextField = newsap.ui.commons.TextField({

value: {   }

});

Example-3

path: “/path”,

mode:sap.ui.model.BindingMode.OneWay

oTextField.bindProperty(“value”, “/../..”);

Example-4

oTextField.bindProperty(“value”, {

path: “value”,

type: new sap.ui.model.type.Integer()

});

Example-5

oTextField.bindValue(“/path”);

When creating a PropertyBinding, you can specify a different binding mode, which is then used exclusively for this specific binding.

**b.)**A**ggregation binding,**

Aggregation binding is used to automatically create child controls according to model data. You can define aggregation binding :

* In the settings object in the constructor
* By calling the bindAggregation method.

Aggregation binding requires the definition of a template, which is cloned for each bound entry of the list.

The aggregated elements are destroyed and recreated whenever the bound list in the data model is changed.

To bind an aggregation, you create a template or provide a factory function, which is then passed when defining the aggregation binding itself.

**Example**

var oItemTemplate = newsap.ui.core.ListItem({text:”{property}”});

var oComboBox = newsap.ui.commons.ComboBox({

items: {

path: “/path”,

template: oItemTemplate

}});

You can also define the aggregation binding by using the bindAggregation method of a control:

oComboBox.bindAggregation(“items”, “/path” ,

newsap.ui.core.ListItem({text:”{name}”}));

Some controls have a typed binding method for aggregations that are likely to be bound by the application:

oComboBox.bindItems(“/../path”, oItemTemplate);

You can provide initial sorting and filtering when specifying the aggregation binding.:

var oSorter = newsap.ui.model.Sorter(“name”, true); // sort descending

var oFilter1 = newsap.ui.model.Filter(“name”,sap.ui.model.FilterOperator.StartsWith, “M”);

var oFilter2 = newsap.ui.model.Filter(“name”,sap.ui.model.FilterOperator.Contains, “Paz”);

var oFilter3 = newsap.ui.model.Filter(“name”,sap.ui.model.FilterOperator.BT, “A”,”G”); // name between A and G

1. **c) Element binding**

This is especially helpful in master/detail scenarios.  Element binding allows to bind elements to

a specific object in the model data, which will create a binding context and allow relative binding within the control and all of its children.

Element bindings can be defined relatively.  In case of relative element bindings, the binding context is updated when the parent binding context is changed.

To define an element binding, use the bindElement method on a control:

oControl.bindElement(“/..”);

oControl.bindProperty(“value”, “name”);

———————————————————————————————

**Binding Mode**

By default, all bindings of a model instance have the default binding mode of the model

The following modes are supported in sapui5:

**One Way**

The model implementation supports one way binding which means from model to view (value changes in the model will update all corresponding bindings and the view).

**Two Way**

The model implementation supports two way binding which means from model to view and from view to model (value changes in the model and in the view will update all corresponding bindings and the view, respective model).

**One Time**

The model implementation supports one time binding which means from model to view once. Example

var oText = newsap.ui.commons.TextField();

oText.bindValue(“/firstName”, null,sap.ui.model.BindingMode.OneWay);

————————————————————————————-

**Routing**

Routing gives the application programmer the ability to support navigation directly to this UI state.

With routing, and appropriate application logic, the UI state in the screenshot could be directly navigated to from an URL and each page can be independently bookmarked and accessed directly through a Navigation mechanism.

Routing can also be used in components.

The routing in components comprises four steps:

* Configuration
* Initialization
* Access
* View ownership

**Configuration**

To implement the navigation framework, configure and initialize the router. You can do this inside the component. First, define the routing metadata as follows:

metadata : {

routing: {

config: { // default values for routing viewType : “XML”,

viewPath: “default.path.view”,

clearTarget: false

},

routes: { // contains routing configuration objects

“myRouteName” : {

name : ,

pattern : “FirstView/{from}”,   view : “myViewId”

}

}

}

}

The routing metadata has two parameters:

* config contains the default values for every route; default values are overwritten by custom route values
* routes contains an array of all routes defined by the component.

**Initializing**

The router now needs to be initialized by the component, for example in the init function. init : function() {

**sap.ui.core.UIComponent.prototype.init.apply(this, arguments);**

// this component should automatically initialize the router!

**this.getRouter().initialize();**

…

}

**Access**

To access the router and to use its functions, use the getRouter() function or the

static getRouterFor function of the UI component. You can pass either a controller, or a view:

**var oRouter =sap.ui.core.UIComponent.getRouterFor(this);**

You can also use the **getRouter**function of your UI component.

All views that are generated by the router are automatically created in the context of the component.

———————————————————————————

**Fragments**

Fragments are light-weight UI parts which can be reused, defined similar to views, but do not have any controller or other behaviour code involved.

UI parts which are to be used in several views cannot be easily defined. They either have to be created as new controls, or they have to be created as views. Creating them as new controls results in a development overhead, while creating them as separate views results in a runtime overhead.

To solve these issues, fragments have been introduced. They can be reused and, if source code is required and for event handler methods, they can connect to existing controllers of the “owning” view. This means that one important characteristic of fragments is that they are independent of the MVC concept and can be used without using MVC. However, if fragments are used together with views and controllers, fragments can make use of them and integrate them neatly.

SAPUI5 provides different types of fragments as below:

* XML fragments
* HTML fragments
* JS fragments

The fragments simply end with \*.fragment instead of \*.view.