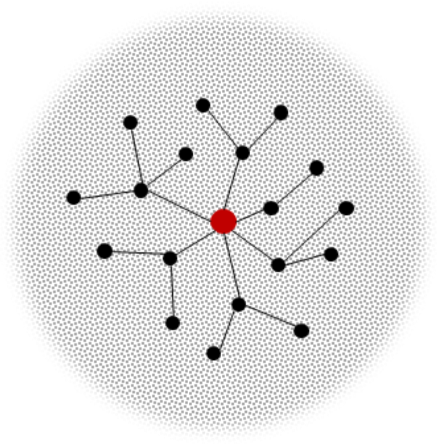


IBM Word-Wide
SAP Technical Enablement

SAP OData

Connecting from the IBM Cloud

Joachim Rese (rese@de.ibm.com)
IBM Germany Research & Development

Document Version Date: 2018-09-24

Table of Contents

Introduction

OData (Open Data Protocol) has been introduced by Microsoft for exchanging data over the Internet. The specification is an OASIS standard that has been approved by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

OData is built on the REST (Representational State Transfer) architecture which is based on the HTTP protocol. It supports all CRUD operations (Create, Read, Update, Delete). Furthermore, the OData specification includes filtering, projection and pagination as well as data relations (associations). The format of the request payload and response is either JSON or XML-based Atom/AtomPub.

SAP has implemented OData support in the SAP NetWeaver Gateway. Tooling for design, implementation, deployment and test of OData services are integral part of the SAP NetWeaver. In addition, SAP provides functions to expose BAPIs, Business Warehouse infocubes and CDS (Core Data Services) views by generated OData services.

With the increasing importance of application running on the cloud and the resulting need for an open, uniform connectivity to SAP systems, OData has become the method of choice for accessing an SAP system from the Internet.

SAP OData is relevant only if the SAP systems acts as a service or data provider. In contrast, SAP OData is not used when an SAP (ABAP) client application calls an IBM Cloud service.

This documentation explains how an OData service can be created and deployed in an SAP system. Furthermore, consumption of SAP OData services by application running on the IBM Cloud is covered. Code examples in node.js, node-RED and java are provided.

Architecture Overview

SAP OData implementation consists of multiple components.

SAP OData Components

Table 1 is list of SAP OData components and their functions.

Component	Function
OData Service	Receives and interprets incoming OData requests.
OData Runtime	Runs instances of MPC (Model Provider Class) and DPC (Data Provider Class) that provide meta data information, acquire data and perform requested operations on the data.
Backend	Data residence; provides access to the data.
SAP Gateway Client	Test tool for OData services.
Service Builder	OData service modeler.

Table 1: OData Components

The various OData components interact which each other as indicated by Figure 1.

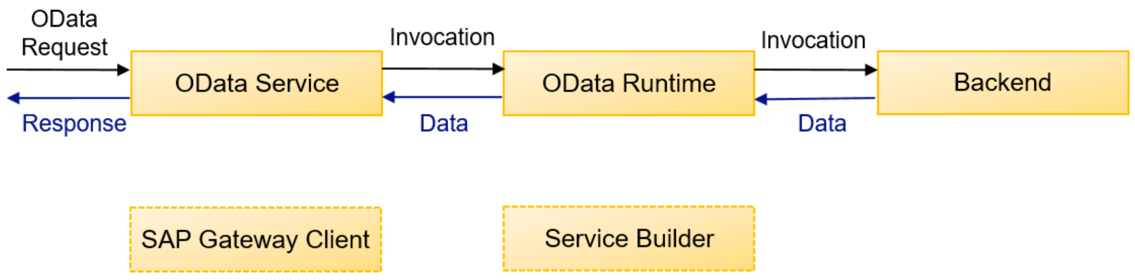


Figure 1: OData components

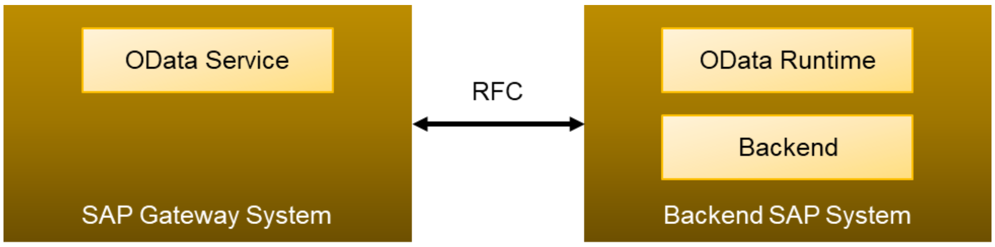
Up to SAP NetWeaver release 7.31 the OData and SAP Gateway functionalities are delivered with SAP software components `IW_FND`, `GW_CORE` and `IW_BEP`. Starting with SAP NetWeaver release 7.40 all functions are implemented in component `SAP_GWFND`, which is an integral part of the SAP NetWeaver.

SAP Gateway Deployment Options

There are multiple options for deploying OData services depending on where the individual OData components run.

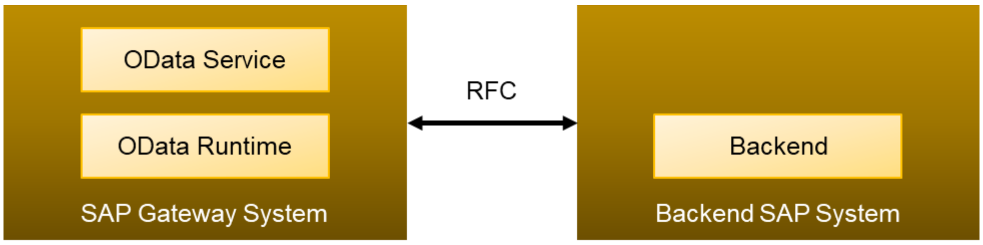
Scenario A: Central Hub

A single central SAP Gateway System connects to multiple backend SAP systems. This option allows good security and good performance since data and the OData Runtime reside on the same system.



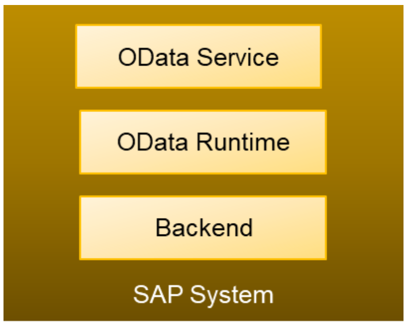
Scenario B: Central Hub with Deployment in SAP Gateway System

A single instance of the OData Runtime runs on the (only) central SAP Gateway System, which connects to multiple backend SAP systems. The SAP Gateway System can be located inside a demilitarized zone (DMZ). This option allows good security and low maintenance cost.



Scenario C: Embedded Deployment

All components are implemented on the backend SAP system. This option has lowest TCO since a dedicated gateway system is not needed. However, there might be limitations regarding security and scalability.



IBM Secure Gateway Client

To establish a communication path between the IBM Cloud and the SAP system which resides on the client’s premises, you must implement a secured connection. Deploy the IBM Secure Gateway to establish a VPN (Virtual Private Network) like illustrated by Figure 2.

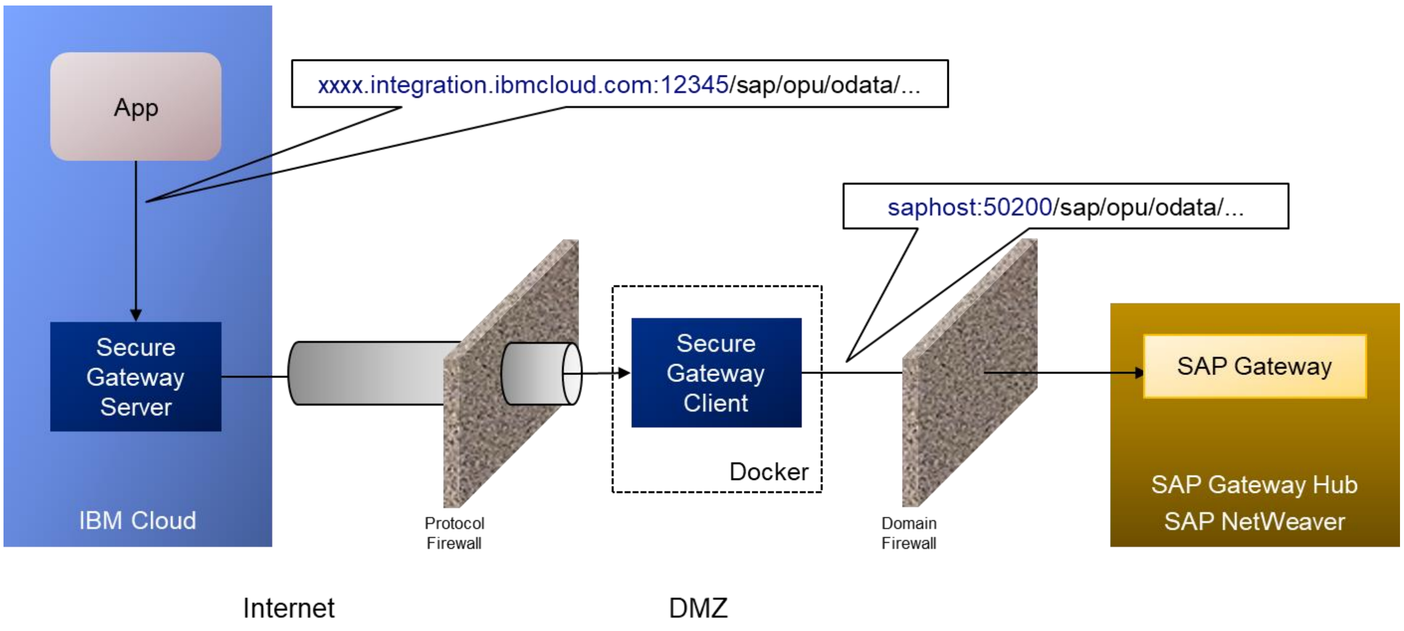


Figure 2: IBM Secure Gateway

Setting up the IBM Secure Gateway is outside the scope of this documentation.

Configuring OData Services

The SAP Gateway must be configured to receive and execute OData requests.

Enabling ICM Services

OData operates on the HTTP or HTTPS protocol. Therefore, the corresponding services must be enabled in the Internet Communication Manager (ICM).

You should set the following SAP profile parameters.

```
icm/server_port_0 = PROT=HTTP,PORT=<http port>,TIMEOUT=60,PROCTIMEOUT=600
icm/server_port_1 = PROT=HTTPS,PORT=<https port>,TIMEOUT=60,PROCTIMEOUT=600
```

Activating OData services

To activate OData services, proceed as follows.

