What is an interface?

In digital world, interface means a layer which is used for communication between 2 different platforms.

SAP achieves interfacing using multiple processes:

1. RFC enabled FMs
2. IDOCs
3. File method
4. SOAP web services
5. RESTful APIs
6. OODATA

The basic data transfer between two platforms could be classified into 4 database operations often known as CRUD/CURD:

Create /POST

Update / PUT

Read / READ

Delete / DELETE

For demo purpose, we would be using 2 different SAP servers where **Server P** will be service provider and **Server C** will be Service Consumer:

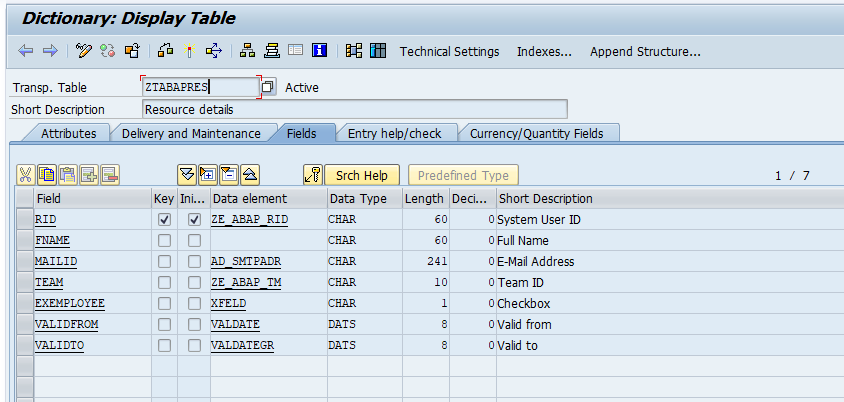
A **service provider** is such a server which exposes its database to the outside world for DB operations and such exposure is done in forms of Web-Service.

A **service consumer** is such a server which consumes such exposed web-services to perform DB operations on the foreign server remotely.

**Objects created in server P**:

1. **Database table**:

We have created a table in SAP **server P** which contains the basic employee details:



1. **An RFC enabled function module with all the DB operations**:

For demo purpose, we have created following RFC enabled FM in server P.

Code:

|  |
| --- |
| FUNCTION z\_fm\_dbops. *\*"----------------------------------------------------------------------* *\*"\*"Local Interface:* *\*"  IMPORTING* *\*"     VALUE(IV\_ACTION) TYPE  STRING OPTIONAL* *\*"     VALUE(IV\_FIELDNAME) TYPE  STRING OPTIONAL* *\*"  EXPORTING* *\*"     VALUE(EV\_SUCCESS) TYPE  STRING* *\*"     VALUE(EV\_MESSAGE) TYPE  STRING* *\*"  CHANGING* *\*"     VALUE(CW\_RES) TYPE  ZTABAPRES* *\*"----------------------------------------------------------------------*   DATA:   lv\_set\_command TYPE string,   lv\_val         TYPE string.    SELECT SINGLE \*     FROM ztabapres     INTO cw\_res     WHERE rid = cw\_res-rid.   IF sy-subrc IS INITIAL AND iv\_action = 'POST'.     ev\_success = 'false'.     ev\_message = 'Entry already exists'.     RETURN.   ELSEIF sy-subrc IS NOT INITIAL.      ev\_success = 'false'.     ev\_message = 'Entry does not exist'.     RETURN.    ELSE.     CASE iv\_action.       WHEN 'GET'.        WHEN 'PUT'.          lv\_val = 'CW\_RES-' && iv\_fieldname.          CONCATENATE         iv\_fieldname         '='         lv\_val         INTO lv\_set\_command SEPARATED BY space.          IF lv\_set\_command IS NOT INITIAL.           UPDATE ztabapres           SET (lv\_set\_command)           WHERE rid = cw\_res-rid.         ENDIF.        WHEN 'POST'.         INSERT ztabapres FROM cw\_res.       WHEN 'DELETE'.         DELETE ztabapres FROM cw\_res.       WHEN OTHERS.         ev\_message = 'Incorrect action'.         RETURN.     ENDCASE.   ENDIF.   IF sy-subrc IS INITIAL.    SELECT SINGLE \*   FROM ztabapres   INTO cw\_res   WHERE rid = cw\_res-rid.      ev\_success = 'true'.     ev\_message = 'DB operation successful'.   ELSE.     ev\_message = 'DB operation failed'.   ENDIF.  ENDFUNCTION. |

1. **Service specific objects**:

Each kind of service may or may not need further objects. We will find out the same soon once we start discussing each of these services in details.

