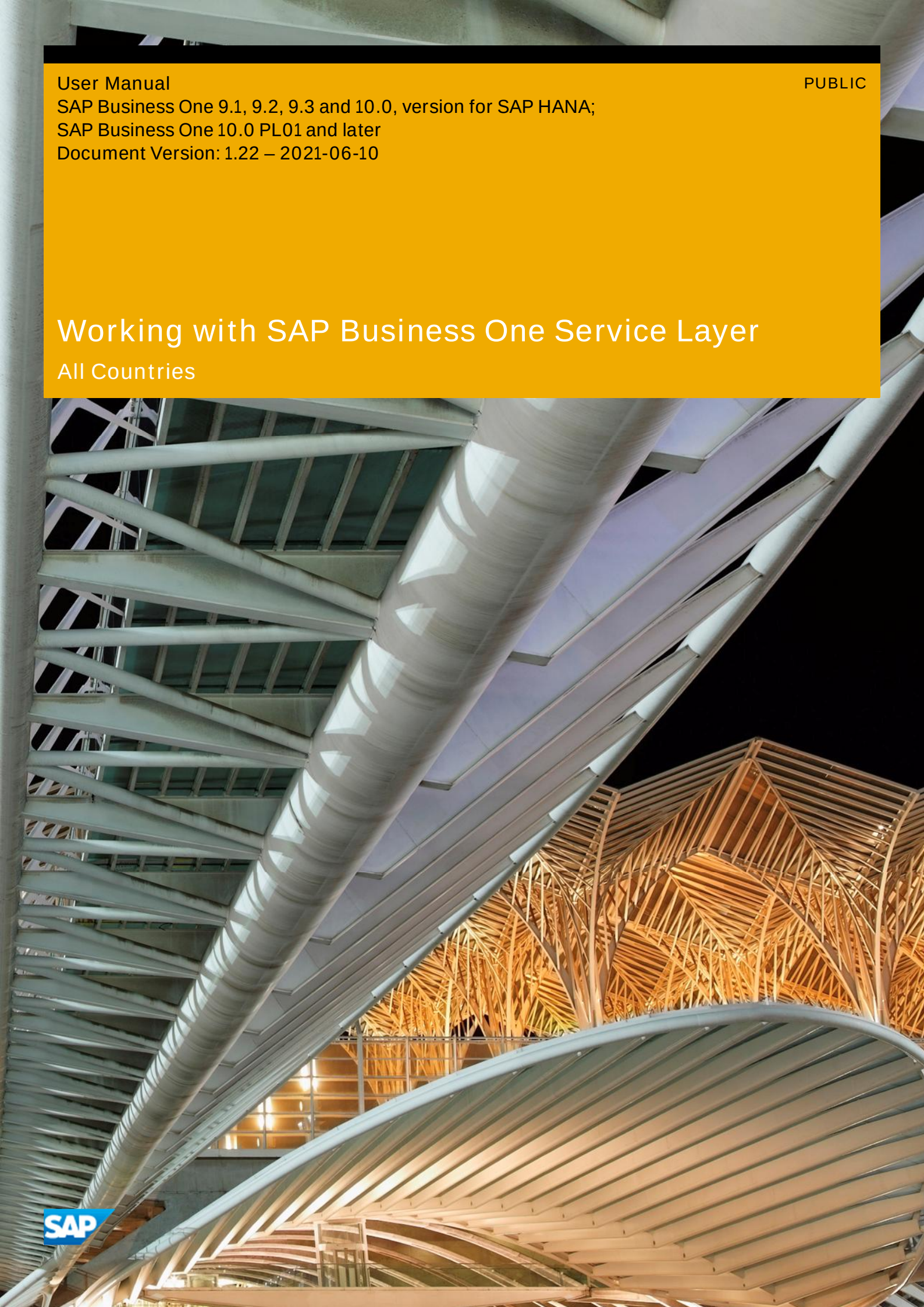
User Manual

PUBLIC

SAP Business One 9.1, 9.2, 9.3 and 10.0, version for SAP HANA;
SAP Business One 10.0 PL01 and later
Document Version: 1.22 – 2021-06-10

Working with SAP Business One Service Layer

All Countries



Typographic Conventions

Type Style	Description
Example	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Textual cross-references to other documents.
Example	Emphasized words or expressions.
EXAMPLE	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
EXAMPLE	Keys on the keyboard, for example, F2 or ENTER.

Document History

Version	Date	Change
1.0	2014-06-27	The first release of SAP Business One Service Layer
1.1	2014-11-11	· SAP Business One service user · Configuration by request · Configuration options (Schema, SessionTimeout) · User-defined schemas · User-defined objects · User-defined fields (metadata management) · Retrieving individual properties · Associations and navigation properties (Experimental)
1.2	2014-12-30	· Support OData version 4 · Metadata for UDF/UDT/UDO · Updates for user-defined objects, user-defined fields and user-defined tables
1.3	2015-03-19	· Use SLD server during logon · Support $inlinecount in OData query · Add the code samples for Service Layer versus DI API · Add the limitations of Service Layer
1.4	2015-07-20	· Add Session in Login/Logout · Add an example "Preview an Order" in Actions · Add Cross Origin Resource Sharing (CORS)
1.5	2015-12-15	Add Single Sign-On
1.6	2016-03-04	Minor update for OBServer.
1.7	2016-05-04	· Add Create Entity with No Content · Add Aggregation · Add Attachments · Add Item Image · Add Cross Origin Resource Sharing
1.8	2016-07-28	· Support JavaScript Extension · Support case-insensitive query · Support query with grouping · Support Cross-Joins · Support allowed request header in CORS

Version	Date	Change
1.9	2017-08-14	Support SAP Business One 9.3, version for SAP HANA
1.10	2018-01-15	· Support Semantic Layer Service · Minor corrections
1.11	2018-02-09	· Support cancel/close entity for UDO · Support row level filter
1.12	2018-12-04	Update Appendix II: Collection object naming difference
1.13	2019-03-13	Add back the example "Preview an Order" in Actions
1.14	2019-05-20	Add the Ping Pong API
1.15	2019-10-20	Support SAP Business One 10.0, version for SAP HANA. Add Configuration Controller for Service Layer.
1.16	2019-12-02	Add Employee Image
1.17	2020-01-02	Support SAP Business One 10.0 PL01 (on Microsoft SQL Server)
1.18	2020-04-08	Support SQL View. Configuration Controller Is Available on the SQL version.
1.19	2020-08-20	Update Appendix II: Collection object naming difference
1.20	2020-10-28	Support SQL Query. Update JavaScript SDK Generator Tool for the Micorsoft Windows enviroment.
1.21	2021-03-22	SQLQueries supports user-defined objects and user-defined tables. Support ETag mechanism.
1.22	2021-06-10	For oData Query, additional selection options are supported: · Single entity $expand with $select list · Collection entity $expand with $select list

Table of Contents

1 Introduction

1.1 About This Document

This document covers the basic usages of SAP Business One Service Layer and explains the technical details of building a stable, scalable Web service using SAP Business One Service Layer.

1.2 Target Audience

We recommend that you refer to this document if you are:

- Developing applications based on Service Layer API
- Planning your first load balancing deployment
- Improving your system's performance
- Assuring your system's stability under heavy work load

This document is intended for system administrators who are responsible for configuring, managing, and maintaining an SAP Business One Service Layer installation. Familiarity with your operating system and your network environment is beneficial, as is a general understanding of web application server management.

This document is also relevant for software developers who build add-ons for SAP Business One.

1.3 About SAP Business One Service Layer

SAP Business One Service Layer is a new generation of extension API for consuming SAP Business One data and services. It builds on core protocols such as HTTP and OData, and provides a uniform way to expose full-featured business objects on top of a highly scalable and high-availability Web server. Currently, Service Layer supports OData version 3, version 4, and a few selected OData client libraries, for example, WCF for .Net developers; data.js for JavaScript developers.

> **Note**
>
> You can use the following URIs to switch the OData versions:
>
> - `/b1s/v1/$metadata` is for odata v3
> - `/b1s/v2/$metadata` is for odata v4

2 Getting Started

2.1 System Requirements

SAP Business One Service Layer runs on both SUSE Linux Enterprise and Microsoft Windows operating systems (as of SAP Business One 10.0 PL01). It is an application server built on the Apache HTTP Web server.

SAP Business One Service Layer can be deployed in one of two different modes:

- An integrated mode, installing on the same database server so as to keep the system landscape as simple as possible
- A distributed mode, installing on separate machines to obtain more computing power for higher concurrent throughput

For hardware requirements, such as memory capacity or number of CPU cores, refer to *SAP HANA hardware specifications* and *Hardware Requirements Guide for SAP Business One*.

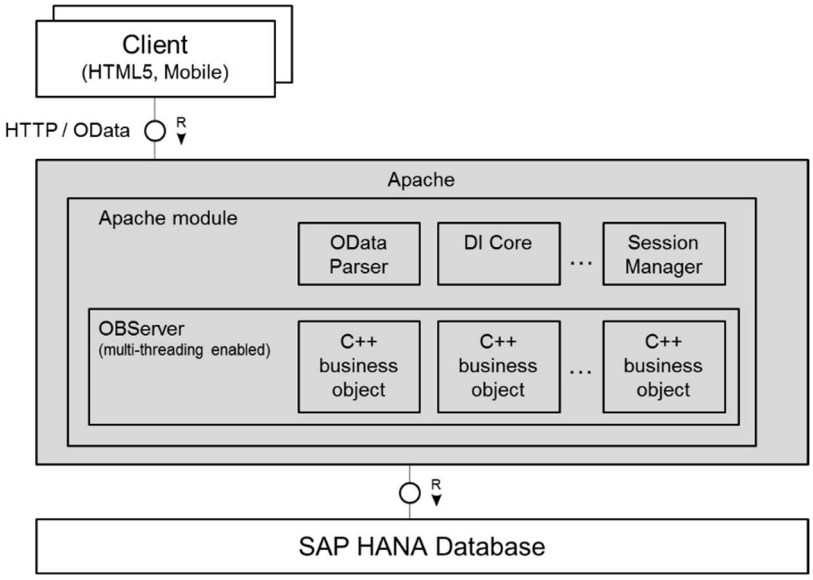
For more information, refer to the Platform Support Matrix in conjunction with the SAP Productivity Availability Matrix and the Administrator's Guides for SAP Business One (versions for both Microsoft SQL and SAP HANA) on SAP Help Portal.