1. Call transaction SICF.
2. Skip filter selection screen by clicking *Execute* (or press F8).
3. Navigate to tree node *default_host→ sap → opu → OData* if you want to activate all OData (V2) services. Alternatively, if you want to activate an individual service, navigate to the corresponding node deeper in the tree.
   Right-click on the OData service node and select *Activate Service* from context menu to activate all OData services.

CSRF (Cross-Site Request Forgery)

Cross-site request forgery (CSRF) is a technique where an attacker run malicious code against a remote site on which the user has previously been authorized. To protect from CSRF, SAP supports CSRF tokens. Such a token can be requested which a HTTP GET request and must be sent with each HTTP POST request. In addition to the CSRF token, a cookie containing a session ID is provided by the SAP system with every HTTP GET request. The session cookie must be sent with each HTTP POST request as well.

For easy prototyping it might be handy to initially ignore CSRF. In subsequent development CSRF token handling can be added to the application. To disable the use of CSRF tokens for a OData service, proceed as follow

1. Call transaction SICF.
2. Navigate to the OData service node *default_host→ sap → opu → OData → ... → <OData service>* and click the *Display/Change Service (F8)* on the toolbar or double-click the service name.
3. Click *Change (Ctrl+F1)* on toolbar.
4. On tab *Service Data* click button *GUI Configuration*.
5. Enter service parameter: `~CHECK_CSRF_TOKEN` (*Parameter Name*) and `0` (*Value*).
6. Press Enter to close the dialog screen.
7. Click *Store (Ctrl+S)*.

When CSRF is disabled, each REST request to the OData service should have the following HTTP header:

```
X-Requested-With = X
```

This (custom) header cannot be set cross-domain and therefore provides a minimum protection against CSRF.

Configuring the SAP Gateway

Perform the following steps to configure the SAP Gateway.

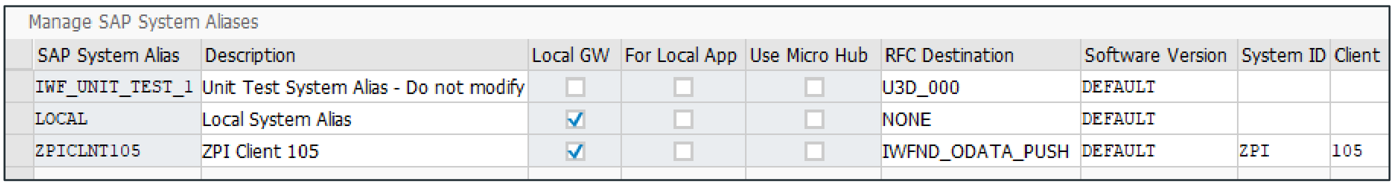
Activating the SAP Gateway

1. Call transaction SPRO and click *SAP Reference IMG*.
2. Right-click on node *SAP NetWeaver* → *SAP Gateway* → *OData Channel* → *Configuration* → *Activate or Deactivate SAP Gateway* and select *Edit Activity* from context menu.
3. If message in dialog reads "SAP Gateway is currently active", click *Cancel*. Otherwise click *Activate*.

Defining SAP System Alias for Backend System in Gateway System

1. Call transaction SPRO and click *SAP Reference IMG*.
2. Right-click on node *SAP NetWeaver* → *SAP Gateway* → *OData Channel* → *Configuration* → *Connection Settings* → *SAP Gateway to SAP System* → *Manage SAP Aliases* and select *Edit Activity* from context menu.
3. If the backend SAP system appears in the list, you are done. Otherwise:
4. Click *New Entries*.
5. Specify *SAP System Alias*, a *Description*, the *RFC Destination* (`NONE` for embedded deployment), the *System ID* and the *Client* of the backend SAP system. Check flag *Local GW* and enter `DEFAULT` for *Software Version*.
6. Click *Save (Ctrl+S)*.

Figure 3 shows example backend SAP system aliases.


Manage SAP System Aliases

SAP System Alias	Description	Local GW	For Local App	Use Micro Hub	RFC Destination	Software Version	System ID	Client
IWF_UNIT_TEST_1	Unit Test System Alias - Do not modify	☐	☐	☐	U3D_000	DEFAULT		
LOCAL	Local System Alias	✓	☐	☐	NONE	DEFAULT		
ZPICLNT105	ZPI Client 105	✓	☐	☐	IWFND_ODATA_PUSH	DEFAULT	ZPI	105

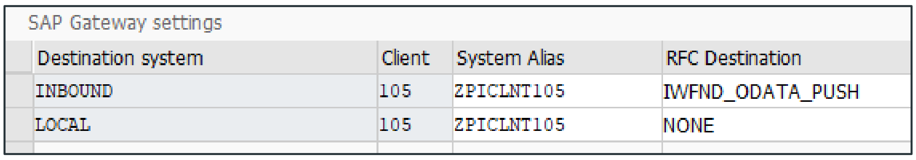
Figure 3: Backend SAP System Aliases

Setting up SAP Gateway in Backend System

1. Call transaction SPRO and click *SAP Reference IMG*.
2. Right-click on node *SAP NetWeaver* → *SAP Gateway Service Enablement* → *Backend OData Channel* → *SAP Gateway Settings* and select *Edit Activity* from context menu.
3. If the Gateway system appears in the list, you are done. Otherwise:
4. Click *New Entries*.
5. Specify *Destination System*, *Client*, *System Alias* and *RFC Destination*.

6. Click *Save (Ctrl+S)*.

Figure 4 shows example SAP Gateway settings.



SAP Gateway settings

Destination system	Client	System Alias	RFC Destination
INBOUND	105	ZPICLNT105	IWFND_ODATA_PUSH
LOCAL	105	ZPICLNT105	NONE

Figure 4: SAP Gateway settings

OData Concepts

The interface that is used with the OData protocol is specified by the Entity Data Model (EDM). The EDM defines an abstract model for communication and does not necessarily match the format of the persistency data layer.

Central EDM Elements

The central elements of the EDM are entity types, entities, entity sets and associations.

An entity type defines the data format as a set of properties. At least one property must be flagged as key. An entity type conceptually corresponds to a table type in the DDIC.

An entity is an instance of an entity type and is therefore by concept like a table record.

An entity set is a collection of entities which corresponds to a table.

An association defines a relation between entity types. The relation cardinality can be 1:m, n:1 or n:m. A navigation property is a special property of an entity type that is bound to an association.

Figure 5 gives an overview of the EDM elements and how they relate.

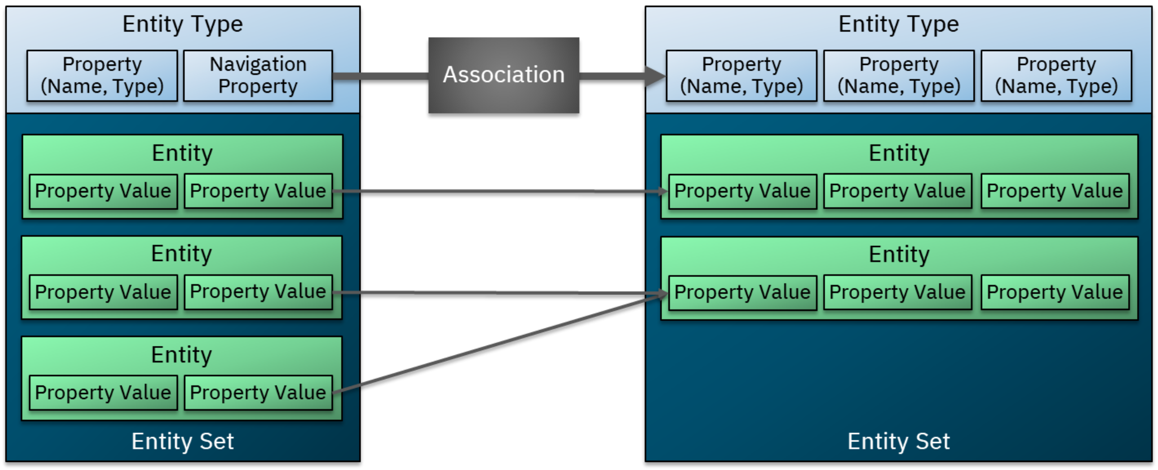


Figure 5: Entity Data Model Elements

OData Operations

OData uses REST requests to read, create, update and delete entities.

The REST request url starts with a service url that is constructed as follows.

```
https://<hostname>:<port>/<path to OData service>/<OData service name>
```

OData Operation	Request	URL
Get Metadata	GET	<service url>/`$metadata`
Read single entity	GET	<service url>/`<entity set>(<entity key>)`
Get number of entities	GET	<service url>/`<entity set>/$count`
Read some properties	GET	<service url>/`<entity set>?$select=prop1,prop2`

Read with filter	GET	<service url>/<entity set>?$filter=prop op value
Read sorted	GET	<service url>/<entity set>?$orderby=prop1,prop2
Read top n	GET	<service url>/<entity set>?$top=n
Read, follow navigation property	GET	<service url>/<entity set>?$expand=nav_prop
Read, skip first n entities	GET	<service url>/<entity set>?$skip=n
Create new entity	POST	<service url>/<entity set>
Update entity	PUT	<service url>/<entity set>(<entity key>)
Delete entity	DELETE	<service url>/<entity set>(<entity key>)

Table 2: OData Operations

Create (POST) and update (PUT) requests must provide the entity properties in the request body. On read requests, several options can be combined within the query string, for example

```
...?$filter=Name eq 'Tom'&$select=Name,Country&$orderby=Country
```

An individual OData service might not support all operations and options. If an unsupported operation or option is used, the corresponding REST request does not necessarily response an error.

Property names are case sensitive and start with a capital followed by small letters.

SAP OData Annotations

The OData protocol provides uniform metadata information about the services. This includes, for example, name and types of entity properties. Metadata can be access at:

Root level:	<service url>
Metadata document:	<service url>/$metadata

As an enhancement to the OData standard, SAP OData provides additional meta information:

Capacity annotations describe which operations can be performed on an entity. For example, if a property can be updated or whether filters are supported.

Label annotations give a language-dependent short description of an entity or property.

Semantic annotations provide usage information on an entity property. As an example, an annotation can tag a property as location or calendar event. Client application can use this information to seamlessly integrate the OData service.