**REST API:**

An **API** is an application programming interface. It is a set of rules that allow programs to talk to each other. The developer creates the API on the server and allows the client to talk to it.

**REST** determines how the API looks like. It stands for “Representational State Transfer”. It is a set of rules that developers follow when they create their API. One of these rules states that you should be able to get a piece of data (called a resource) when you link to a specific URL.

For more details refer to attached document below:



**The Anatomy Of A Request**

It’s important to know that a request is made up of four things:

**The endpoint (URL + PARAMS )**

**The method (GET/PUT/PATCH/POST/DELETE)**

**The headers(CONTENT-TYPE/AUTH/**

**The data (or body)**

**Format of Data Transfer(Media type):**

The format in which data can be transferred from client to Server and from server to client are of the following two types:

application/xml:

Extensible Markup Language (XML) is a [mark-up language](https://en.wikipedia.org/wiki/Markup_language) that defines a set of rules for encoding [documents](https://en.wikipedia.org/wiki/Electronic_document) in a [format](https://en.wikipedia.org/wiki/File_format) that is both [human-readable](https://en.wikipedia.org/wiki/Human-readable_medium) and [machine-readable](https://en.wikipedia.org/wiki/Machine-readable_data). In simple words XML is information wrapped in tags.

Sample XML:

*<object>*

*<EBELN> ”45000001”</EBELN>*

*<BUKRS>: ”8000”</BUKRS>*

*<array name = “ITEM”>*

*<object>*

*<EBELN> ”450000001”</EBELN>*

*<EBELP> ”00001”</EBELP>*

*<WERKS> ”ABCD”</WERKS>*

*<MATNR> ”ARTICLE1”</MATNR>*

*</object>*

*<object>*

*<EBELN> ”450000001”</EBELN>*

*<EBELP> ”00002”</EBELP>*

*<WERKS> ”ABCD”</WERKS>*

*<MATNR> ”ARTICLE2”</MATNR>*

*</object>*

*</array>*

*<object>*

application/json:

**JSON** (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

JSON is built on two structures:

* A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
* An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

Sample JSON(PO data):

*{*

*“EBELN”: ”45000001”,*

*“BUKRS”: ”8000”,*

*“ITEM”: [*

*{*

*“EBELN”: ”450000001”*

*“EBELP”: ”00001”,*

*“WERKS”: ”ABCD”,*

*“MATNR” : ”ARTICLE1”*

*},*

*{*

*“EBELN”: ”450000001”*

*“EBELP”: ”00002”,*

*“WERKS”: ”ABCD”,*

*“MATNR” : ”ARTICLE2”*

*}*

*]*

*}*

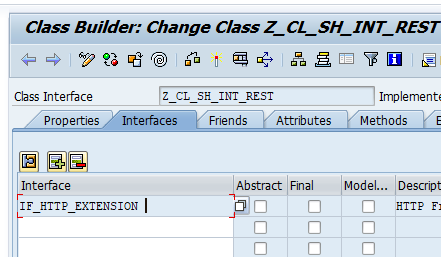
In simple terms, ‘{ }’ represents dictionary which corresponds to work area in an ABAP program and [] represents list which corresponds to internal table in an ABAP program.

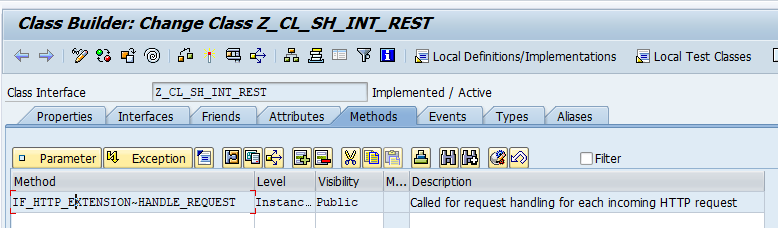
The above example can be mapped in ABAP with a deep structure LW\_PORDER as declared below:

  TYPES:  
  BEGIN OF ty\_item,  
  ebeln TYPE char10,  
  ebelp TYPE char5,  
  werks TYPE char4,  
  matnr TYPE char18,  
  END OF   ty\_item,  
  
  BEGIN OF ty\_porder,  
  ebeln TYPE char10,  
  bukrs TYPE char4,  
  item  TYPE STANDARD TABLE OF ty\_item WITH NON-UNIQUE DEFAULT KEY,  
  END OF   ty\_porder.  
  