2.2 Architecture Overview

SAP Business One Service Layer has a 3-tier architecture: the clients communicate with the Web server using HTTP/OData, and the Web server relies on the database for data persistence.

Within the Web server, several key components are involved in handling incoming OData-based HTTP requests:

- The OData Parser looks at the requested URL and HTTP methods (`GET`/`POST`/`PATCH`/`DELETE`), translates them into the business objects to be operated on, and calls each object's respective method for create/retrieve/update/delete (CRUD) operations. In reverse, the OData Parser also receives the returned data from business objects, translates them into HTTP return code and JSON data representatives, and responds to the original client.
- The DI Core is the interface for accessing SAP Business One objects and services, the same one that is used by SAP Business One DI API. As a result, Service Layer API and DI API have identical definitions for objects and object properties, smoothing the learning curve for developers who have already acquired DI API development experience.
- The session manager implements session stickiness, working with the Service Layer load balancer, so that requests from the same client will be handled by the same Service Layer node.
- OBServer is the body of business logic dealing with the actual work, for example, tax calculation, posting, and so on. Service Layer achieves high performance and scalability by leveraging multi-processing.

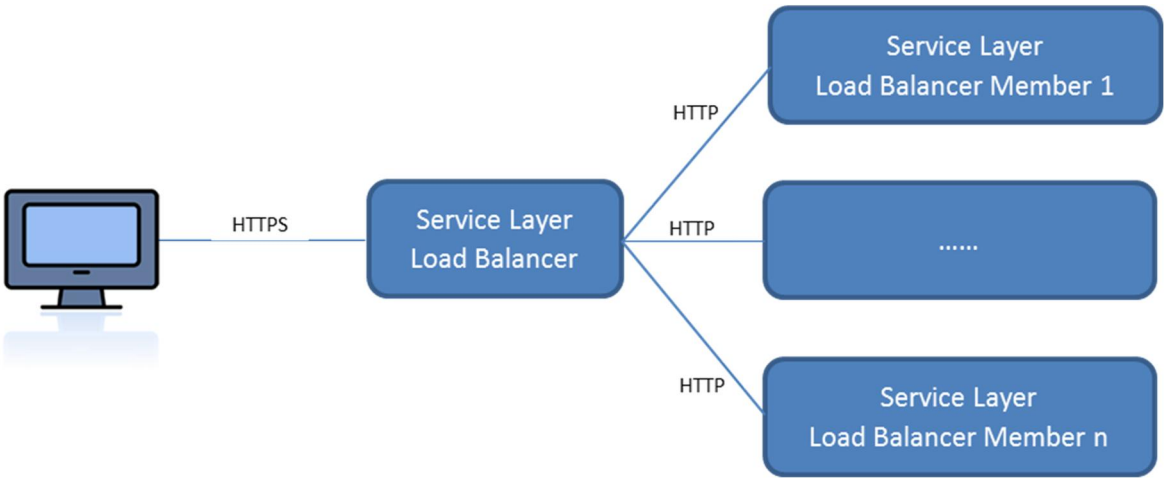


In order to achieve even higher availability and scalability, we recommend deploying multiple Service Layer instances with a load balancer in the front. The benefits include the following:

- Client requests can be dispatched to different Service Layer instances and executed in parallel.
- If Service Layer is installed in a distributed mode, and there is a hardware failure in one host machine, Service Layer is smart enough to re-dispatch client requests to another live instance without asking users to log on again.

2.3 Installing SAP Business One Service Layer

The Service Layer is an application server that provides Web access to SAP Business One services and objects and uses the Apache HTTP Server (or simply Apache) as the load balancer, which works as a transit point for requests between the client and various load balancer members. The architecture of the Service Layer is illustrated below:



Recommendation

As the communication between the load balancer and the load balancer members is transmitted via HTTP instead of HTTPS, you should configure the firewall on each load balancer member machine in such a way that only visits from the load balancer are allowed to the load balancer members.

You may set up Service Layer in one of the following ways:

- [Recommended] The load balancer and all load balancer members are installed on the same machine.
- The load balancer and load balancer members are all installed on different physical machines. Note that at least one load balancer member must be installed on the same machine as the load balancer.

Remote installation of Service Layer is not supported. For example, if you intend to install the load balancer on server A and two load balancer members on servers B and C, you must run the server components setup wizard on each server separately.

For more information, see the *Installing the Service Layer* charpter in the *SAP Business One Administrator's Guides (version for SQL and version for SAP HANA)*.

3 Consuming SAP Business One Service Layer

This section explains how to consume SAP Business One Service Layer and provides examples. For a full list of exposed entities and actions, refer to metadata returned by your service or the API reference of SAP Business One Service Layer.

Before interacting with Service Layer, refer to the following table for the key elements and terms:

Key Elements and Terms	Description/Activity	URL/Sample Code

Service Root URL	Identifies the root of Service Layer API. Service layer supports HTTPS by default.	`https://<server>:<port>/b1s/<version>`
		Example:
		`https://databaseserver:50000/b1s/v1`
		Note
		To use OData version 3, send the following HTTP request:
		`https://databaseserver:50000/b1s/v1`
		To use OData version 4, send the following HTTP request:
		`https://databaseserver:50000/b1s/v2`

Resource Path	Identifies the resource to be interacted with. It can be a collection of entities or a single entity.	`https://<server>:<port>/b1s/<version>/<resource_path>`
		Example:
		`https://databaseserver:50000/b1s/v1/Items`

Query Options	Specifies multiple query options and operation parameters.	`https://<server>:<port>/b1s/<version>/<resource_path>?<query_options>`
		Example:
		`https://databaseserver:50000/b1s/v1/Items?$top=2&$orderby=itemcode`

HTTP Verb	Indicates the action to be taken against the resource, in accordance with the RESTful architectural principles.	In the following example, the 2 requests are equivalent: · `POST https://databaseserver/b1s/v1/Login` · `POST /Login`
JSON Resource Representation	Represents and interacts with structured content, embedded in Service Layer requests and responses.	`{"key1": "value1", "arr1": [100, 200], "key2": "value2"}`

Recommendation

To test Service Layer without developing a program, you can install the "POSTMAN" browser extension in Google Chrome, or install equivalent add-ons on other browsers.

3.1 Login and Logout

Before you perform any operation in Service Layer, you first need to log into Service Layer.

Send this HTTP request for login:

```
POST https://<Server Name/IP>:<Port>/b1s/v1/Login
{"CompanyDB": "US506", "UserName": "manager", "Password": "1234"}
```

If the login is successful, you get the following response:

```
HTTP/1.1 200 OK
Set-Cookie: B1SESSION=PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy;HttpOnly;
Set-Cookie: ROUTEID=.node1; path=/b1s

{
    "odata.metadata":
"https://databaseserver:50000/b1s/v1/$metadata#B1Sessions/@Element",
    "SessionId": "PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy",
    "Version": "1000110",
    "SessionTimeout": 30
}
```

The response of `Login` request indicates that Service Layer inserts a cookie in the response header, with the cookie name 'B1SESSION' and cookie value 'PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy' respectively. In addition, another cookie item (ROUTEID=.node1) is returned by Apache server to ensure the load balancer stickiness.

Send this HTTP request for logout:

```
POST /Logout
Cookie: B1SESSION=PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy; ROUTEID=.node1
```

If the logout is successful, you get the following response, without any response content:

```
HTTP/1.1 204 No Content
```

3.1.1 Session

A session is started by a login request and is ended by a logout request. Each valid session has a unique session ID which is distinguished by a GUID-like string. To make subsequent requests after login, the cookie items `B1SESSION` and `ROUTEID` are mandatory and shall both be set in each request header. For example, to get an `Item` with ID='i001', send the following request with a cookie:

```
GET /Items('i001')
Cookie: B1SESSION=PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy; ROUTEID=.node1
```

If you write a client application in Windows desktop mode (not in Browser Access mode), do not forget to add the cookie item in the HTTP header, as in the above example of Logout. Otherwise, you may receive the "Invalid session" error:

```
HTTP/1.1 401 Unauthorized
```

```
{
   "error" : {
      "code" : 301,
      "message" : {
         "lang" : "en-us",
         "value" : "Invalid session or session already timeout."
      }
   }
}
```

> **Note**
>
> If your application is written in JavaScript and runs in Browser Access mode, you do not need to set the cookie each time you send a request, since most Web browsers are able to handle the cookie transparently.

3.2 Single Sign-On (SSO)

As of SAP Business One 9.2, version for SAP HANA, besides the normal login, Service Layer supports SSO as well.

Service Layer, as a service provider (SP), implements the SSO process via the SAML2 protocol on top of SLD, which functions as an identity service provider (IDP). Service Layer supports two SSO methods:

- HTTP-POST: used in the browser environment
- PAOS (Reverse HTTP Binding for SOAP Specification): used for the non-browser client

It is recommended that Service Layer be deployed on the same Linux machine as SLD. If you deploy it on two separate machines, the time must be synced between the two machines before the SSO process is implemented.

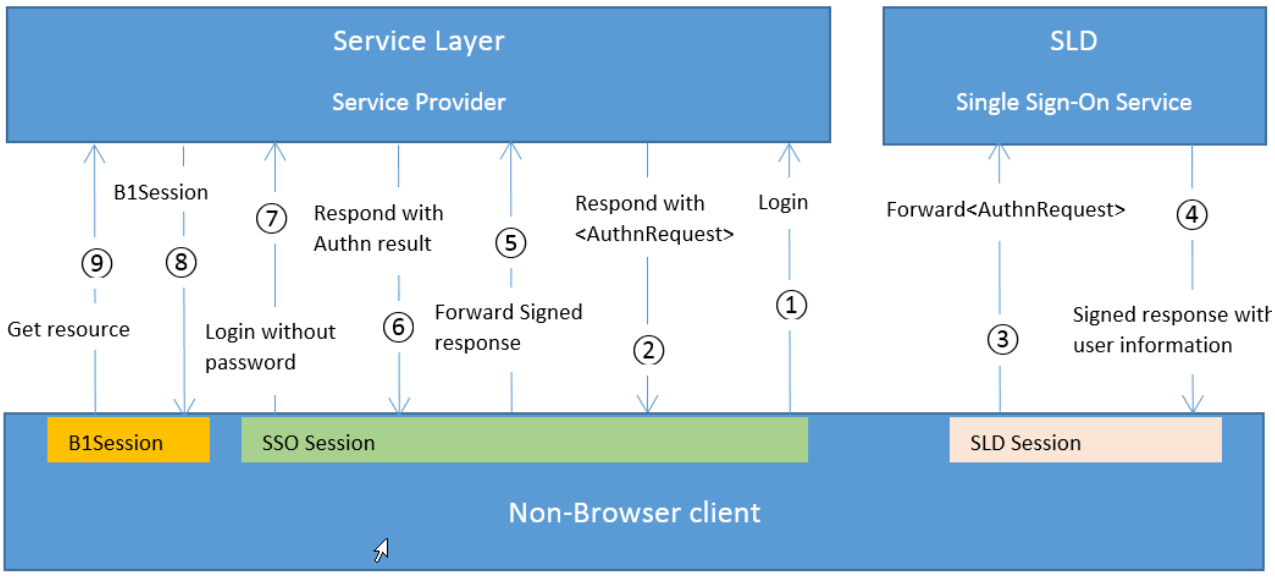
> **Note**
>
> Service Layer SSO does not work with windows account bound using service principal name (SPN).