SAP OData annotations are materialized as attribute to an XML / Atom tag, for example:

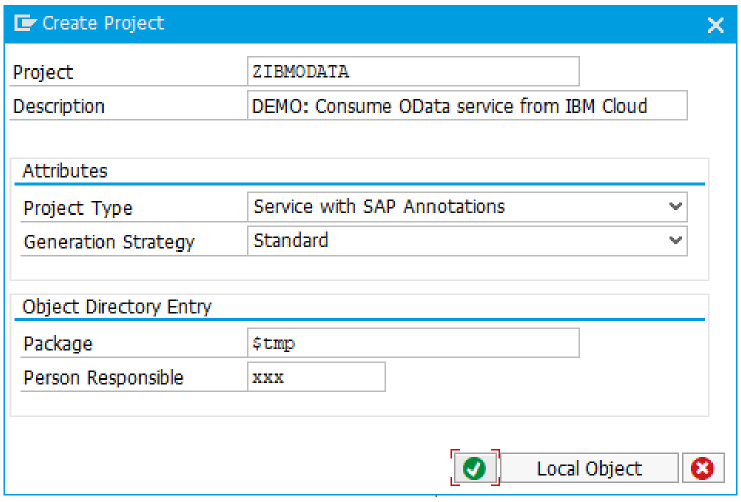
```
<Property Name="State" sap:filterable="false" sap:sortable="false" sap:updatable="false" sap:creatable="false"
                       sap:label="State of Residence" sap:semantics="region" ... />
```

Building an OData service

Throughout this documentation, you will model an OData service, deploy and test it and finally consume it by an application running on IBM Cloud.

Creating and deploying an OData service

1. Logon to the SAP system and call transaction `SEGW` to start the SAP Gateway Service Builder.
2. Select *Project* → *Create* to create a new project. Enter *Project* name (e.g. ZIBMODATA), a *Description* and the *Package* to which you want to assign the project or click *Local Object*.



Create Project

Project	ZIBMODATA
Description	DEMO: Consume OData service from IBM Cloud

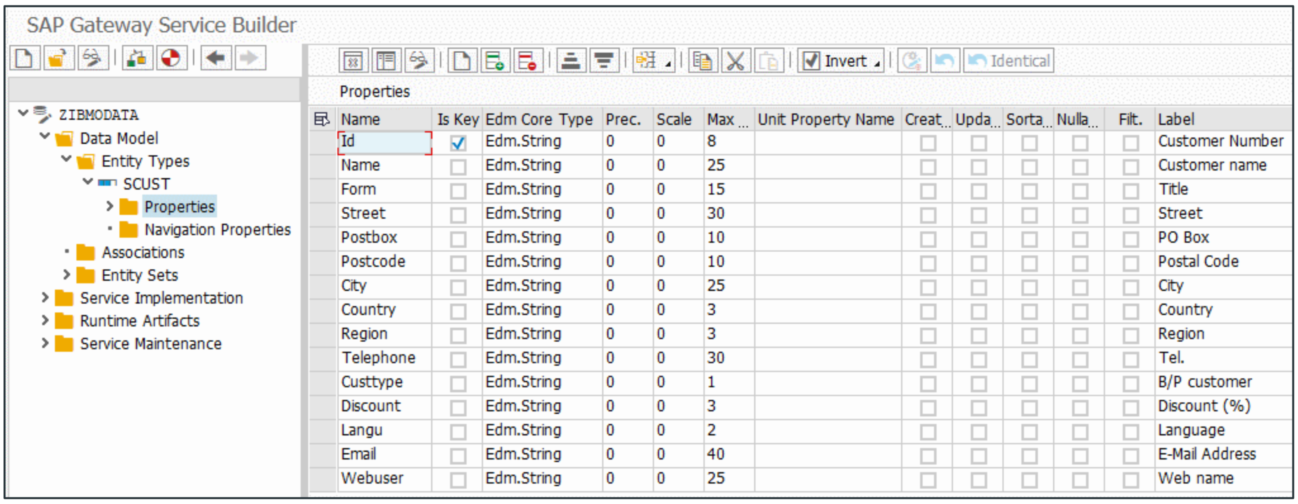
Attributes

Project Type	Service with SAP Annotations
Generation Strategy	Standard

Object Directory Entry

Package	$tmp
Person Responsible	xxx

3. Right click on *Data Model* and select *Import* → *DDIC Structure* from context menu.
4. Enter *Name* `SCUST` and *ABAP Structure* `SCUSTOM` and click *Next*.
5. Select all fields but `MANDT` and click *Next*.
6. Check field ID for *Key* and click *Finish*. You might experience warnings indicating unprecise mapping of data types. This is because DDIC type NUMC does not have an equivalent data type in the EDM. These warnings can be ignored here.



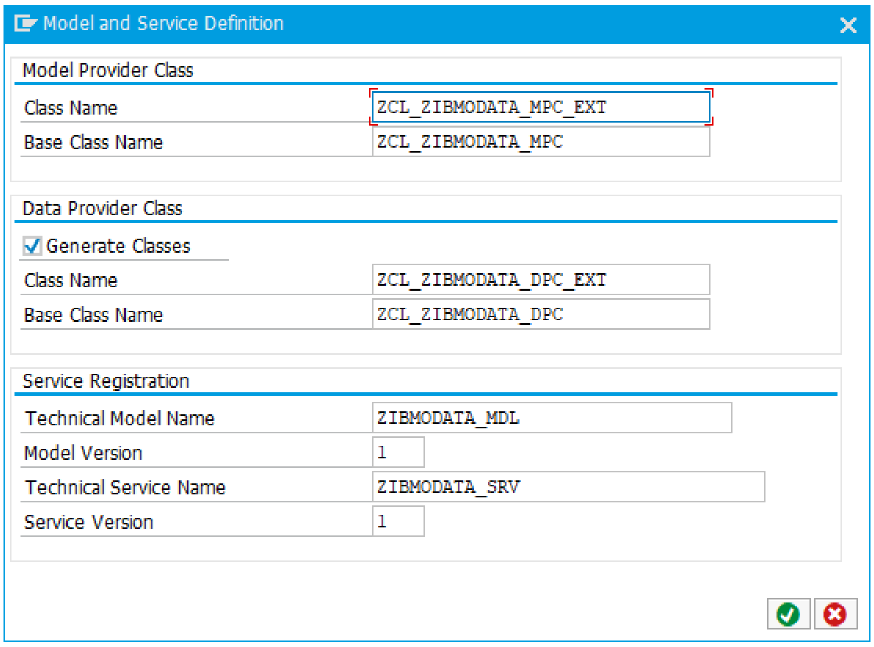
SAP Gateway Service Builder

Properties

Name	Is Key	Edm Core Type	Prec.	Scale	Max ...	Unit Property Name	Creat...	Upda...	Sorta...	Nulla...	Filt.	Label
Id	☑	Edm.String	0	0	8		☐	☐	☐	☐	☐	Customer Number
Name	☐	Edm.String	0	0	25		☐	☐	☐	☐	☐	Customer name
Form	☐	Edm.String	0	0	15		☐	☐	☐	☐	☐	Title
Street	☐	Edm.String	0	0	30		☐	☐	☐	☐	☐	Street
Postbox	☐	Edm.String	0	0	10		☐	☐	☐	☐	☐	PO Box
Postcode	☐	Edm.String	0	0	10		☐	☐	☐	☐	☐	Postal Code
City	☐	Edm.String	0	0	25		☐	☐	☐	☐	☐	City
Country	☐	Edm.String	0	0	3		☐	☐	☐	☐	☐	Country
Region	☐	Edm.String	0	0	3		☐	☐	☐	☐	☐	Region
Telephone	☐	Edm.String	0	0	30		☐	☐	☐	☐	☐	Tel.
Custtype	☐	Edm.String	0	0	1		☐	☐	☐	☐	☐	B/P customer
Discount	☐	Edm.String	0	0	3		☐	☐	☐	☐	☐	Discount (%)
Langu	☐	Edm.String	0	0	2		☐	☐	☐	☐	☐	Language
Email	☐	Edm.String	0	0	40		☐	☐	☐	☐	☐	E-Mail Address
Webuser	☐	Edm.String	0	0	25		☐	☐	☐	☐	☐	Web name

Entity type SCUST and entity set `SCUSTSet` have been created.

7. Select *Project* → *Generate*. Keep the suggested names for *Model Provider Class* and *Data Provider Class* and click *Enter*. When asked for *Object Directory Entry* attributes, click *Local Object* or specify *Package* and click *Save (Enter)*.



Model and Service Definition

Model Provider Class

Class Name	ZCL_ZIBMODATA_MPC_EXT
Base Class Name	ZCL_ZIBMODATA_MPC

Data Provider Class

☑ Generate Classes

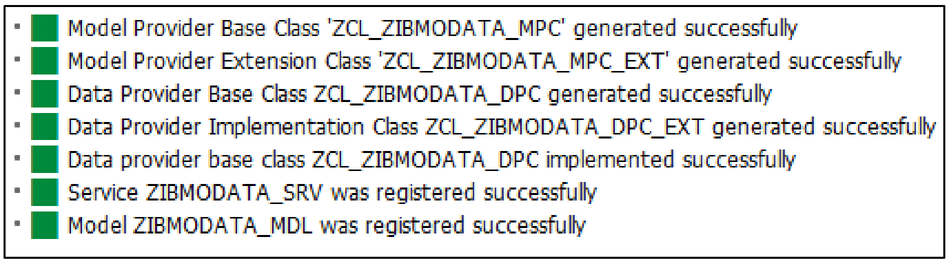
Class Name	ZCL_ZIBMODATA_DPC_EXT
Base Class Name	ZCL_ZIBMODATA_DPC

Service Registration

Technical Model Name	ZIBMODATA_MDL
Model Version	1
Technical Service Name	ZIBMODATA_SRV
Service Version	1

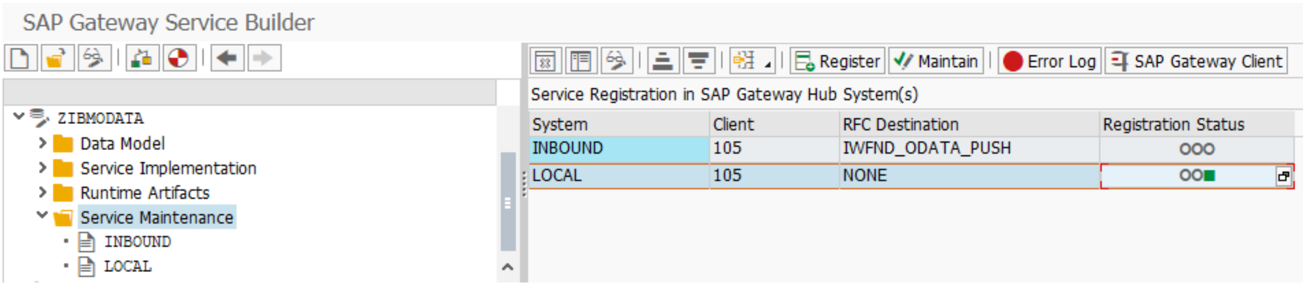
This step generates classes `[Z]CL_<Project name>_MPC` and `[Z]CL_<Project name>_DPC`. The methods of these classes implement the operation that the OData service provides.