  DATA: **lw\_porder** TYPE **ty\_porder**.

For creating a REST API in SAP, following objects are required to be created

1. A custom class implementation of interface ‘IF\_HTTP\_EXTENSION’ with method implementation for IF\_HTTP\_EXTENSION~HANDLE\_REQUEST

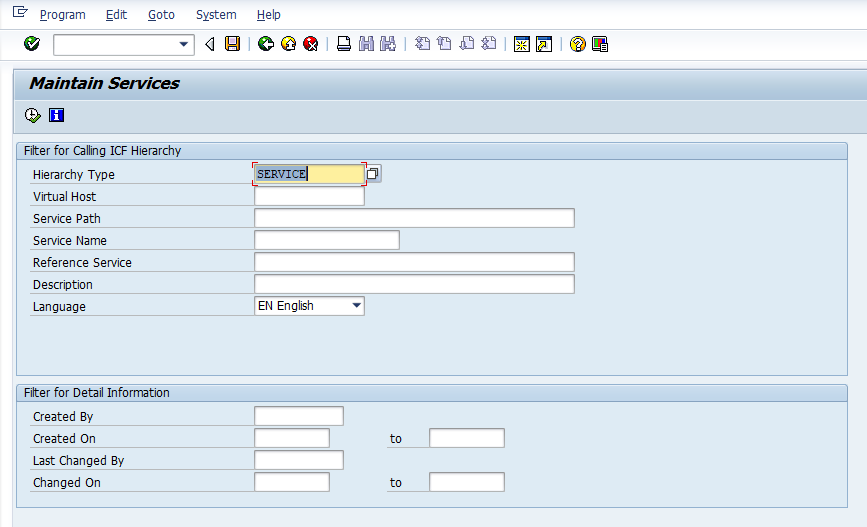




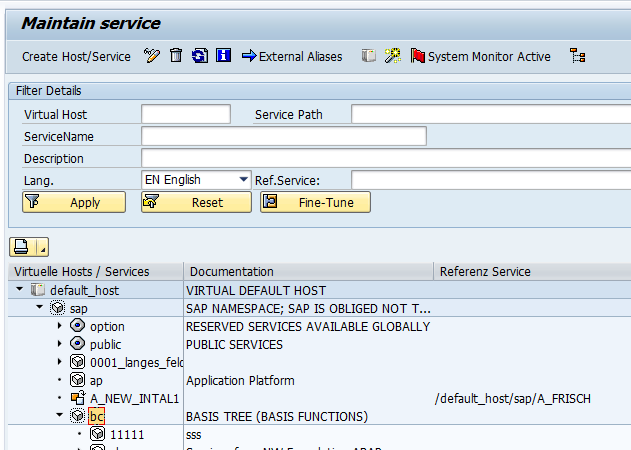
**Code:**

|  |
| --- |
| METHOD if\_http\_extension~handle\_request.    TYPES:   BEGIN OF ty\_response,   success TYPE string,   msg     TYPE string,   data    TYPE ztabapres,   END OF   ty\_response.    DATA:   lo\_json\_ser  TYPE REF TO cl\_trex\_json\_serializer,   lo\_json\_dsr  TYPE REF TO cl\_trex\_json\_deserializer,   lw\_response  TYPE ty\_response,   lv\_action    TYPE string,   l\_json       TYPE string,   lv\_fieldname TYPE string,   lt\_dd03l     TYPE STANDARD TABLE OF dd03l,   lw\_dd03l     TYPE dd03l,   lw\_res       TYPE ztabapres,   lv\_path      TYPE string,   lt\_header    TYPE tihttpnvp,   lv\_req\_json  TYPE string.    FIELD-SYMBOLS: <lfs\_field> TYPE any.    lv\_req\_json = server->request->get\_cdata( ).   CREATE OBJECT lo\_json\_dsr.   lo\_json\_dsr->deserialize( EXPORTING json = lv\_req\_json IMPORTING abap = lw\_response-data ).    lv\_action = server->request->get\_header\_field( name = '~request\_method' ).    IF lv\_action = 'PUT'.     lv\_fieldname = server->request->get\_form\_field( name = 'FIELDNAME' ).   ENDIF.    CALL FUNCTION 'Z\_FM\_DBOPS'     EXPORTING       iv\_action    = lv\_action       iv\_fieldname = lv\_fieldname     IMPORTING       ev\_success   = lw\_response-success       ev\_message   = lw\_response-msg     CHANGING       cw\_res       = lw\_response-data.    CREATE OBJECT lo\_json\_ser     EXPORTING       data = lw\_response.    lo\_json\_ser->serialize( ).   l\_json = lo\_json\_ser->get\_data( ).   server->response->set\_cdata( data = l\_json ). ENDMETHOD. |