3.2.1 SSO via PAOS

Prerequisites

In order to maintain a session, it is essential to first log in to SLD, before logging in to Service Layer via SSO.

SSO Flow



Steps Description

1. Post a login request to Service Layer with the PAOS header as below:

   ```
   POST /b1s/v1/ssob1s/ HTTP/1.1
   Accept: application/vnd.paos+xml
   PAOS: ver='urn:liberty:paos:2003-08';'urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp'
   ```

2. From the response header, get the `JSESSIONID` from `Set-Cookie` header as the SSO session between Service Layer and the client:

   ```
   Set-Cookie: JSESSIONID=5AE19B213830AF766BCC477DF4A40AE5;
   ```

   Add `JSESSIONID` in subsequent requests to Service Layer, so that the client can maintain a session with Service Layer for the whole SSO process. From the response body, get the IDP location and the soap body content.

3. Forward the soap body content from step 2 to the IDP location, with the session ID maintained between the client and SLD as the request cookie.
4. IDP issues a signed response containing user login information.
5. The client forwards the signed response to Service Layer, with the SSO session ID retrieved from step 2.
6. If authentication is passed, Service Layer responds with the http status 200.
7. The client posts a login request to Service Layer with the SSO Session ID in the request cookie as below:

   ```
   POST /b1s/v1/ssob1s HTTP/1.1
   Cookie: JSESSIONID=5AE19B213830AF766BCC477DF4A40AE5
   ```

8. On success, Service Layer responds with:

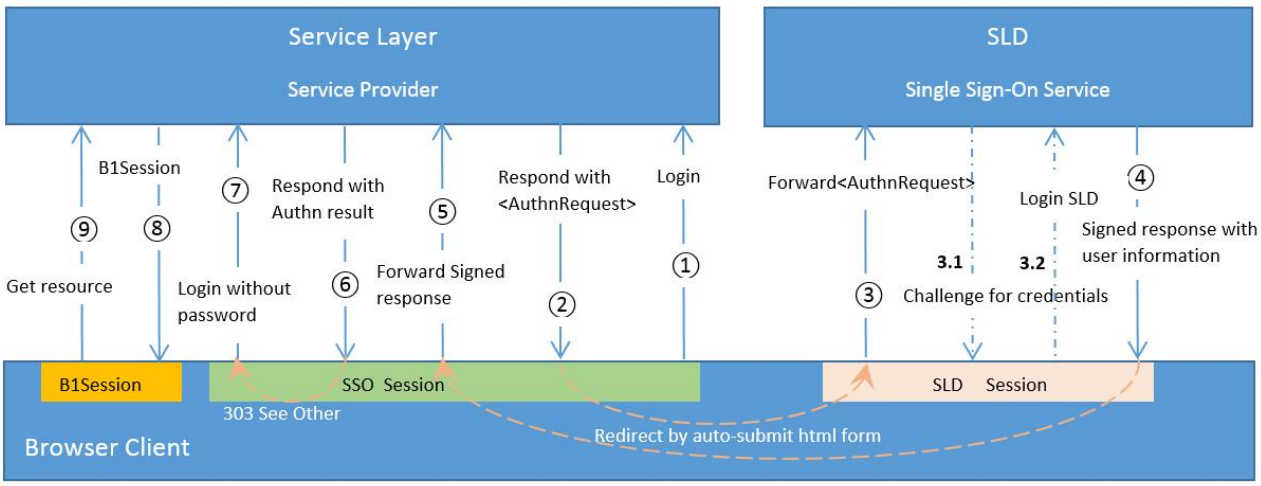
   ```
   HTTP/1.1 200 OK
   Set-Cookie: B1SESSION=PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy;HttpOnly;
   Set-Cookie: ROUTEID=.node1; path=/b1s

   { "SessionId" : "PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy", "Version" : "920110",
   "SessionTimeout" : 30 }
   {
       "odata.metadata":
   "https://databaseserver:50000/b1s/v1/$metadata#B1Sessions/@Element",
       "SessionId": "PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy",
       "Version": "1000110",
       "SessionTimeout": 30
   }
   ```

9. With the `B1SESSION` and `ROUTEID`, clients are allowed to access the resources of Service Layer.

3.2.2 SSO via HTTP-POST

SSO Flow



Steps Description

The SSO flow via the browser client is basically the same as SSO via PAOS, except for the following:

1. Browser clients finish the SSO process via HTTP-POST.
2. Browser clients automatically redirect the request to and the response from Service Layer by auto-submitting HTML forms.
3. Between step 3 and step 4, a login SLD form is returned if the user does not log in to SLD before the start of the SSO process. After the SLD login, the response of step 4 is to redirect to Service Layer.

3.3 Metadata Document

Metadata describes the capability of the service. It mainly defines types, entities (for example, SAP Business One objects) and actions (for example, SAP Business One services).

Send the following HTTP request to retrieve metadata:

```
GET /$metadata
```

Using SAP Business One business partners and sales orders as examples, you can see the following sections in the metadata:

```
<!-- section 1.1 -->
<EnumType Name="BoCardTypes">
  <Member Name="cCustomer" Value="C"/>
  <Member Name="cSupplier" Value="S"/>
  <Member Name="cLid" Value="L"/>
</EnumType>

<!-- section 1.2 -->
<EntityType Name="BusinessPartner">
  <Key>
    <PropertyRef Name="CardCode"/>
  </Key>
  <Property Name="CardCode" Nullable="false" Type="Edm.String"/>
  <Property Name="CardName" Type="Edm.String"/>
  <Property Name="CardType" Type="SAPB1.BoCardTypes"/>
  ...
</EntityType>

<!-- section 1.3 -->
<ComplexType Name="DocumentParams">
  <Property Name="DocEntry" Nullable="false" Type="Edm.Int32"/>
</ComplexType>

<!-- section 1.4 -->
<EntityType Name="Document">
  <Key>
    <PropertyRef Name="DocEntry"/>
  </Key>
  <Property Name="DocEntry" Nullable="false" Type="Edm.Int32"/>
  <Property Name="DocNum" Type="Edm.Int32"/>
  <Property Name="DocType" Type="SAPB1.BoDocumentTypes"/>
  ...
```

```
  <Property Name="DocumentLines" Type="Collection(SAPB1.DocumentLine)"/>
  ...
</EntityType>

<!-- section 1.5 -->
<ComplexType Name="DocumentLine">
  <Property Name="LineNum" Nullable="false" Type="Edm.Int32"/>
  <Property Name="ItemCode" Type="Edm.String"/>
  <Property Name="ItemDescription" Type="Edm.String"/>
  <Property Name="Quantity" Type="Edm.Double"/>
  ...
</ComplexType>

<!-- section 2 -->
<Action IsBindable="true" Name="Close">
  <Parameter Name="Document" Type="SAPB1.Document"/>
</Action>

<!-- section 3 -->
<EntityContainer Name="ServiceLayer">
  <EntitySet EntityType="SAPB1.BusinessPartner" Name="BusinessPartners"/>
  <EntitySet EntityType="SAPB1.Document" Name="Orders"/>
  ...
</EntityContainer>
```

The above metadata sections indicate how the entities and actions are exposed:

- In Section 3, you can see that entities `BusinessPartners` and `Orders` are exposed. You can perform standard create/retrieve/update/delete (CRUD) operations on them.
- In Section 2, you can see that a bindable action named `Close` is defined and can be bound to type `SAPB1.Document`. As orders are of this entity type, therefore, orders has a `Close` action (`POST /Orders(id)/Close`).

> **Note**
>
> Metadata for UDFs/UDTs/UDOs:
>
> In SAP Business One 9.1 patch level 05 and later, information from the user-defined fields (UDFs), user-defined tables (UDTs) and user-defined objects (UDOs) is added to the metadata. As different SAP Business One company databases have different UDFs/UDTs/UDOs, the metadata of the service may vary if you connect to a different company database.
>
> For UDTs, only the "no object" type is added to the metadata. UDTs are treated as simple entities that have only one main table. Thus, third-party tools, such as MS WCF, can generate code for UDFs/UDTs/UDOs from the metadata.

3.4 Service Document

The service document is a list of exposed entities. Use the root service URL to retrieve the service document. Send the HTTP request:

```
GET /
```

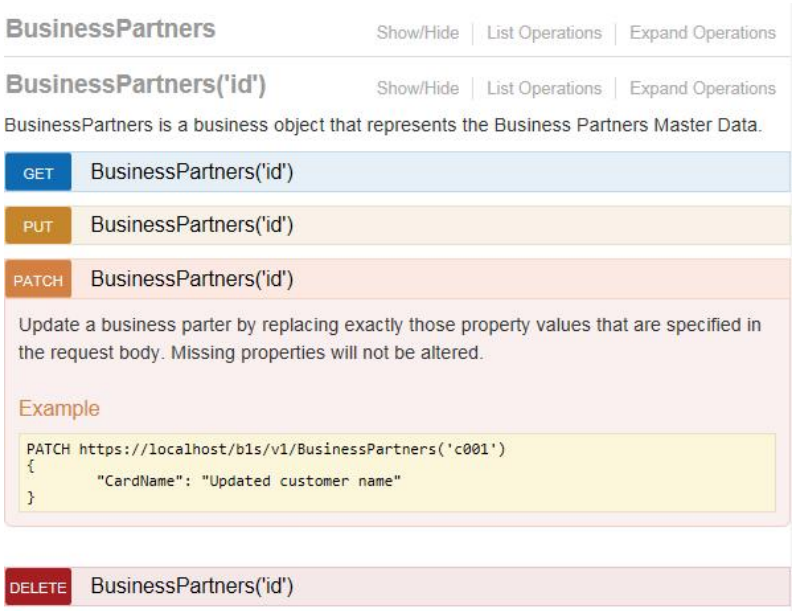
The response is:

```
HTTP/1.1 200 OK

{
    "value": [
        {
            "name": "ChartOfAccounts",
            "kind": "EntitySet",
            "url": "ChartOfAccounts"
        },
        {
            "name": "SalesStages",
            "kind": "EntitySet",
            "url": "SalesStages"
        },
        ...
       ]
}
```

3.5 Create/Retrieve/Update/Delete (CRUD) Operations

OData protocol defines a standard way to create/retrieve/update/delete (CRUD) an entity. The CRUD operations are all similar. You can refer to the API reference document for details (see the screenshot below).