Once the class generation is done, corresponding success messages are shown.



```
▪ 🟩 Model Provider Base Class 'ZCL_ZIBMODATA_MPC' generated successfully
▪ 🟩 Model Provider Extension Class 'ZCL_ZIBMODATA_MPC_EXT' generated successfully
▪ 🟩 Data Provider Base Class ZCL_ZIBMODATA_DPC generated successfully
▪ 🟩 Data Provider Implementation Class ZCL_ZIBMODATA_DPC_EXT generated successfully
▪ 🟩 Data provider base class ZCL_ZIBMODATA_DPC implemented successfully
▪ 🟩 Service ZIBMODATA_SRV was registered successfully
▪ 🟩 Model ZIBMODATA_MDL was registered successfully
```

8. Next you must register the OData service in the SAP Gateway. To do so, double-click on *Service Maintenance* and mark the RFC destination of the SAP Gateway. (Select *LOCAL* if the OData Runtime resides on the SAP Gateway system.) Select *Register* from tool bar. Assign package on appearing popup dialog and click *Enter*. A green traffic light should appear in column *Registration Status*.



Service Registration in SAP Gateway Hub System(s)

System	Client	RFC Destination	Registration Status
INBOUND	105	IWFND_ODATA_PUSH	○○○
LOCAL	105	NONE	○○🟩

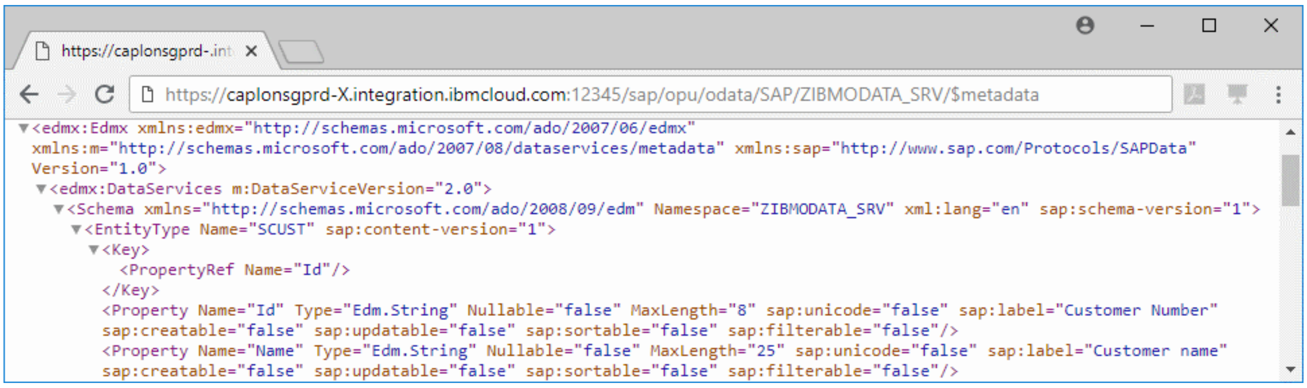
As an alternative, you can also use transaction `/n/IWFND/MAINT_SERVICE` to register the OData service in the SAP Gateway.

The OData service is available now. The service name is `<Project name>_SRV`, e.g. ZIBMODATA_SRV.

For a quick test, open your web browser or a REST API test tool like Postman and navigate to URL

```
https://<host>:<port>/sap/opu/odata/SAP/ZIBMODATA_SRV/$metadata
```

When asked for user and password, enter the credentials of your SAP user id.



https://caplonsgprd-X.integration.ibmcloud.com:12345/sap/opu/odata/SAP/ZIBMODATA_SRV/$metadata

```
▼<edmx:Edmx xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns:sap="http://www.sap.com/Protocols/SAPData"
  Version="1.0">
  ▼<edmx:DataServices m:DataServiceVersion="2.0">
    ▼<Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm" Namespace="ZIBMODATA_SRV" xml:lang="en" sap:schema-version="1">
      ▼<EntityType Name="SCUST" sap:content-version="1">
        ▼<Key>
           <PropertyRef Name="Id"/>
         </Key>
         <Property Name="Id" Type="Edm.String" Nullable="false" MaxLength="8" sap:unicode="false" sap:label="Customer Number"
         sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
         <Property Name="Name" Type="Edm.String" Nullable="false" MaxLength="25" sap:unicode="false" sap:label="Customer name"
         sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false"/>
```

Implementing OData Service Operations

The OData service has been deployed and is available. However, it does not provide any functionality apart from providing metadata information.

OData functions are implemented by the Model Provider Class and the Data Provider Class. Each class has a corresponding subclass that must be used for implementing OData service functionality. Never modify the base class.

Class Type	Base Class Name	Subclass Name
Model Provider Class	[Z]CL_<Project Name>_MPC	[Z]CL_<Project Name>_MPC_EXT
Data Provider Class	[Z]CL_<Project Name>_DPC	[Z]CL_<Project Name>_DPC_EXT

Table 3: OData Runtime Classes

In most cases, you must redefine only a small subset of methods to implement the relevant functions for each entity set of your OData service, see Table 4.

Function	Class	Method
Read Single Entity	*_DPC_EXT	<Entity Set Name>_GET_ENTITY
Query Set of Entities	*_DPC_EXT	<Entity Set Name>_GET_ENTITYSET
Create (Insert) a New Entity	*_DPC_EXT	<Entity Set Name>_CREATE_ENTITY
Update an Entity	*_DPC_EXT	<Entity Set Name>_UPDATE_ENTITY
Delete an Entity	*_DPC_EXT	<Entity Set Name>_DELETE_ENTITY

Table 4: OData Runtime Methods

To implement functions on entity set `SCUSTSet`, proceed as follows.

1. Call the SAP Gateway Service Builder (transaction SEGW), right-click on `ZIBMODATA` → *Runtimes Artifact* → `ZCL_ZIBMODATA_DPC_EXT` and select *Go to ABAP Workbench* from context menu.
2. In ABAP Workbench right-click on `ZCL_ZIBMODATA_DPC_EXT` → *Methods* → *Inherited Methods* → `SCUSTSET_GET_ENTITYSET` and select *Redefine* from context menu.
3. Insert code for method SCUSTSET_GET_ENTITYSET.

```
  method scustset_get_entityset.
```

```
    " DDIC table (data persistence)
    constants: c_table_name type tabname value 'SCUSTOM'.

    data: lt_field     type standard table of string,
          lr_data      type ref to data,
          ls_entityset like line of et_entityset.
    field-symbols: <ls_data> type any,
                   <lt_data> type any table.

    " create internal table dynamically
    create data lr_data type standard table of (c_table_name).
    assign lr_data->* to <lt_data>.

    " support FILTER
    data(lv_osql_where_clause)
           = io_tech_request_context->get_osql_where_clause( ).

    " support ORDERBY
    data(lt_orderby) = io_tech_request_context->get_orderby( ).
    data: lv_orderby type string value is initial,
          lv_order   type string.
    loop at lt_orderby into data(ls_orderby).
      if ls_orderby-order eq 'desc'.
        lv_order = 'DESCENDING'.
      else.
        clear lv_order.
      endif.
      concatenate lv_orderby ls_orderby-property lv_order
        into lv_orderby separated by space.
    endloop.

    " select data from DDIC table
    select * from (c_table_name) into table <lt_data>
     up to is_paging-top rows                    " support TOP
     where (lv_osql_where_clause)
     order by (lv_orderby).

    " transfer data to returning internal table
    loop at <lt_data> assigning <ls_data>.
      move-corresponding <ls_data> to ls_entityset.
      append ls_entityset to et_entityset.
    endloop.

  endmethod.
```

This code explicitly implements the operations FILTER, ORDERBY and TOP. SELECT projection and COUNT aggregation are implemented by the calling OData runtime. Nevertheless, performance optimizations could be implemented here. For example, you can implement optimized COUNT aggregation by adding code that - in case method `io_tech_request_context->has_count( )` returns `'X'` - writes the number of matching entities to exporting parameter `es_response_context-count` without actually reading the data records.
Note that the code above does not support pagination (operation SKIP).

4. Activate the code by clicking on the corresponding tool bar icon (or pressing CTRL+F3).
5. Redefine method SCUSTSET_GET_ENTITY and insert the code below for this method.

```
method scustset_get_entity.

  " DDIC table (data persistence)
  constants: c_table_name type tabname value 'SCUSTOM'.

  data: whereclause type string value ''.

  loop at it_key_tab into data(ls_keytab).
    if sy-tabix > 1.
      whereclause = whereclause && ` AND `.
    endif.
    whereclause = whereclause && ls_keytab-name &&
                  ` = '` && ls_keytab-value && `'`.
  endloop.

  select single * from (c_table_name) into er_entity
   where (whereclause).

endmethod.
```

6. Activate the code.
7. Redefine method SCUSTSET_CREATE_ENTITY and insert the code below for this method.

```
method scustset_create_entity.

  " DDIC table (data persistence)
  constants: c_table_name type tabname value 'SCUSTOM'.

  data: ls_input like er_entity,
        lr_data  type ref to data.
  field-symbols: <ls_tabline> type any.

  " create table line dynamically
  create data lr_data type (c_table_name).
  assign lr_data->* to <ls_tabline>.

  " read new entity from POST request
  io_data_provider->read_entry_data( importing es_data = ls_input ).

  " convert new entity to table line
  move-corresponding ls_input to <ls_tabline>.

  " insert data into DDIC table
  insert (c_table_name) from <ls_tabline>.
  call function 'DB_COMMIT'.

  " return new entity
  move ls_input to er_entity.

endmethod.
```

In this example the entity type basically matches the layout of the table that stores the data. Usually the mapping is much more complex. The input can be a deep structure type and the data persistence can be distributed over multiple DDIC tables. In this case the implementation of method `*_CREATE_ENTITY` is more complex.