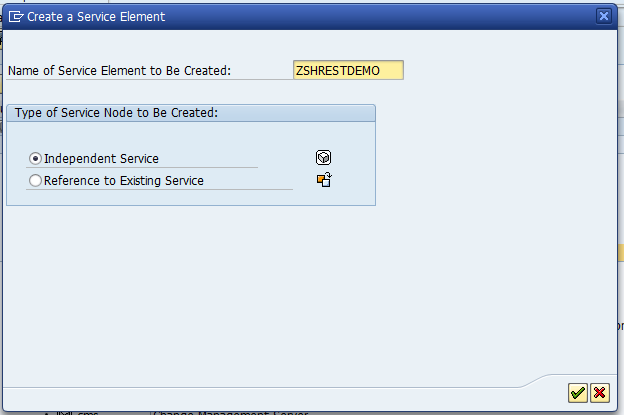
1. **SICF configuration:**
2. Run t-code SICF



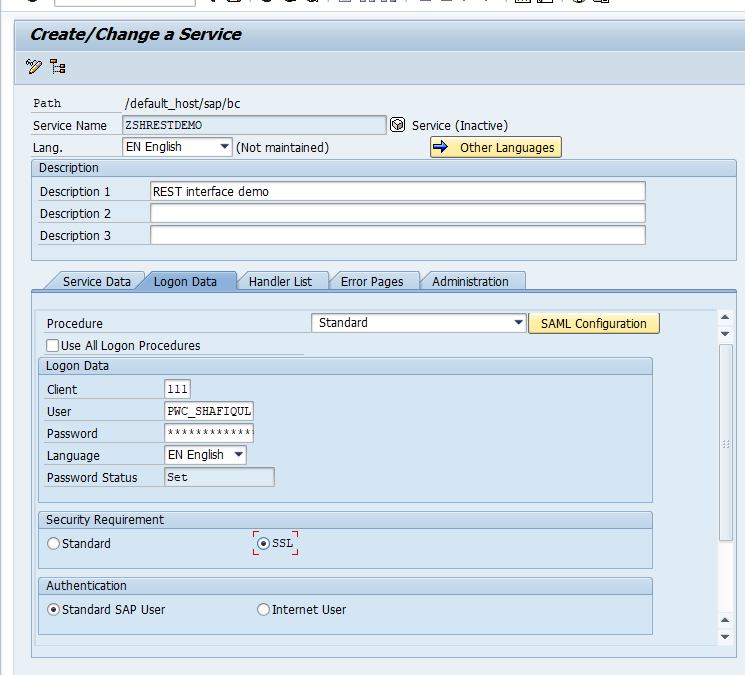
1. Execute with Hierarchy type as “SERVICE”
2. Navigate to default\_host/sap/bc



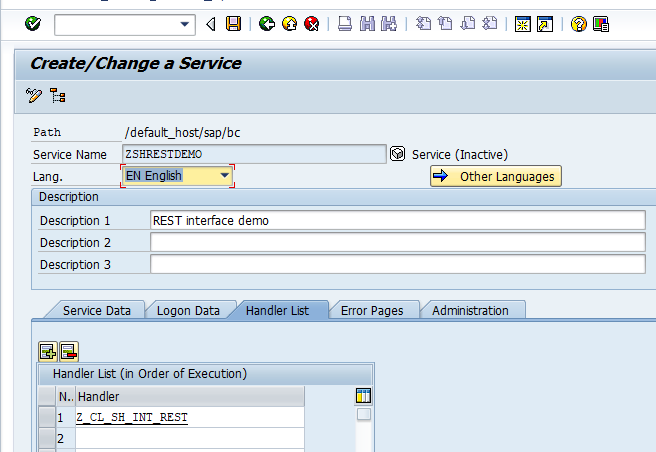
1. Right click on “bc” and click on “New sub-element”
2. Click ok on information message window
3. Enter name of Service and select radio button “Independent Service”