3.5.1 Creating Entities

Use the HTTP verb `POST` and the content of an entity to create the entity.

For most cases, the response on success is also the content of the entity.

Example

How to create a customer (business partner) named "c1"

Send this HTTP request:

```
POST /BusinessPartners
{
    "CardCode": "c1",
    "CardName": "customer c1",
    "CardType": "cCustomer"
}
```

All valid fields are defined in its type - `SAPB1.BusinessPartner` in metadata section 1.2.

Note that `CardType` is of type `Enumeration` (`BoCardTypes`, defined in metadata section 1.1). Both the enumeration name and value are accepted by Service Layer. So these two statements are equivalent:

```
{"CardType": "cCustomer",}
{"CardType": "C",}
```

On success, the server returns HTTP code `201` (Created) and the content of the entity is as follows:

```
HTTP/1.1 201 Created

{
```

```
    "CardCode": "c1",
    "CardName": "customer c1",
    "CardType": "cCustomer",
    "GroupCode": 100,
    ...
}
```

On error, the server returns HTTP code 4XX (for example, 400) and the error message as content is as follows (suppose customer "c1" exists):

```
HTTP/1.1 400 Bad Request

{
    "error": {
        "code": -10,
        "message": {
            "lang": "en-us",
            "value": "1320000140 - Business partner code 'c1' already assigned
to a business partner; enter a unique business partner code"
        }
    }
}
```

Example

How to create a sales order with two document lines

The POST content - entity `Orders` - is of type `Document` and defined in metadata section 1.4. `DocumentLines`, known as the sub-object of sales order, is a collection of the complex type `DocumentLine`, which is defined in metadata section 1.5. In JSON format, it is an array in square brackets [].

Send this HTTP request:

```
POST /Orders

{
    "CardCode": "c1",
    "DocDate": "2014-04-01",
    "DocDueDate": "2014-04-01",
    "DocumentLines": [
        {
            "ItemCode": "i1",
            "UnitPrice": 100,
            "Quantity": 10,
            "TaxCode": "T1",
        },
        {
```

```
            "ItemCode": "i2",
            "UnitPrice": 120,
            "Quantity": 8,
            "TaxCode": "T1",
        },
    ]
}
```

On success, the server returns `201` (Created) and the content of the entity is as follows:

```
HTTP/1.1 201 Created

{
    "DocEntry": 22,
    "DocNum": 11,
    "DocType": "dDocument_Items",
    ...
    "DocumentLines": [
        {
            "LineNum": 0,
            "ItemCode": "i1",
            ...
        },
        {
            "LineNum": 1,
            "ItemCode": "i2",
            ...
        }
    ],
    ...
}
```

3.5.2 Retrieving Entities

Use the HTTP verb `GET` and the key fields to retrieve the entity.

 Example

How to get the customer "c1" in the previous example

As defined in metadata section 1.2, `CardCode` is the key property (type is string). To retrieve the customer "c1", send the HTTP request:

```
GET /BusinessPartners('c1')
```

or

```
GET /BusinessPartners(CardCode='c1')
```

The service returns HTTP code 200 that indicates success with the content of the object in JSON format:

```
HTTP/1.1 200 OK

{
    "CardCode": "c1",
    "CardName": "customer c1",
    "CardType": "cCustomer",
    "GroupCode": 100,
    ...
}
```

 Example

How to get the sales order in the previous example

As defined in metadata section 1.4, DocEntry is the key property (type is Int32). To retrieve the sales order, send the HTTP request:

```
GET /Orders(22)
```

or

```
GET Orders(DocEntry=22)
```

Note

Single quotes are required for string values such as 'c1', and no single quotes around integer values such as 22.

If the entity key contains multiple properties, send the HTTP request:

```
GET /SalesTaxAuthorities(Code='AK',Type=-3)
```

3.5.3 Updating Entities

Use the HTTP verb PATCH or PUT to update the entity. Generally, PATCH is recommended.

The difference between PATCH and PUT is that PATCH ignores (keeps the value) those properties that are not given in the request, while PUT sets them to the default value or to null.

 Example

How to update the name of the customer "c1"

Send the HTTP request:

```
PATCH /BusinessPartners('c1')

{
    "CardName": "Updated customer name"
}
```

On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

Note

Read-only properties (for example, `CardCode`) cannot be updated. They are ignored silently if assigned in the request.

3.5.4 Deleting Entities

Use the HTTP verb `DELETE` and the key fields to delete the entity.

Example

How to delete the customer "c1"

Send the HTTP request:

```
DELETE /BusinessPartners('c1')
```

On success, HTTP code `204` is returned without content.

```
HTTP/1.1 204 No Content
```

Note

You cannot delete the sales order in SAP Business One. If you try to delete the sales order No.22:

```
DELETE /Orders(22)
```

An error is reported to deny the operation:

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": -5006,
        "message": {
            "lang": "en-us",
            "value": "The requested action is not supported for this object."
        }
    }
}
```

3.5.5 Create Entity with No Content

Considering the fact that returning all the entity content on creating one entity may be not suitable for the high performance demanding scenario, Service Layer provides a way to respond no content by specifying a special header `Prefer` with the value `return-no-content`. For example:

```
POST /b1s/v1/Items HTTP/1.1
Prefer: return-no-content
```

```
{
    "ItemCode": "i011"
}
```

On success, HTTP code 204 is returned without content, instead of having the usual 201 resource created.

```
HTTP/1.1 204 No Content
Location: /b1s/v1/Items('i011')
Preference-Applied: return-no-content
```

> **Note**
>
> Response header includes `Preference-Applied` to confirm that the server accepts this preference option.
>
> The URI of the created resource is in the `Location` header.

3.6 Actions

Besides the basic entity CRUD operations, Service Layer provides you with two kinds of actions:

- Bound action (bound to entity for operations other than CRUD)
- Global action (mainly used to expose SAP Business One services)

The request and response for each action are described in the metadata. For example, the login function that was introduced above is a global action. You can find its definition in metadata.

> **Note**
>
> "Action" is an OData version 4 concept. In OData version 3, it is called "FunctionImport".

You can use the HTTP verb `POST` for OData actions.

> **Example**
>
> How to use the bound action
>
> In the metadata section 2.1, you can see a bindable action named "Close" with the first parameter bound to the `Document` type:
>
> ```
> <!-- section 2.1 -->
> <Action IsBindable="true" Name="Close">
>   <Parameter Name="Document" Type="SAPB1.Document"/>
> </Action>
> ```
>
> As orders are of type `Document`, that means orders have a "Close" action. You can send the following HTTP request to close the document No. 22:
>
> ```
> POST /Orders(22)/Close
> ```

Example

How to use the global action

In SAP Business One DI API, you can use the SAPbobsCOM.Activity object to operate the activities in SAP Business One. However, in SAP Business One 9.1 patch level 01, from Service Layer, you cannot find the Activity entity. Then how to use it?

By searching in metadata, you can find the action definitions, as follows:

```
<Action Name="ActivitiesService_GetActivity">
  <Parameter Name="ActivityParams" Type="SAPB1.ActivityParams"/>
  <ReturnType Type="SAPB1.Activity"/>
</Action>

<Action Name="ActivitiesService_AddActivity">
  <Parameter Name="Activity" Type="SAPB1.Activity"/>
  <ReturnType Type="SAPB1.ActivityParams"/>
</Action>
```

Note that the example follows the format of OData version 4. For OData version 3, "FunctionImport" is used instead of "Action". The result is as follows:

```
<FunctionImport Name="ActivitiesService_GetActivity">
  <Parameter Name="ActivityParams" Type="SAPB1.ActivityParams"/>
  <ReturnType Type="SAPB1.Activity"/>
</FunctionImport>

<FunctionImport Name="ActivitiesService_AddActivity">
  <Parameter Name="Activity" Type="SAPB1.Activity"/>
  <ReturnType Type="SAPB1.ActivityParams"/>
</FunctionImport>
```

It shows that you can use ActivitiesService to get and add activity objects. The related types are also defined in metadata, as follows:

```
<ComplexType Name="ActivityParams">
  <Property Name="ActivityCode" Nullable="false" Type="Edm.Int32"/>
</ComplexType>

<ComplexType Name="Activity">
  <Property Name="ActivityCode" Nullable="false" Type="Edm.Int32"/>
  <Property Name="CardCode" Type="Edm.String"/>
  <Property Name="Notes" Type="Edm.String"/>
  ...
</ComplexType>
```

To add an activity, send the HTTP request:

```
POST /ActivitiesService_AddActivity
{
    "Activity":{
        "ActivityCode": 1,
        "CardCode": "c1"
    }
}
```

On success, it returns the content of type `SAPB1.ActivityParams` as defined.

To get an activity, send the HTTP request:

```
POST /ActivitiesService_GetActivity
{
    "ActivityParams": {
        "ActivityCode": 1
    }
}
```

On success, it returns the content of type `SAPB1.Activity` as defined.

Note that from SAP Business One 9.1 patch level 02 and later, "Activity" has been exposed as an entity, and, therefore, the global actions were hidden by default.

Example

Previewing an order

A hidden action named `OrdersService_Preview` allows you to preview an order to create without actually creating it. Its metadata is as follows:

```
<Action Name="OrdersService_Preview">
  <Parameter Name="Document" Type="SAPB1.Document"/>
  <ReturnType Type="SAPB1.Document"/>
</Action>
```

An order to create can be previewed this way:

```
POST /b1s/v1/OrdersService_Preview

{
  "Document": {
    "CardCode": "c1",
    "DocDate": "2014-04-01",
    "DocDueDate": "2014-04-01",
    "DocumentLines": [
      {
        "ItemCode": "i1",
```