8. Activate the code.

Now you can use OData service `ZIBMODATA_SRV` to query data that resides in table `SCUSTOM` and to insert a new record into that table.

Testing an OData service

The SAP Gateway Client is a tool that can be used to execute OData requests. It is integral part of the SAP Gateway.

The tool is started by calling transaction `/n/IWFND/GW_CLIENT`. You can specify the OData request URI, the HTTP method, the protocol (HTTP or HTTPS), the request headers and the request body. After you have entered all necessary request data, click button “Execute” to send the request. Afterwards the response headers and response data are displayed.

Here are some examples for testing OData service `ZIBMODATA_SRV`.

- Read METADATA
  Method: GET, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/$metadata`

- Query all records having POSTCODE = 12345 (FILTER)
  Method: GET, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet?$format=json&$filter=Postcode eq '12345'`

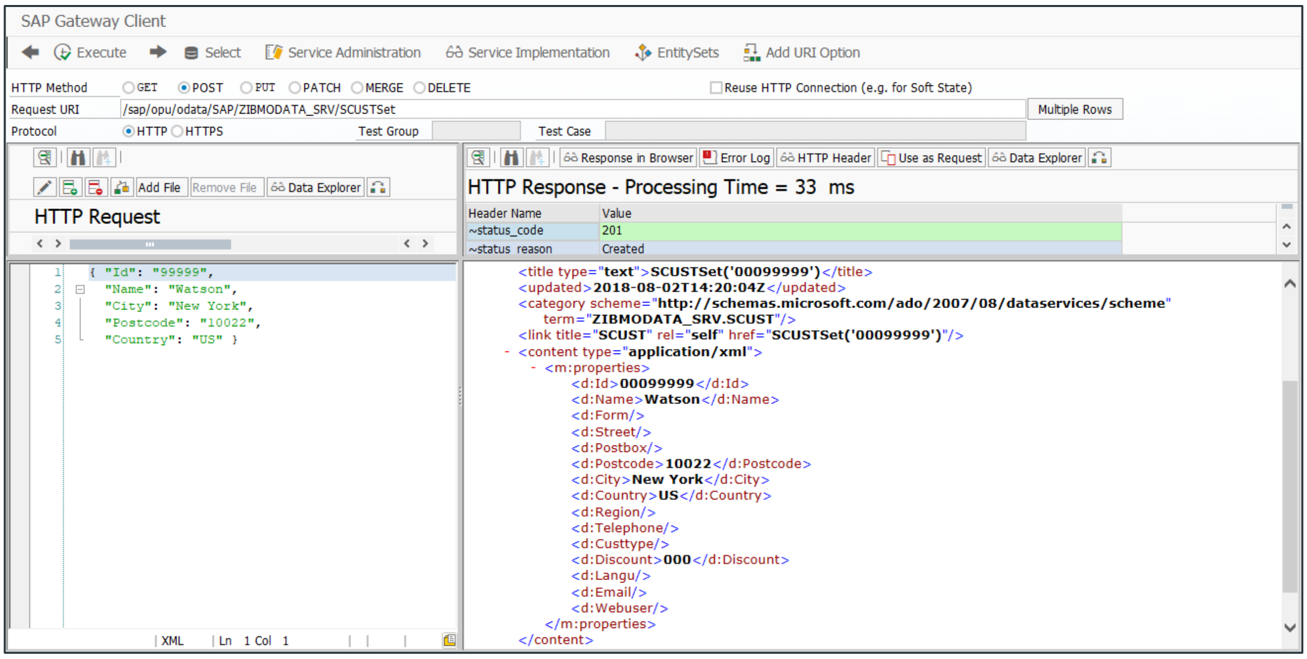
- Select only Name and City of customer "King" (SELECT)
  Method: GET, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet?$format=json&$filter=Name eq 'King' and Country eq 'GB'&$select=Name,City`

- Read Name and City of all British customers, ordered descending by City (ORDERBY)
  Method: GET, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet?$format=json&$filter=Country eq 'GB'&$select=Name,City&$orderby=City desc`

- Select three British customers with lowest ID (TOP)
  Method: GET, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet?$format=json&$filter=Country eq 'GB'&$select=Name,Id&$orderby=Id&$top=3`

- Get number of German customers (COUNT)
  Method: GET, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet/$count?$filter=Country eq 'DE'`

- Insert new customer
  Method: POST, Request URI:
  `/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet`
  Body:
  ```
  { "Id": "99999",
    "Name": "Watson",
    "City": "New York",
    "Postcode": "10022",
    "Country": "US" }
  ```



HTTP Method: GET [✓ POST] PUT PATCH MERGE DELETE
Reuse HTTP Connection (e.g. for Soft State): ☐
Request URI: /sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet
Protocol: [✓ HTTP] HTTPS
Test Group:
Test Case:

HTTP Request

```
{ "Id": "99999",
  "Name": "Watson",
  "City": "New York",
  "Postcode": "10022",
  "Country": "US" }
```

HTTP Response - Processing Time = 33 ms

Header Name	Value
~status_code	201
~status_reason	Created

```
    <title type="text">SCUSTSet('00099999')</title>
    <updated>2018-08-02T14:20:04Z</updated>
    <category scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme"
       term="ZIBMODATA_SRV.SCUST"/>
    <link title="SCUST" rel="self" href="SCUSTSet('00099999')"/>
  - <content type="application/xml">
    - <m:properties>
        <d:Id>00099999</d:Id>
        <d:Name>Watson</d:Name>
        <d:Form/>
        <d:Street/>
        <d:Postbox/>
        <d:Postcode>10022</d:Postcode>
        <d:City>New York</d:City>
        <d:Country>US</d:Country>
        <d:Region/>
        <d:Telephone/>
        <d:Custtype/>
        <d:Discount>000</d:Discount>
        <d:Langu/>
        <d:Email/>
        <d:Webuser/>
      </m:properties>
    </content>
```

Consuming an OData service

Since OData bases on the REST protocol, OData services can easily be consumed by an application using REST and HTTP, respectively. Most programming languages provide libraries or packages that implement a REST client interface. There are also client libraries for OData available for java, javascript, python and others.

Find sample code below for consuming SAP OData services by applications running on the IBM Cloud.

Node.js

This section shows how an SAP OData service can be consumed by a node.js application running on IBM Cloud.

Below it is assumed that Cloud Foundry sample application "node.js starter application" has been created and deployed on IBM Cloud. This sample application is generated and started by default when an "SDK for node.js" instance is created on IBM Cloud.

You should also enable a toolchain for that sample application including a connection to Git and clone the Git repository onto your local computer to develop and test your application locally. When done, commit your updates to Git. Then the Delivery Pipeline that is part of the toolchain re-builds the Cloud Foundry application and re-deploys it on the IBM Cloud automatically.

Start with the "node.js starter application" code to implement an application that provides a web interface for creating a new entity in the target SAP system.

Replace content of file `public/index.html` by the code below. It has a html form for prompting the user for the new entity's data.

```
<!DOCTYPE html>
<html>
	<head>
		<title>OData Demo</title>
		<meta charset="utf-8">
		<meta http-equiv="X-UA-Compatible" content="IE=edge">
		<meta name="viewport" content="width=device-width, initial-scale=1">
		<link rel="stylesheet" href="stylesheets/style.css">
	</head>

	<body>
		<form style="text-align:center" action="/new_entity" method="post">
			<table style="width:350px;padding:30px">
				<tr><td style="width:150px"><label for="id">Id:</label></td>
				<td style="width:150px"><input type="text" id="id" name="Id"
							placeholder="Enter Id" /></td></tr>
				<tr><td><label for="name">Name:</label></td>
				<td><input type="text" id="name" name="Name"
						placeholder="Enter name" /></td></tr>
				<tr><td><label for="city">City:</label></td>
				<td><input type="text" id="city" name="City"
						placeholder="Enter city" /></td></tr>
				<tr><td><label for="postcode">Postcode:</label></td>
				<td><input type="text" id="postcode" name="Postcode"
						placeholder="Enter postcode" /></td></tr>
				<tr><td><label for="country">Country:</label></td>
```

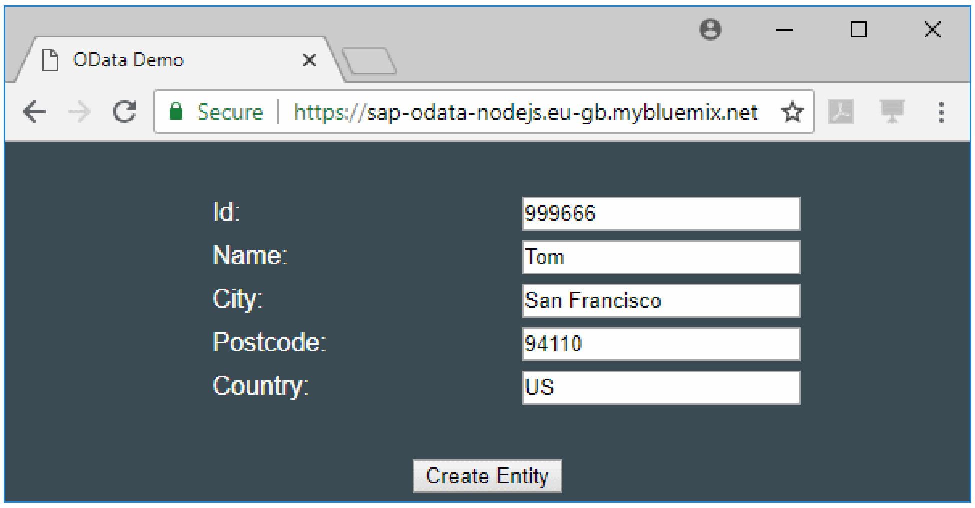
```
                                <td><input type="text" id="country" name="Country"
                                           placeholder="Enter country id" /></td></tr>
                        </table>
                        <input type="submit" value="Create Entity" />
                </form>
        </body>
</html>
```