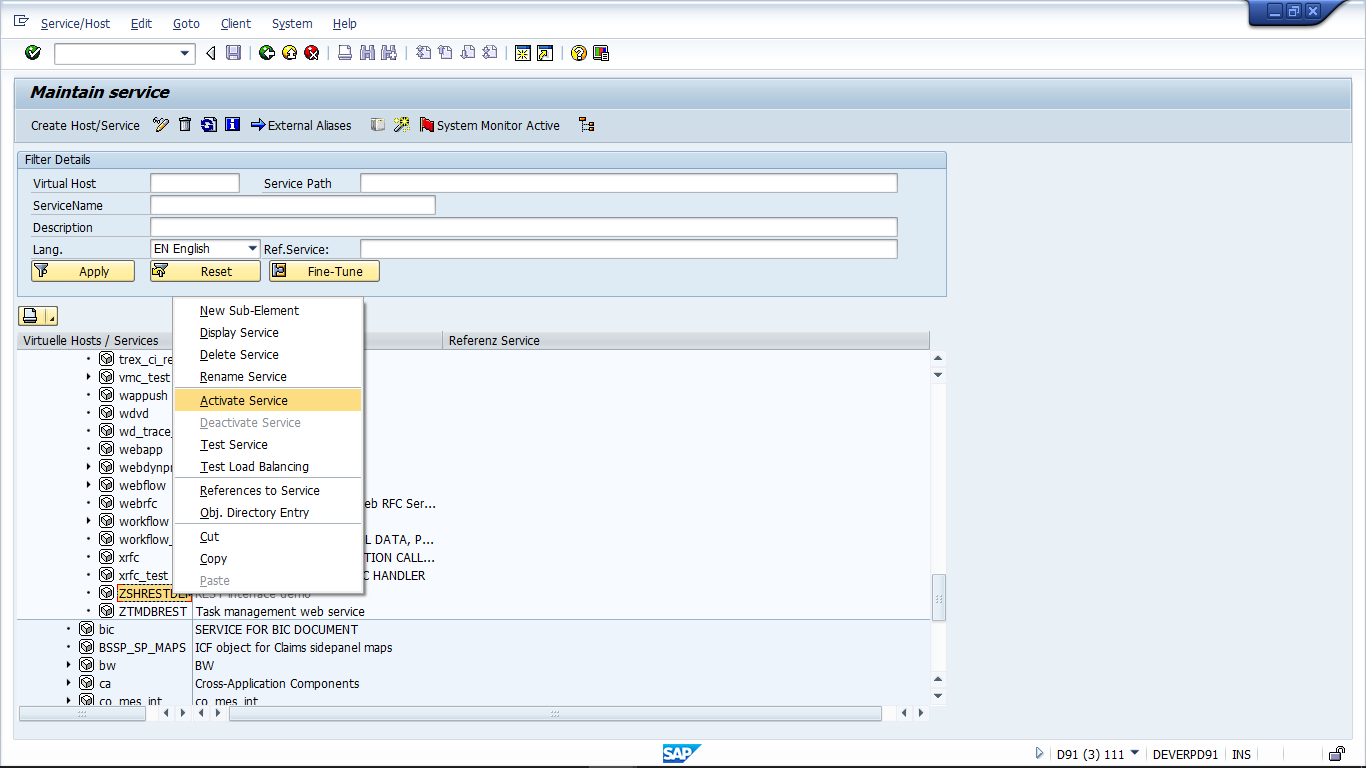
1. Enter description and logon details on next screen



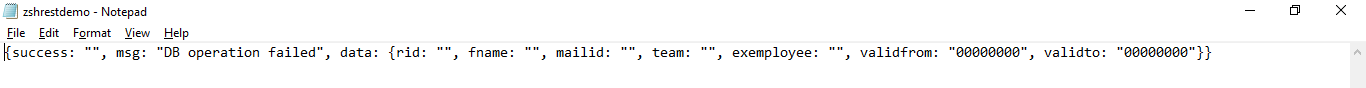
1. Enter implemented class name in Handlers List



1. Click on back and activate created sub-element



1. Right click on sub-element and click on Test service. It will result in a json file as below:



**Sanity Testing using Postman:**

Postman is tool provided by google to test APIs.

This can help us check whether our web-service can be consumed externally.

**Example :**

Refer program Z\_SH\_PROG\_REST\_DEMO in D90 for usage example.

Refer following document for detailed demo of using External REST APIs.