```
        "UnitPrice": 100,
        "Quantity": 10,
        "TaxCode": ""
      }
    ]
  }
}
```

On success, the server returns HTTP code 200 (OK) and part of the response is as follows:

```
HTTP/1.1 200 OK

{
  "DocEntry": null,
  "DocNum": null,
  "DocType": "dDocument_Items",
  "Printed": "psNo",
  "DocDate": "2014-04-01",
  "DocDueDate": "2014-04-01",
  "CardCode": "c1",
  "CardName": "customer 1",
  "DocTotal": 1000,
  "DocCurrency": "$",
  "JournalMemo": "Sales Orders - 0af75168-60cd-4",
  "TaxDate": "2014-04-01",
  "DocObjectCode": "17",
  "DocTotalSys": 1000,
  "DocumentStatus": "bost_Open",
  "TotalDiscount": 0,
  "DocumentLines": [
    {
      "LineNum": 0,
      "ItemCode": "i1",
      "ItemDescription": "i01",
      "Quantity": 10,
      "ShipDate": "2014-04-01",
      "Price": 100,
      "PriceAfterVAT": 100,
      "Currency": "$",
      "WarehouseCode": "01",
      "AccountCode": "_SYS00000000081",
      "TaxCode": "",
```

```
      "LineTotal": 1000,
      "TaxTotal": 0,
      "UnitPrice": 100,
      "LineStatus": "bost_Open",
      "PackageQuantity": 10,
      "LineType": "dlt_Regular",
      "OpenAmountSC": 1000,
      "DocEntry": null,
      "UoMCode": "Manual",
      "InventoryQuantity": 10,
.....
    }
  ],
  ......
}
```

3.7 Query Options

Query options within the request URL can control how a particular request is processed by Service Layer. The following table shows the query options supported by Service Layer.

Option	Description	Example

`$filter`	Queries collections of entities.
	Currently supported functions for `$filter` are:
	· `startswith`
	· `endswith`
	· `contains`
	· `substringof`
	Currently supported logical and relational operators include:
	· `and`
	· `or`
	· `le` (less than or equal to)
	· `lt` (less than)
	· `ge` (greater than or equal to)
	· `gt` (greater than)
	· `eq` (equal to)
	· `ne` (not equal to)
	· `not`
	Note
	The operator `not` is supported as of 9.1 patch level 01.
	Parentheses are also supported.
		`/Orders?$filter=DocTotal gt 3000`
		
		`/Orders?$filter=DocEntry lt 8 and (DocEntry lt 8 or DocEntry gt 116) and CardCode eq 'c1'`
		
		`/Orders?$filter=DocEntry lt 8 and ((DocEntry lt 8 or DocEntry gt 116) and startswith(CardCode,'c1'))`
		
		`/Items?$filter=not (startswith(ItemName, 'item') and ForeignName eq null)`

`$select`	Returns the properties that are explicitly requested.	`/Orders?$select=DocEntry, DocTotal`
`$orderby`	Specifies the order in which entities are returned.	`/Orders?$orderby=DocTotal asc, DocEntry desc`
`$top`	Returns the first n (non-negative integer) records.	`/Orders?$top=3`
`$skip`	Specifies the result excluding the first n entities.	`/Orders?$top=3&$skip=2` Where `$top` and `$skip` are used together, the `$skip` is applied before the `$top`, regardless of the order of appearance in the request.
`$count`	Returns the count of an entity collection.	`/Orders/$count`

Option	Description	Example
		`/Items/$count?&filter=ItemCode eq 'test'`
`$inlinecount`	Allows clients to request the number of matching resources inline with the resources in the response. Note `$inlinecount` query option applies to OData 3.0 protocols only. This feature is available in SAP Business One 9.1 patch level 06 and later.	For more information, see the section *inlinecount* below.

The combination of query options enables Service Layer to support any complex query scenarios, while keeping the API interface as simple as possible.

3.7.1 Get All Entities

You can use the following ways to get all entity records:

```
GET /Items
```

or

```
GET /Items?$select=*
```

3.7.2 Get Fields of an Entity

You can use the following ways to get item fields:

```
GET /Items('i1')?$select=ItemCode,ItemName,ItemPrices
```

or

```
GET /Items(ItemCode='i1')?$select=ItemCode,ItemName,ItemPrices
```

3.7.3 Query Properties of the Enumeration Type

Enumeration value and enumeration name are both supported in a query option. You can use the following ways to get all customers:

```
GET /BusinessPartners?$filter=CardType eq 'C'
```

or

```
GET /BusinessPartners?$filter=CardType eq 'cCustomer'
```

Note that 'C' is an enumeration value while 'cCustomer' is an enumeration name.

3.7.4 Query Properties of the Datetime Type

Multiple date formats are supported. For example:

```
GET /Orders?$filter=DocDate eq '2014-04-23'
GET /Orders?$filter=DocDate eq '20140423'
GET /Orders?$filter=DocDate eq datetime'2014-04-23'
GET /Orders?$filter=DocDate eq datetime'20140423'
GET /Orders?$filter=DocDate eq '2014-04-23T12:21:21'
GET /Orders?$filter=DocDate eq '20140423000000'
```

Note that SAP Business One ignores the HOUR/MINUTE/SECOND parts. The `datetime` keyword prefix can also be added before the datetime value.

3.7.5 Query Properties of the Time Type

Multiple time formats are supported. For example:

```
GET /Orders?$filter=DocTime eq '18:38:00'
GET /Orders?$filter=DocTime eq '18:38'
GET /Orders?$filter=DocTime eq '183800'
GET /Orders?$filter=DocTime eq '1838'
GET /Orders?$filter=DocTime eq '2014-06-18T18:38:00Z'
GET /Orders?$filter=DocTime eq '2014-06-18T18:38'
```

Note that SAP Business One ignores the YEAR/MONTH/DAY parts; only the HOUR/MINUTE parts are effective.

3.7.6 Paginate the Selected Orders

The pagination mechanism is implemented through top and skip. It allows the data to be fetched chunk by chunk. For example, after you send the HTTP request:

```
GET /Orders
```

The service returns:

```
HTTP/1.1 200 OK

{
       "value": [
              {"DocEntry": 7,"DocNum": 2,...},
              {"DocEntry": 8,"DocNum": 3,...},
              ...
              {"DocEntry": 26,"DocNum": 21,...}
       ],
```

```
    "odata.nextLink": "/b1s/v1/Orders?$skip=20"
}
```

Annotation `odata.nextLink` is contained in the body for the link of the next chunk.

> **Note**
>
> For OData V3, the next link annotation is `odata.nextLink`; For OData V4, the next link annotation is `@odata.nextLink`.
>
> The default page size is 20. You can customize the page size by changing the following options:
>
> - Set the configuration option `PageSize` in `conf/b1s.conf`.
> - Use the OData recommended annotation `odata.maxpagesize` in the `Prefer` header of the request:
>
>   ```
>   GET /Orders
>   Prefer:odata.maxpagesize=50
>   ... (other headers)
>   ```
>
>   The response contains HTTP header `Preference-Applied` to indicate whether and how the request is accepted:
>
>   ```
>   HTTP/1.1 200 OK
>   Preference-Applied: odata.maxpagesize=50
>   ...
>   ```
>
> If `PageSize` or `odata.maxpagesize` is set to 0, the pagination mechanism is turned off.
>
> The by-request option `odata.maxpagesize` is prior to the configuration option `PageSize`.

3.7.7 Aggregation

As of SAP Business One 9.1 patch level 12, version for SAP HANA, aggregation is partly supported by Service Layer.

Aggregation behavior is triggered using the query option `$apply`. Any aggregate expression that specifies an aggregation method MUST define an alias for the resulting aggregated value. Aggregate expressions define the alias using the "as" keyword, followed by a `SimpleIdentifier`. The alias will introduce a dynamic property in the aggregated result set. The introduced dynamic property is added to the type containing the original expression.

Currently, the supported aggregation methods include sum, avg, min, max, count and distinctcount.

3.7.7.1 sum

The standard aggregation method sum can be applied to numeric values to return the sum of the non-null values, or null if there are no non-null values.

For example, to sum the DocRate of the Orders, send a request such as:

```
GET /b1s/v1/Orders?$apply=aggregate(DocRate with sum as TotalDocRate)
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(TotalDocRate)",
```

```
   "value" : [
      {
         "odata.id" : null,
         "TotalDocRate" : 4.0
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT SUM(T0."DocRate") AS "TotalDocRate" FROM "ORDR" T0
```

3.7.7.2 average

The standard aggregation method average can be applied to numeric values to return the sum of the non-null values divided by the count of the non-null values, or null if there are no non-null values.

For example, to calculate the average VatSum of the Orders, send a request such as:

```
GET /b1s/v1/Orders?$apply=aggregate(VatSum with average as AvgVatSum )
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(AvgVatSum)",
   "value" : [
      {
         "odata.id" : null,
         "AvgVatSum" : 1.70
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT AVG(T0."VatSum") AS "AvgVatSum" FROM "ORDR" T0
```

3.7.7.3 max

The standard aggregation method max can be applied to values with a totally ordered domain to return the largest of the non-null values, or null if there are no non-null values. The result property will have the same type as the input property.

For example, to get the maximum DocEntry of the Orders, send a request such as:

```
GET /b1s/v1/Orders?$apply=aggregate(DocEntry with max as MaxDocEntry)
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(MaxDocEntry)",
```

```
   "value" : [
      {
         "odata.id" : null,
         "MaxDocEntry" : 6
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT MAX(T0."DocEntry") AS "MaxDocEntry" FROM "ORDR" T0
```

3.7.7.4 min

The standard aggregation method min can be applied to values with a totally ordered domain to return the smallest of the non-null values, or null if there are no non-null values. The result property will have the same type as the input property.

For example, to get the minimum DocEntry of the Orders, send a request such as:

```
GET/b1s/v1/Orders?$apply=aggregate(DocEntry with min as MinDocEntry)
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(MinDocEntry)",
   "value" : [
      {
         "odata.id" : null,
         "MinDocEntry" : 2
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT MIN(T0."DocEntry") AS "MinDocEntry" FROM "ORDR" T0
```

3.7.7.5 countdistinct

The aggregation method countdistinct counts the distinct values, omitting any null values.

For example, to count the distinct CardCode of the Orders, send a request such as:

```
GET /b1s/v1/Orders?$apply=aggregate(CardCode with countdistinct as
CountDistinctCardCode)
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(CountDistinctCardCode)",
```

```
   "value" : [
      {
         "odata.id" : null,
         "CountDistinctCardCode" : "2"
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT COUNT(DISTINCT T0."CardCode") AS "CountDistinctCardCode" FROM "ORDR" T0
```

3.7.7.6 count

The value of the virtual property $count is the number of instances in the input set. It must always specify an alias and must not specify an aggregation method.

For example, to count the number of Orders, send a request such as:

```
GET /b1s/v1/Orders?$apply=aggregate($count as OrdersCount)
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(OrdersCount)",
   "value" : [
      {
         "odata.id" : null,
         "OrdersCount" : 4
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT COUNT(T0."DocEntry") AS "OrdersCount" FROM "ORDR" T0
```

3.7.7.7 inlinecount

The `$inlinecount` query option allows clients to request the number of matching resources in line with the resources in the response. This is most useful when a service implements server-side paging, as it allows clients to retrieve the number of matching resources even if the service decides to respond with only a single page of matching resources.

You must specify the `$inlinecount` query option with a value of `allpages` or none (or not specified); otherwise, the service returns an HTTP Status code of 400 Bad Request.

- The `$inlinecount` query option with a value of `allpages` specifies that the total count of entities matching the request must be returned along with the result. The following example returns the total number of banks in the result set along with the banks.