Submitting the html form (by clicking button "Create Entity") posts the data to api endpoint `/new_entity`. To process the user's input, you must implement that api endpoint in the backend web application. To do so, replace content of file `app.js` by the code below.

```
/*** Adjust credentials here ********************************************/
// hard-coded SAP user; must be adjusted according to authorization policy
const auth = 'Basic ' + new Buffer('watson:********').toString('base64');

// connection info
const hostname = "caplonsgprd-3.integration.ibmcloud.com";
const port = 12345;
const odata_service_path = "/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet";
/***********************************************************************/

 // This application uses express as its web server
// for more info, see: http://expressjs.com
var express = require('express');

// cfenv provides access to your Cloud Foundry environment
// for more info, see: https://www.npmjs.com/package/cfenv
var cfenv = require('cfenv');

// create a new express server
var app = express();

// serve the files out of ./public as our main files
app.use(express.static(__dirname + '/public'));

//use the https protocol for secured communication
var https = require("https");

// body-parser exposes the request body as property req.body
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));

// get the app environment from Cloud Foundry
var appEnv = cfenv.getAppEnv();

// start server on the specified port and binding host
app.listen(appEnv.port, '0.0.0.0', function() {
        // print a message when the server starts listening
        console.log("server starting on " + appEnv.url);
});


// Implementation of api "/new_entity" uses OData service GET to
//   check entity existence and to fecth session id and CSRF token.
//   Writes (new) entity afterwards.
app.post('/new_entity', function(req_web, resp_web) {

        // body of calling request contains new entity
        var new_entity = JSON.stringify(req_web.body);
        if (typeof new_entity === "undefined") {
                resp_web.send('New entity not defined!');
                return;
        }

        var entity_id = req_web.body.Id;

        // Check entity existence and fetch session ID and CSRF token
        var options = {
                hostname: hostname,
                port: port,
                path: odata_service_path + "('" + entity_id + "')?$format=json",
                method: 'GET',
                headers: {
                        'Content-Type': 'application/json',
                        'accept': 'application/json',
                        'Authorization': auth,     // if ODtata service requires logon
                        'x-csrf-token': 'fetch'    // request CSRF token
                }
        };

        var response = "";
        var req_sap = https.request(options, function(res_sap) {
                res_sap.setEncoding('utf8');
                res_sap.on('data', function (data) {
                        // called multiple times -> combine chunks to a single response
                        response += data;
                });
                res_sap.on('end', function() {

                        if ((res_sap.statusCode !== 200) && (res_sap.statusCode !== 404)) {
                                // response contains html error page
                                resp_web.send(response);
                        } else {

                                // check existence
                                var result = JSON.parse(response);
                                if (typeof result.d  !== 'undefined') {
                                        resp_web.send('Entity ' + result.d.Id + ' already exists.');
                                } else {

                                        // save cookie / CSRF token and write entity
                                        var cookie = res_sap.headers['set-cookie'];
                                        var csrf_token = res_sap.headers['x-csrf-token'];
                                        console.log('CSRF Token:' + csrf_token);
                                        write_entity(req_web, resp_web, cookie, csrf_token);

                                }
                        }
                });
        });

        req_sap.on('error', function(e) {
                console.log('problem with request: ' + e.message);
                return;
        });

        req_sap.end();
});


// Function write_entity
//   Called by API /new_entity; creates a new entity in the target system
function write_entity(req_web, resp_web, cookie, csrf_token) {

        // OData request options including headers
```

```
	var req_odata_options = {
		hostname: hostname,
		port: port,
		path: odata_service_path,
		method: 'POST',
		headers: {
			'Content-Type': 'application/json',
			'accept': 'application/json',
			//'X-Requested-With': 'X',      // if CSRF is disabled
			'Authorization': auth,          // if ODtata service requires logon
			'cookie': cookie,               // session id
			'x-csrf-token': csrf_token      // CSRF token
		}
	};

	// Build OData request
	var req_odata = https.request(req_odata_options, function(resp_odata) {
		var body_odata = "";
		resp_odata.setEncoding('utf8');

		resp_odata.on('data', function (data) {
			// called multiple times -> concatenate chunks to a single string
			body_odata += data;
		});

		resp_odata.on('end', function() {
			// OData service response complete
			// -> should look like { "d": { "__metadata": {...}, "Id": .., .. } }
			var result = JSON.parse(body_odata);

			// return new entity or error message to calling request
			if (typeof result.d !== 'undefined') {
				var entity = result.d;
			    delete entity.__metadata;
				resp_web.send( 'Entity has been created: ' +
							JSON.stringify(entity) );
			} else {
				resp_web.send('Error: ' + JSON.stringify(body_odata.error) );
			}

		});
	});

	req_odata.on('error', function(e) {
		resp_web.send('Call to OData service failed!');
	});


	// body of request contains new entity data
	// -> transfer it to body of odata request
	req_odata.write(JSON.stringify(req_web.body));

	// finalize request
	req_odata.end();
}
```

Submit your changes. After your application has been re-deployed, type in your application URL in a web browser, enter new entity's data and click *Create Entity*.



Id:	999666
Name:	Tom
City:	San Francisco
Postcode:	94110
Country:	US

In response, your application returns the new entity's data. In the SAP system, check table SCUSTOM for the new entity.

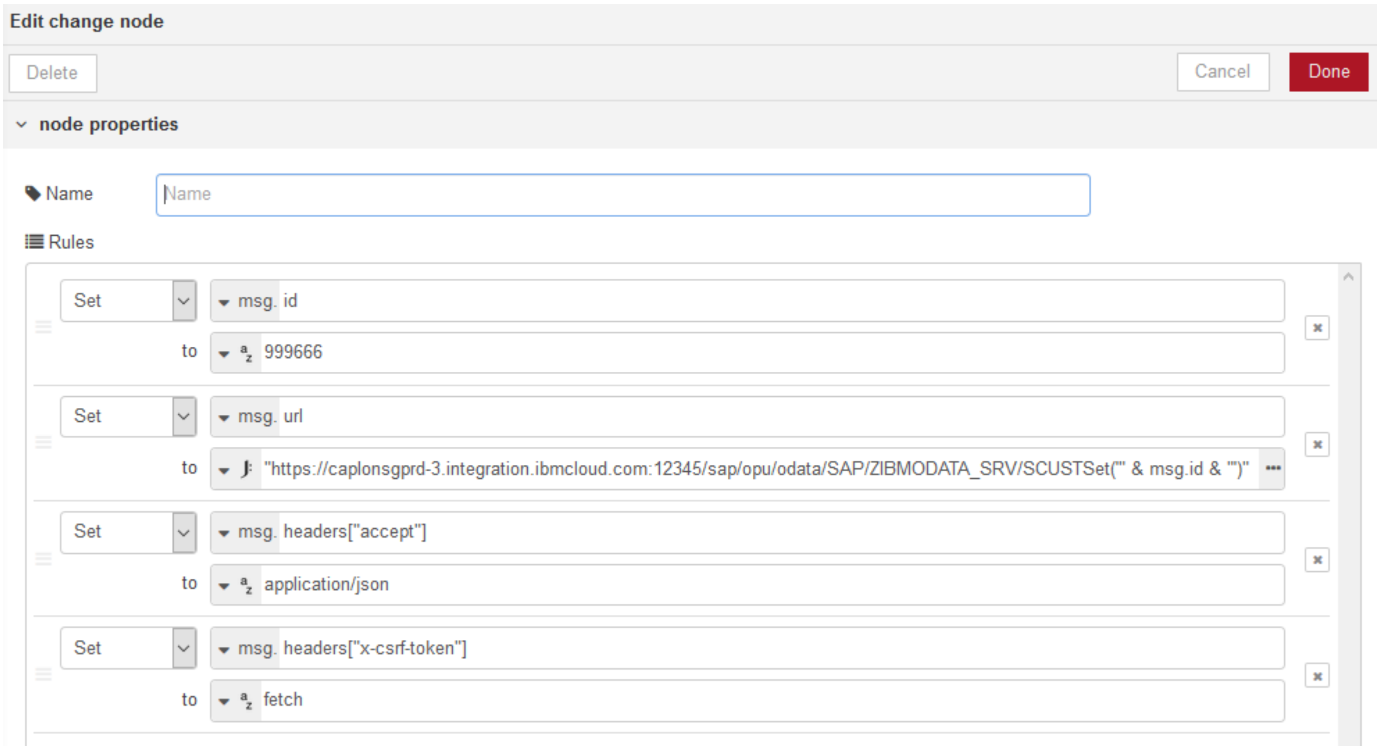
Node-RED

With Node-RED a "http request" node is used to call an SAP OData service. The Node requires the following parameters as attributes of the `msg` object.

Attribute	Value	Remark
`msg.url`	OData service url	
`msg.headers["Content-Type"]`	`application/json`	Not for GET requests
`msg.headers["accept"]`	`application/json`	
`msg.headers["cookie"]`	<cookie>	Session cookie
`msg.headers["x-csrf-token"]`	`fetch` (GET) <token> (POST)	CSRF token
`msg.headers["X-Requested-With"]`	X	If CSRF tokens are disabled
`msg.payload`	Entity data in json format	Not for GET requests

Table 5: Node-RED parameters for OData request

You can set the http request parameters within a "change" node that is passed before the "http request" node gets executed.