```
GET /Banks?$inlinecount=allpages
{
    "odata.count": "5",
    "value": [
        { "BankCode": "bank001", ... },
        { "BankCode": "bank002", ... },
        { "BankCode": "bank003", ... },
        { "BankCode": "bank004", ... },
        { "BankCode": "bank005", ... }
    ]
}
```

- The `$inlinecount` query option with a value of `none` (or not specified) signifies that the service should not return a count. For example:

```
GET /Banks?$inlinecount=none
{
    "value": [
        { "BankCode": "bank001", ... },
        { "BankCode": "bank002", ... },
        { "BankCode": "bank003", ... },
        { "BankCode": "bank004", ... },
        { "BankCode": "bank005", ... }
    ]
}
```

The `$inlinecount` query option can also work with `$top` and `$filter`.

- The following example returns the first two banks and the count of all banks.

```
GET /Banks?$inlinecount=allpages&$top=2
{
    "odata.count": "5",
    "value": [
        { "BankCode": "bank001", ... },
        { "BankCode": "bank002", ... }
    ]
}
```

- The following example returns the count of all banks with `BankCode` greater than "bank003".

```
GET /Banks?$inlinecount=allpages&$filter=BankCode gt 'bank003'
{
```

```
    "odata.count": "2",
    "value": [
        { "BankCode": "bank004", ... },
        { "BankCode": "bank005", ... }
    ]
}
```

3.7.8 Grouping

Grouping behavior is triggered using the query option `apply` and the `groupby` keyword. This keyword specifies the grouping properties, a comma-separated list of one or more single-valued property paths that is enclosed in parentheses. The same property path should not appear more than once; redundant property paths may be considered valid, but must not alter the meaning of the request.

> **Note**
>
> As of SAP Business One 9.2 PL03, version for SAP HANA, grouping is supported.

3.7.8.1 Simple Group

Simply enclose the group properties within parentheses. For example, to group the orders by `CardCode`, `DocEntry`, send the following request:

```
GET /b1s/v1/Orders?$apply=groupby((CardCode, DocEntry))
```

Or

```
/b1s/v1/Orders?$apply=groupby((Orders/CardCode, Orders/DocEntry))
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(CardCode, DocEntry)",
   "value" : [
      {
         "odata.id" : null,
         "CardCode" : "c001",
         "DocEntry" : 2
      },
      {
         "odata.id" : null,
         "CardCode" : "c002",
         "DocEntry" : 3
      },
      {
```

```
         "odata.id" : null,
         "CardCode" : "c001",
         "DocEntry" : 5
      },
      {
         "odata.id" : null,
         "CardCode" : "c001",
         "DocEntry" : 6
      }
   ]
}
```

The equivalent SQL on SAP database is:

```
SELECT T0."CardCode", T0."DocEntry" FROM "ORDR" T0 GROUP BY T0."CardCode",
T0."DocEntry"
```

3.7.8.2 Group with Aggregation Method

Service Layer also supports combining grouping with aggregation. For example, to aggregate the `DocNum` property on grouping `CardCode`, send the following request:

```
GET /b1s/v1/Orders?$apply=groupby((CardCode), aggregate(DocNum with sum as
TotalDocNum))
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(CardCode,TotalDocNum)",
   "value" : [
      {
         "odata.id" : null,
         "CardCode" : "c001",
         "TotalDocNum" : 8
      },
      {
         "odata.id" : null,
         "CardCode" : "c002",
         "TotalDocNum" : 2
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT T0."CardCode", SUM(T0."DocNum") AS "TotalDocNum" FROM "ORDR" T0  GROUP BY
T0."CardCode"
```

3.7.8.3 Group with Aggregation Method and Filter

SL allows you to filter before grouping. These two operations are separated by a forward slash (/) to express that they are consecutively applied. For example, to filter before grouping with the aggregation method, send the following request:

```
GET /b1s/v1/Orders?$apply=filter(Orders/CardCode eq 'c001')/groupby((CardCode),
aggregate(DocNum with sum as TotalDocNum))
```

On success, the response is as follows:

```
{
   "odata.metadata" : "$metadata#Orders(CardCode,TotalDocNum)",
   "value" : [
      {
         "odata.id" : null,
         "CardCode" : "c001",
         "TotalDocNum" : 8
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT T0."CardCode", SUM(T0."DocNum") AS "TotalDocNum" FROM "ORDR" T0  WHERE
T0."CardCode" = 'c001' GROUP BY T0."CardCode"
```

> **Note**
>
> The `filter` option can also be specified as below, which is functionally equivalent.
>
> ```
> GET /b1s/v1/Orders?$apply=groupby((CardCode), aggregate(DocNum with sum as
> TotalDocNum))&$filter=(Orders/CardCode ne 'c001')
> ```

3.7.9 Cross-Joins

Cross-Joins is supported as of SAP Business One 9.2, version for SAP HANA patch level 07.

OData supports querying related entities through defining navigation properties in the data model. These navigation paths help guide regular consumers in understanding and navigating relationships. In some cases, however, requests need to span entity sets with no predefined associations. Such requests can be sent to the special resource $crossjoin instead of to an individual entity set.

3.7.9.1 Cross-Joins with Expand

Expand across two entities

To expand across two entities according to given filter conditions, a request such as the one below,

```
GET /b1s/v1/$crossjoin(Orders,BusinessPartners)?$expand=Orders($select=DocEntry,
DocNum),BusinessPartners($select=CardCode)&$filter=Orders/CardCode eq
BusinessPartners/CardCode and Orders/DocNum le 3 and
startswith(BusinessPartners/CardCode,'c00')
```

results in:

```
{
   "odata.metadata" : "$metadata#Collection(Edm.ComplexType)",
   "value" : [
      {
         "BusinessPartners" : {
            "CardCode" : "c002"
         },
         "Orders" : {
            "DocEntry" : 3,
            "DocNum" : 2
         }
      },
      {
         "BusinessPartners" : {
            "CardCode" : "c001"
         },
         "Orders" : {
            "DocEntry" : 2,
            "DocNum" : 1
         }
      },
      {
         "BusinessPartners" : {
            "CardCode" : "c001"
         },
         "Orders" : {
            "DocEntry" : 5,
            "DocNum" : 3
         }
      }
   ]
}
```

The equivalent SQL on database is:

```
"SELECT T0."DocEntry", T0."DocNum", T1."CardCode" FROM  "ORDR" T0 ,"OCRD" T1  WHERE
T0."CardCode" = T1."CardCode"  AND  T0."DocNum" <= 3 AND T1."CardCode" Like 'c00%'
```

Expand across more entities

Service Layer supports expanding across more entities as well. A request such as the one below,

```
GET
/b1s/v1/$crossjoin(Orders,BusinessPartners,Activities)?$expand=Orders($select=DocEntry
,
DocNum),BusinessPartners($select=CardCode),Activities($select=ActivityCode)&$filter=Or
ders/CardCode eq BusinessPartners/CardCode and BusinessPartners/CardCode eq
Activities/CardCode
```

results in:

```
{
   "odata.metadata" : "$metadata#Collection(Edm.ComplexType)",
   "value" : [
      {
         "Activities" : {
            "ActivityCode" : 1
         },
         "BusinessPartners" : {
            "CardCode" : "c001"
         },
         "Orders" : {
            "DocEntry" : 2,
            "DocNum" : 1
         }
      },
      {
         "Activities" : {
            "ActivityCode" : 1
         },
         "BusinessPartners" : {
            "CardCode" : "c001"
         },
         "Orders" : {
            "DocEntry" : 5,
            "DocNum" : 3
         }
      },
      {
         "Activities" : {
            "ActivityCode" : 1
         },
         "BusinessPartners" : {
            "CardCode" : "c001"
```

```
         },
         "Orders" : {
            "DocEntry" : 6,
            "DocNum" : 4
         }
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT T0."DocEntry", T0."DocNum", T1."CardCode", T1."CardName", T2."ClgCode" FROM
"ORDR" T0 , "OCRD" T1, "OCLG" T2  WHERE T0."CardCode" = T1."CardCode" AND
T1."CardCode" = T2."CardCode"
```

More Examples

```
GET
/b1s/v1/$crossjoin(SalesOpportunities,BusinessPartners)?$expand=SalesOpportunities($se
lect=CardCode,CustomerName,StartDate),BusinessPartners($select=EmailAddress,
CardName)&$filter=SalesOpportunities/StartDate le '2017-09-20' and
BusinessPartners/CardCode eq SalesOpportunities/CardCode
```

```
{
    "odata.metadata": "$metadata#Collection(Edm.ComplexType)",
    "value": [
        {
            "SalesOpportunities": {
                "CardCode": "c2",
                "CustomerName": "customer c22",
                "StartDate": "2017-09-20"
            },
            "BusinessPartners": {
                "EmailAddress": null,
                "CardName": "customer c22"
            }
        },
        {
            "SalesOpportunities": {
                "CardCode": "c1",
                "CustomerName": "customer c11",
                "StartDate": "2017-09-20"
            },
            "BusinessPartners": {
```