Name: Name

Rules

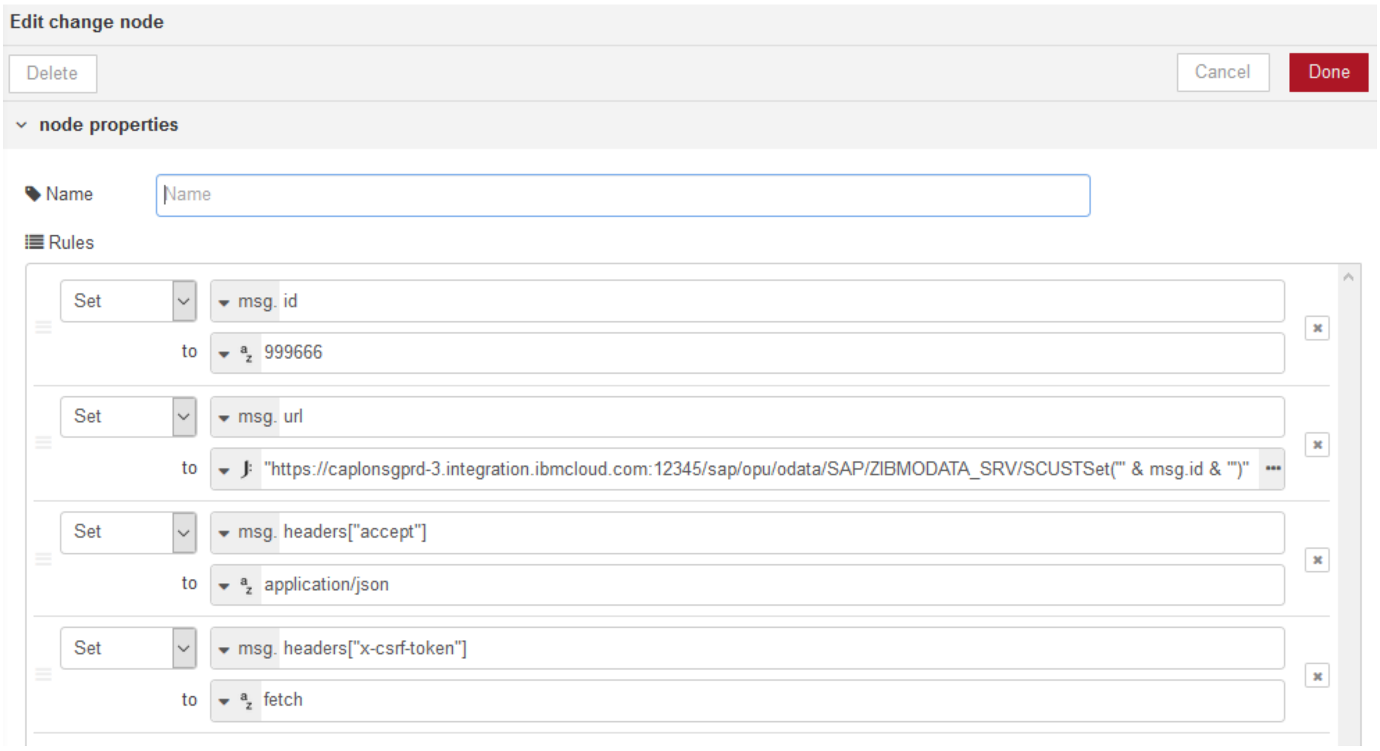
Set msg. id to 999666

Set msg. url to "https://caplonsgprd-3.integration.ibmcloud.com:12345/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet('" & msg.id & "')"

Set msg. headers["accept"] to application/json

Set msg. headers["x-csrf-token"] to fetch

Figure 6 shows an example node configuration for a GET request that reads an entity.


Edit change node

Delete | Cancel | Done

node properties

Name: Name

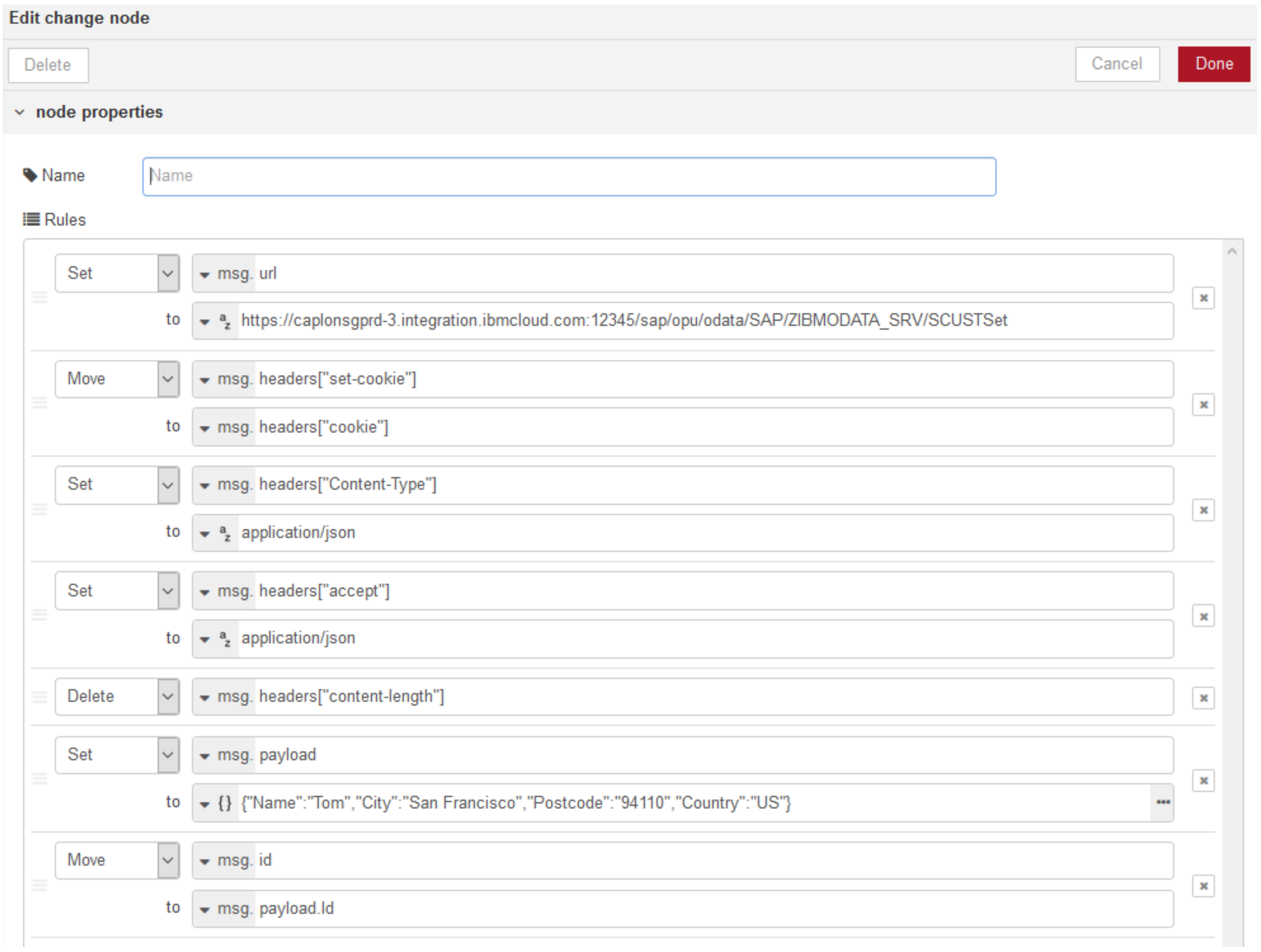
Rules

- Set msg. id to 999666
- Set msg. url to "https://caplonsgprd-3.integration.ibmcloud.com:12345/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet('" & msg.id & "')"
- Set msg. headers["accept"] to application/json
- Set msg. headers["x-csrf-token"] to fetch

Figure 6: Configuration of a "change" node to prepare reading an OData entity

When configuring the "http request" node, select *Method* GET and check *Use basic authentication*. Also provide the SAP *Username* and *Password*, unless you have saved the logon data within the configuration of SAP OData service.

Adjust this procedure to configure an OData request to create an entity. Thus, add a "change" node to set the required attributes of the msg object. We assume that a OData GET request has been executed before and therefore the `msg` objects holds the correct values for the `set-cookie` and `x-csrf-token` headers. See Figure 7 for required settings.



Edit change node

Delete | Cancel | Done

node properties

Name: Name

Rules

Set	msg. url	to https://caplonsgprd-3.integration.ibmcloud.com:12345/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet
Move	msg. headers["set-cookie"]	to msg. headers["cookie"]
Set	msg. headers["Content-Type"]	to application/json
Set	msg. headers["accept"]	to application/json
Delete	msg. headers["content-length"]	
Set	msg. payload	to {"Name":"Tom","City":"San Francisco","Postcode":"94110","Country":"US"}
Move	msg. id	to msg. payload.Id

Figure 7: Configuration of a "change" node to prepare creation of an OData entity

At next, configure a "http request" node, select *Method* POST and check *Use basic authentication*. Also provide the SAP *Username* and *Password*, if logon is required for the SAP OData service.

For test purposes you might want to add a "http" node before the sequence of nodes that you have created, for example a http api end-point `/new_entity` for GET requests. Also add a "http response" node at the end. You end up with a flow like shown in Figure 8.

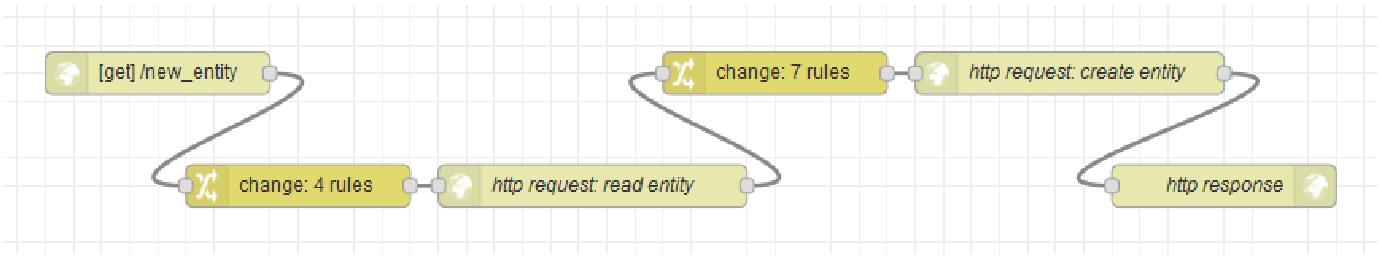
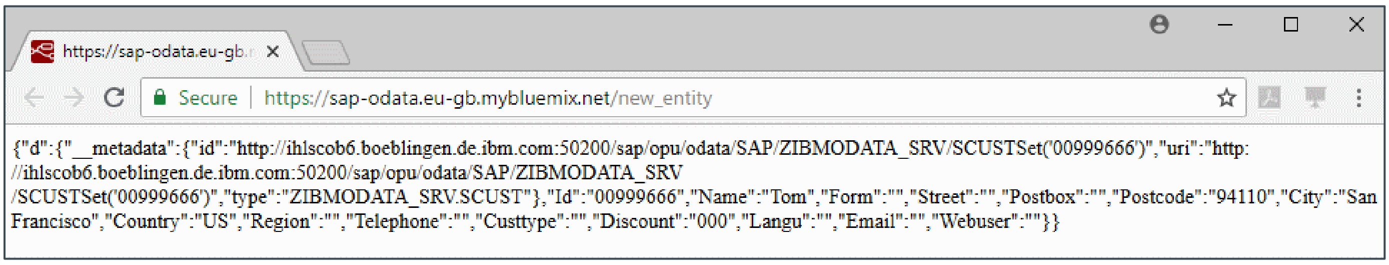


Figure 8: Node-RED flow

After implementing the example above, you can trigger the creation of a new entity by entering address <your application URL>`/new_entity` in your web browser. The response should look like shown in Figure 9.


https://sap-odata.eu-gb.r
Secure
https://sap-odata.eu-gb.mybluemix.net/new_entity

```
{"d":{"__metadata":{"id":"http://ihlscob6.boeblingen.de.ibm.com:50200/sap/opu/odata/SAP/ZIBMODATA_SRV/SCUSTSet('00999666')","uri":"http:
//ihlscob6.boeblingen.de.ibm.com:50200/sap/opu/odata/SAP/ZIBMODATA_SRV
/SCUSTSet('00999666')","type":"ZIBMODATA_SRV.SCUST"},"Id":"00999666","Name":"Tom","Form":"","Street":"","Postbox":"","Postcode":"94110","City":"San
Francisco","Country":"US","Region":"","Telephone":"","Custtype":"","Discount":"000","Langu":"","Email":"","Webuser":""}}
```

Figure 9: Call Node-RED application from web browser

The OData entity has been created in SAP table `SCUSTOM`.

Java

For accessing OData services from a java program you might want to use the Apache Olingo, which is an OData client library (http://olingo.apache.org/).

The following java code implements an application that reads and creates an entity using SAP OData service. It uses the Olingo client library which must be available in the class path. After execution, the new entity is written to the console and you should find that entity in SAP table SCUSTOM.

```java
import java.net.URL;
import java.net.URI;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;

import java.util.Map;
import java.util.Set;
import java.util.Map.Entry;
import java.util.HashMap;
import java.util.Base64;

import org.apache.olingo.odata2.api.edm.Edm;
import org.apache.olingo.odata2.api.edm.EdmEntityContainer;
import org.apache.olingo.odata2.api.edm.EdmEntitySet;
import org.apache.olingo.odata2.api.ep.EntityProvider;
import org.apache.olingo.odata2.api.ep.EntityProviderReadProperties;
import org.apache.olingo.odata2.api.ep.EntityProviderWriteProperties;
import org.apache.olingo.odata2.api.ep.entry.ODataEntry;
import org.apache.olingo.odata2.api.processor.ODataResponse;
import org.apache.olingo.odata2.api.commons.HttpStatusCodes;


public class OlingoClient {

      /*** Adjust credentials here **********************************************************/
      private static final String SERVICE_URL =
       "https://caplonsgprd-3.integration.ibmcloud.com:12345/sap/opu/odata/SAP/ZIBMODATA_SRV";
      private static final String USERNAME = "watson";
      private static final String PASSWORD = "********";
      /*************************************************************************************/

      public static final String HTTP_METHOD_POST = "POST";
      public static final String HTTP_METHOD_PUT  = "PUT";
      public static final String HTTP_METHOD_GET  = "GET";

      public static final String HTTP_HEADER_CONTENT_TYPE = "Content-Type";
      public static final String HTTP_HEADER_ACCEPT       = "Accept";
      public static final String APPLICATION_JSON = "application/json";
      public static final String APPLICATION_XML  = "application/xml";

      public static final String METADATA = "$metadata";
      public static final String SEPARATOR = "/";


      /* Function main
       * creates an entity via SAP OData service ZIBMODATA_SRV
       */
      public static void main(String[] paras) throws Exception {
```

```
// SAP OData service and entityset
String serviceUrl = SERVICE_URL;
String usedFormat = APPLICATION_JSON;
String entitySetName = "SCUSTSet";
String entityId = "999222";

HttpURLConnection connection;
HttpStatusCodes httpStatusCode;
ODataEntry entity = null;


/*** get EDM (Entity Data Model) ***/
connection = initializeConnection(serviceUrl + SEPARATOR + METADATA,
     APPLICATION_XML, HTTP_METHOD_GET);
connection.connect();
httpStatusCode =
     HttpStatusCodes.fromStatusCode(connection.getResponseCode());
if (400 <= httpStatusCode.getStatusCode() &&
    httpStatusCode.getStatusCode() <= 599) {
      throw new RuntimeException("Http Connection failed with status " +
        httpStatusCode.getStatusCode() + " " + httpStatusCode.toString());
}
InputStream metadata_content = connection.getInputStream();
Edm edm = EntityProvider.readMetadata(metadata_content, false);

// get EDM properties
EdmEntityContainer entityContainer = edm.getDefaultEntityContainer();
EdmEntitySet entitySet = entityContainer.getEntitySet(entitySetName);


/*** Read entity ***/
connection = initializeConnection(serviceUrl + SEPARATOR + entitySetName +
     "('" + entityId + "')", APPLICATION_JSON, HTTP_METHOD_GET);


connection.setRequestProperty("x-csrf-token", "fetch");

connection.connect();
httpStatusCode =
     HttpStatusCodes.fromStatusCode(connection.getResponseCode());
int statusCode = httpStatusCode.getStatusCode();
if (statusCode >= 400 && statusCode <= 599 && statusCode != 404) {
      throw new RuntimeException("Http Connection failed with status " +
            statusCode + " " + httpStatusCode.toString());
}

// save cookie and CSRF token
String cookie = connection.getHeaderField("set-cookie");
String x_csrf_token = connection.getHeaderField("x-csrf-token");

// check entity existence
if (statusCode == 404) {
      System.out.println("Entity with Id " +  entityId +
            " does not yet exist.\n");
} else {
      System.out.println("Entity found:");
      InputStream read_content = connection.getInputStream();
      entity = EntityProvider.readEntry(APPLICATION_JSON, entitySet,
            read_content, EntityProviderReadProperties.init().build());

      printEntity(entity);
}
```

```
/*** OData CREATE request ***/

// set entity data
java.util.Map<String, Object> data = new HashMap<String, Object>();
data.put("Id",       entityId);
data.put("Name",     "Tom");
data.put("City",     "San Francisco");
data.put("Postcode", "94110");
data.put("Country",  "US");

// set all properties that are not nullable
data.put("Form",     "");
data.put("Street",   "");
data.put("Postbox",  "");
data.put("Region",   "");
data.put("Telephone","");
data.put("Custtype", "");
data.put("Discount", "");
data.put("Langu",    "");
data.put("Email",    "");
data.put("Webuser",  "");

// initialize HTTP connection
connection = initializeConnection(serviceUrl + SEPARATOR + entitySetName,
     usedFormat, HTTP_METHOD_POST);

// set header for CSRF token
connection.setRequestProperty("cookie", cookie);
connection.setRequestProperty("x-csrf-token", x_csrf_token);

// get properties that are necessary to write entity
URI rootUri = new URI(entitySetName);
EntityProviderWriteProperties properties =
     EntityProviderWriteProperties.serviceRoot(rootUri).build();

// serialize data into ODataResponse object
ODataResponse response = EntityProvider.writeEntry(usedFormat, entitySet,
     data, properties);

// write data (chunk-wise) to (http) entity (which is for default Olingo
//   implementation an InputStream)
Object entity_stream = response.getEntity();
if (entity_stream instanceof InputStream) {
       ByteArrayOutputStream buffer = new ByteArrayOutputStream();
       int readCount;
       byte[] chunk = new byte[8192];
       while ((readCount = ((InputStream)entity_stream).read(
             chunk, 0, chunk.length)) != -1) {
              buffer.write(chunk, 0, readCount);
       }
       buffer.flush();
       ((InputStream)entity_stream).close();
       connection.getOutputStream().write(buffer.toByteArray());
}

// if a entity is created (via POST request) the response body contains
//   the new created entity
entity = null;
httpStatusCode =
     HttpStatusCodes.fromStatusCode(connection.getResponseCode());
if(httpStatusCode == HttpStatusCodes.CREATED) {

       // get the content as InputStream and de-serialize it into an
```

```
                //   ODataEntry object
                InputStream content = connection.getInputStream();
                entity = EntityProvider.readEntry(usedFormat, entitySet, content,
                     EntityProviderReadProperties.init().build());
                System.out.println("Entity has been created successfully: ");

                printEntity(entity);

          } else {
                throw new RuntimeException("Http Connection failed with status " +
                     statusCode + " " + httpStatusCode.toString());
          }

          connection.disconnect();
    }


    /* Function HttpURLConnection
     * Establishes a connection for REST calls.
     */
    private static HttpURLConnection initializeConnection(String serviceUrl,
         String contentType, String httpMethod)
         throws MalformedURLException, IOException {

          URL url = new URL(serviceUrl);
          HttpURLConnection connection = (HttpURLConnection) url.openConnection();

          connection.setRequestMethod(httpMethod);

          // set "Authorization" header to base64-encoded SAP logon credentials
          String userpassword = USERNAME + ":" + PASSWORD;
          Base64.Encoder enc = Base64.getEncoder();
          String encodedAuthorization = enc.encodeToString(
               userpassword.getBytes("UTF-8") );
          connection.setRequestProperty("Authorization",
               "Basic "+ encodedAuthorization);

          // set "Accept" and "Content-Type" headers
          connection.setRequestProperty(HTTP_HEADER_ACCEPT, contentType);
          if( HTTP_METHOD_POST.equals(httpMethod) ||
              HTTP_METHOD_PUT.equals(httpMethod) ) {
                connection.setDoOutput(true);
                connection.setRequestProperty(HTTP_HEADER_CONTENT_TYPE, contentType);
          }

          // disabled CSRF token
          // connection.setRequestProperty("X-Requested-With", "X");

          return connection;
    }


    /* Function printEntity
     * Prints an entity to the console.
     */
    private static void printEntity(ODataEntry entity) {
          StringBuilder b = new StringBuilder();
          Map<String, Object> props = entity.getProperties();
          Set<Entry<String, Object>> entries = props.entrySet();
          for (Entry<String, Object> en : entries) {
                b.append(en.getKey()).append(": ");
                Object value = en.getValue();
                b.append(value).append("\n");
```

```
            }
            System.out.println(b.toString());
      }

}
```

Appendix

Transaction Codes

See Table 6 for useful transaction codes.

Application	Transaction Code
Web Service Catalog	SICF
ICM Monitor	SMICM
SAP Gateway Service Builder	SEGW
SAP Gateway Service Catalog	/n/IWFND/MAINT_SERVICE
SAP Gateway Client	/n/IWFND/GW_CLIENT
SAP Gateway Error Log	/n/IWFND/ERROR_LOG

Table 6: Transaction codes

References

[1] OData Homepage
https://www.odata.org

[2] SAP Community: OData – Everything that you need to know (Part 1 – 10)
https://blogs.sap.com/2016/02/08/odata-everything-that-you-need-to-know-part-1/

[3] SAP Gateway Community
https://www.sap.com/community/topic/gateway.html

[4] SAP Community Wiki: SAP Annotations for OData
https://wiki.scn.sap.com/wiki/display/EmTech/SAP+Annotations+for+OData+Version+2.0