```
                "EmailAddress": null,
                "CardName": "customer c11"
            }
        }
    ]
}
```

3.7.9.2 Cross-Joins with Calculation

Service Layer allows you to perform simple arithmetic operations on the selected properties and filter conditions. The supported operations include:

- add
- sub
- mul
- div

For example, a request such as the one below,

```
/b1s/v1/$crossjoin(Orders,BusinessPartners)?$expand=Orders($select=DocEntry mul
(DocNum sub 1) as DocSeq),BusinessPartners($select=CardCode,
CardName)&$filter=Orders/CardCode eq BusinessPartners/CardCode and Orders/DocEntry ge
Orders/DocNum sub 3
```

results in:

```
{
   "odata.metadata" : "$metadata#Collection(Edm.ComplexType)",
   "value" : [
      {
         "BusinessPartners" : {
            "CardCode" : "c001",
            "CardName" : null
         },
         "Orders" : {
            "DocSeq" : 0
         }
      },
      {
         "BusinessPartners" : {
            "CardCode" : "c002",
            "CardName" : null
         },
         "Orders" : {
```

```
            "DocSeq" : 3
         }
      },
      {
         "BusinessPartners" : {
            "CardCode" : "c001",
            "CardName" : null
         },
         "Orders" : {
            "DocSeq" : 18
         }
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT T0."DocEntry" * (T0."DocNum" - 1) AS "DocSeq", T1."CardCode", T1."CardName"
FROM  "ORDR" T0 , "OCRD" T1  WHERE T0."CardCode" = T1."CardCode"  AND  T0."DocEntry"
>= T0."DocNum" - 3
```

3.7.9.3 Cross-Joins with Aggregation

To aggregate the properties of `Orders` and `BusinessPartners`, send the following request:

```
/b1s/v1/$crossjoin(Orders,BusinessPartners)?$apply=filter(Orders/CardCode eq
BusinessPartners/CardCode)/groupby((BusinessPartners/CardCode,
Orders/DocEntry),aggregate(Orders(DocNum with countdistinct as DistinctDocNum)))
```

On success, the server replies this:

```
{
   "odata.metadata" : "$metadata#Collection(Edm.ComplexType)",
   "value" : [
      {
         "BusinessPartners" : {
            "CardCode" : "c001"
         },
         "Orders" : {
            "DistinctDocNum" : 1,
            "DocEntry" : 2
         }
      },
      {
```

```
         "BusinessPartners" : {
            "CardCode" : "c002"
         },
         "Orders" : {
            "DistinctDocNum" : 1,
            "DocEntry" : 3
         }
      },
      {
         "BusinessPartners" : {
            "CardCode" : "c001"
         },
         "Orders" : {
            "DistinctDocNum" : 1,
            "DocEntry" : 5
         }
      }
   ]
}
```

The equivalent SQL on database is:

```
SELECT T1."CardCode", T0."DocEntry", COUNT(DISTINCT T0."DocNum") AS "DistinctDocNum"
FROM "ORDR" T0 , "OCRD" T1  WHERE T0."CardCode" = T1."CardCode"   GROUP BY
T1."CardCode", T0."DocEntry"
```

Examples of max

```
GET /b1s/v1/$crossjoin(Orders,BusinessPartners)?$apply=filter(Orders/CardCode eq
BusinessPartners/CardCode)/groupby((BusinessPartners/CardCode),aggregate(Orders(DocNum
with max as MaxDocNum)))
```

is equivalent to:

```
SELECT T1."CardCode", MAX(T0."DocNum") AS "MaxDocNum" FROM "ORDR" T0 , "OCRD" T1 WHERE
T0."CardCode" = T1."CardCode" GROUP BY T1."CardCode"
```

Examples of count

```
GET /b1s/v1/$crossjoin(Orders,BusinessPartners)?$apply=filter(Orders/CardCode eq
BusinessPartners/CardCode)/groupby((BusinessPartners/CardCode),aggregate(Orders/$count
as CountDocEntry))
```

is equivalent to:

```
SELECT T1."CardCode", COUNT(T0."DocEntry") AS "CountDocEntry" FROM "ORDR" T0 , "OCRD"
T1 WHERE T0."CardCode" = T1."CardCode" GROUP BY T1."CardCode"
```

Note

Simply crossing join entities without any query options would not work, as this rarely has practical usage and would fetch large volumes of data under extreme conditions. For example, a request such as the one below,

```
/b1s/v1/$crossjoin(Orders,BusinessPartners)
```

results in:

```
{
    "error": {
        "code": -1000,
        "message": {
            "lang": "en-us",
            "value": "invalid $crossjoin query"
        }
    }
}
```

3.7.10 Row-Level Filter

As of SAP Business One 9.2 PL11, version for SAP HANA, Service Layer allows you to do row level filtering (for example, document line filtering).

To fully comply with OData, Service Layer exposes a new query service for the row level filter, which is implemented based on the `$crossjoin` capabilities by separating the `QueryPath` and `QueryOption` in the query URL.

3.7.10.1 Metadata for Query Service

Query Service is exposed in the manner of `FunctionImport` in the following way:

```
<FunctionImport Name="QueryService_PostQuery" ReturnType="Edm.String"
m:HttpMethod="POST">
    <Parameter Name="QueryOption" Type="Edm.String"/>
    <Parameter Name="QueryPath" Type="Edm.String"/>
</FunctionImport>
```

3.7.10.2 Examples for Query Service

Filter on joining document header and document line

A request such as the one below,

```
POST /b1s/v1/QueryService_PostQuery
{
  "QueryPath": "$crossjoin(Orders,Orders/DocumentLines)",
  "QueryOption": "$expand=Orders($select=DocEntry,
DocNum),Orders/DocumentLines($select=ItemCode,LineNum)&$filter=Orders/DocEntry eq
```

```
Orders/DocumentLines/DocEntry and Orders/DocEntry ge 3 and
Orders/DocumentLines/LineNum eq 0"
}
```

results in:

```
{
   "odata.metadata" : "$metadata#Collection(Edm.ComplexType)",
   "value" : [
      {
         "Orders" : {
            "DocEntry" : 9,
            "DocNum" : 5
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i1",
            "LineNum" : 0
         }
      },
      {
         "Orders" : {
            "DocEntry" : 12,
            "DocNum" : 6
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i1",
            "LineNum" : 0
         }
      },
      ...
      {
         "Orders" : {
            "DocEntry" : 20,
            "DocNum" : 12
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i1",
            "LineNum" : 0
         }
      },
      {
         "Orders" : {
```

```
            "DocEntry" : 44,
            "DocNum" : 22
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i1",
            "LineNum" : 0
         }
      }
   ]
}
```

Filter on joining document header and document line with parenthesis

A request such as the one below,

```
POST /b1s/v1/QueryService_PostQuery

{
  "QueryPath": "$crossjoin(Orders,Orders/DocumentLines)",
  "QueryOption": "$expand=Orders($select=DocEntry,
DocNum),Orders/DocumentLines($select=ItemCode,LineNum)&$filter=Orders/DocEntry eq
Orders/DocumentLines/DocEntry and (Orders/DocumentLines/LineNum eq 0 or
Orders/DocumentLines/LineNum eq 1 or Orders/DocumentLines/LineNum eq 2)"
}
```

results in:

```
{
   "odata.metadata" : "$metadata#Collection(Edm.ComplexType)",
   "value" : [
      {
         "Orders" : {
            "DocEntry" : 9,
            "DocNum" : 5
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i1",
            "LineNum" : 0
         }
      },
      {
         "Orders" : {
            "DocEntry" : 3,
            "DocNum" : 1
```

```
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i1",
            "LineNum" : 0
         }
      },
      ...
      {
         "Orders" : {
            "DocEntry" : 28,
            "DocNum" : 17
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i2",
            "LineNum" : 1
         }
      },
      {
         "Orders" : {
            "DocEntry" : 44,
            "DocNum" : 22
         },
         "Orders/DocumentLines" : {
            "ItemCode" : "i2",
            "LineNum" : 1
         }
      }
   ]
}
```

> **Note**
>
> The response is a raw string with the same structure as JSON and the *content-type* is `text/plain`. Some JSON utility libraries can be used to convert the response to a valid JSON structure to analyze.

3.7.11 Expand Query Enhancements

As of SAP Business One 10.0 FP 2105, the OData `$expand` query capability is enhanced. You can specify a `$select` clause in the `$expand`.

3.7.11.1 Single Entity

This enhancement allows a single entity to expand its navigation properties. For example, you send the following query to the Service Layer:

```
GET
https://hanaserver:50000/b1s/v2/ServiceCalls(1)?$expand=BusinessPartner($select=Contac
tPerson, CardCode),Item($select=ItemCode, ItemName)&$select=Subject HTTP/1.1
```

You get the following response:

```
{
    "@odata.context":
"https://hanaserver:50000/b1s/v2/$metadata#ServiceCalls/$entity",
    "BusinessPartner": {
        "CardCode": "c1",
        "ContactPerson": "contact001"
    },
    "Item": {
        "ItemCode": "i001",
        "ItemName": "i001 name"
    },
    "Subject": "subject1"
}
```

This query has the two variants below, which means the following two queries would produce the same result.

- ```
  GET
  https://hanaserver:50000/b1s/v2/ServiceCalls(1)?$expand=BusinessPartner($select=Con
  tactPerson, CardCode),Item&$select=Item/ItemCode, Item/ItemName, Subject  HTTP/1.1
  ```
- ```
  GET
  https://hanaserver:50000/b1s/v2/ServiceCalls(1)?$expand=BusinessPartner,Item&$selec
  t=BusinessPartner/ContactPerson, BusinessPartner/CardCode, Item/ItemCode,
  Item/ItemName, Subject HTTP/1.1
  ```

3.7.11.2 Collection Entity

This enhancement allows each entity in a collection to expand its navigation properties. For example, you send the following query to the Service Layer:

```
GET
https://hanaserver:50000/b1s/v2/ServiceCalls?$expand=BusinessPartner($select=ContactPe
rson, CardCode),Item($select=ItemCode, ItemName)&$select=Subject HTTP/1.1
```

You get the following response:

```
{
    "@odata.context": "https://hanaserver:50000/b1s/v2/$metadata#ServiceCalls",
    "value": [
```

```
        {
            "BusinessPartner": {
                "CardCode": "c1",
                "ContactPerson": "contact001"
            },
            "Item": {
                "ItemCode": "i001",
                "ItemName": "i001 name"
            },
            "Subject": "subject1"
        },
        {
            "BusinessPartner": {
                "CardCode": "c1",
                "ContactPerson": "contact001"
            },
            "Item": {
                "ItemCode": "i001",
                "ItemName": "i001 name"
            },
            "Subject": "subject2"
        },
        {
            "BusinessPartner": {
                "CardCode": "c1",
                "ContactPerson": "contact001"
            },
            "Item": {
                "ItemCode": "i001",
                "ItemName": "i001 name"
            },
            "Subject": "subject3"
        }
    ]
}
```

This query has the two variants below, which means the following two queries would produce the same results.

- `GET https://hanaserver:50000/b1s/v2/ServiceCalls?$expand=BusinessPartner($select=ContactPerson, CardCode),Item&$select=Item/ItemCode, Item/ItemName, Subject  HTTP/1.1`
- `GET https://hanaserver:50000/b1s/v2/ServiceCalls?$expand=BusinessPartner,Item&$select=B`

```
usinessPartner/ContactPerson, BusinessPartner/CardCode, Item/ItemCode,
Item/ItemName, Subject HTTP/1.1
```

3.8 SAP Business One Semantic Layer View Exposure

Semantic Layer View is for SAP Business One, version for SAP HANA only.

As of SAP Business One 9.3 PL02, version for SAP HANA, Service Layer supports to automatically discover and expose the Semantic Layer views, which are available upon deploying the SAP HANA models in SAP Business One analytics powered by SAP HANA. In this way, Semantic Layer works as an OData web service which is possible to be consumed by clients using OData protocol.

3.8.1 Views Deployment

Semantic Layer views are on top of SAP Business One Analytic Service and fall into two categories: system built-in views and customized views.

The author of the first category is generally SAP and you can deploy the views by following these steps after installing Analytic Service:

1. Open the SAP Business One analytics home page. (The URL is basically like: https://databaseserver:40000/Enablement/)
2. Navigate to the *Company* tab
3. Click the *Initialize* button to start the initialization process.

   After initialization, the views should be available in the Content package of the current SAP HANA instance in the SAP HANA studio.

For the customer view deployment, see subsequent sections.

3.8.2 View Exposure Scope

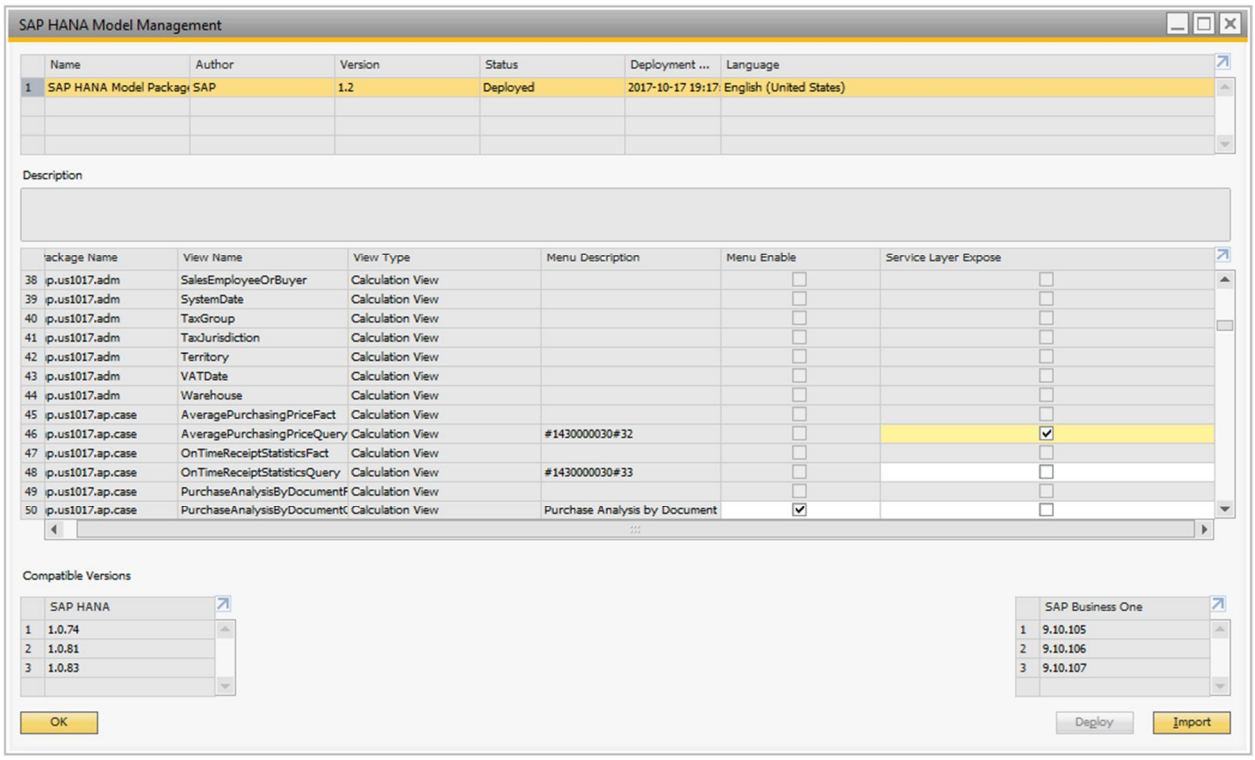
Semantic Layer has various kinds of views. Not all views are appropriate to be in the exposure scope.

- For the system built-in views, only the views satisfying all the below conditions are eligible for exposure:
  - With the calculation view type.
  - With the `Query` postfix in its name, for example, `SalesOrderDetailQuery`, `BalanceSheetQuery`, and so on.
- For the customized views, as long as the view is of type calculation, the view is eligible for exposure.

All eligible views are not exposed by default. To expose them, you can manually perform the following steps:

1. Start SAP Business One client.
2. Open the *SAP HANA Model Management* window.
3. Select views and check the corresponding *Service Layer Expose* checkbox.

4. Restart Service Layer to effect the changes.



3.8.3 View Exposure OData Version

Considering OData version 4 is the latest and prevalent protocol in OData world, Semantic Layer service is exposed in this version by default. Another advantage that comes with this is that implementing OData version 4 would make it possible for Semantic Layer service to be integrated with those SAP components (e.g. WEB IDE), which have supported or are going to support OData 4.

Meanwhile, OData 3 is supported as well, but it is not the default supported OData version. Clients must set the request header `OData-MaxVersion: 3.0` or `MaxDataServiceVersion: 3.0` to specify OData 3.

3.8.4 Semantic Layer Service Root

To distinguish Semantic Layer service from Service Layer, the root URL for this service is `/b1s/v1/sml.svc`.

Upon successfully accessing this URL, the response is as follows:

```
HTTP/1.1 200 OK

{
    "@odata.context": "https://databaseserver:50000/b1s/v1/sml.svc/$metadata",
    "value": [
```

```
        {
            "name": "PurchaseOrderFulfillmentCycleTimeQuery",
            "kind": "EntitySet",
            "url": "PurchaseOrderFulfillmentCycleTimeQuery"
        },
        {
            "name": "VendorBalanceAnalysisQuery",
            "kind": "EntitySet",
            "url": "VendorBalanceAnalysisQuery"
        },
        {
            "name": "AveragePurchasingPriceQuery",
            "kind": "EntitySet",
            "url": "AveragePurchasingPriceQuery"
        },
       ...
}
```

> **Note**
>
> `@odata.context` is one annotation from OData 4.

3.8.5 Semantic Layer Service Metadata

The service metadata URL is as follows:

```
GET /b1s/v1/sml.svc/$metadata
```

Upon successfully accessing the metadata, the service returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx Version="4.0"
    xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx">
    <edmx:DataServices>
        <Schema Namespace="SAPB1"
            xmlns="http://docs.oasis-open.org/odata/ns/edm">
            <EntityType Name="AveragePurchasingPriceQuery">
                <Key>
                    <PropertyRef Name="id__"/>
                </Key>
                <Property MaxLength="160" Name="LineDocumentOwner" Nullable="true"
Type="Edm.String"/>
```

```
                <Property MaxLength="15" Name="PaymentMethodCode" Nullable="true"
Type="Edm.String"/>
                <Property Name="PostingDateSQL" Nullable="true" Type="Edm.DateTime"/>
                            ...
                <Property Name="id__" Nullable="false" Type="Edm.Int32"/>
            </EntityType>
            <EntityType Name="OnTimeReceiptStatisticsQuery">
                <Key>
                    <PropertyRef Name="id__"/>
                </Key>
                <Property MaxLength="160" Name="DocumentOwner" Nullable="true"
Type="Edm.String"/>
                <Property Name="NumberOfPurchaseOrder" Nullable="true"
Type="Edm.Int32"/>
                <Property Name="id__" Nullable="false" Type="Edm.Int32"/>
            </EntityType>

            <EntityContainer Name="SemanticLayer">
                <EntitySet EntityType="SAPB1.PurchaseOrderFulfillmentCycleTimeQuery"
Name="PurchaseOrderFulfillmentCycleTimeQuery"/>
                    ...
                <EntitySet EntityType="SAPB1.BalanceSheetComparisonQueryParameter"
Name="BalanceSheetComparisonQueryParameters">
                    <NavigationPropertyBinding Path="BalanceSheetComparisonQuery"
Target="BalanceSheetComparisonQuery"/>
                </EntitySet>
                    ...
             <EntitySet EntityType="SAPB1.KPICashFlowStatementQueryParameter"
Name="KPICashFlowStatementQueryParameters">
                    <NavigationPropertyBinding Path="KPICashFlowStatementQuery"
Target="KPICashFlowStatementQuery"/>
                </EntitySet>
            </EntityContainer>
        </Schema>
    </edmx:DataServices>
</edmx:Edmx>
```

> **Note**
>
> Please refer to OData-CSDL (Common Schema Definition Language) for more information on the metadata format.
>
> `Version="4.0"` in the metadata indicates the service exposes resources with OData 4.

All Semantic Layer views are exposed as entities, as OData only allows to perform queries on entities. Due to the OData specification, each entity must at least have a primary key. However, this is contradictory to the fact that

views do not have keys from the database perspective. To address this issue in a generic way, a virtual property id__ is defined as the entity key for the typical views, as seen from the following example.

```
<!-->For the view sap.sbodemous.ar.doc/SalesOrderDetailQuery<-->
        <EntityType Name="SalesOrderDetailQuery">
             <Key>
                   <PropertyRef Name="id__"/>
               </Key
               <Property Name="DocumentNumber" Nullable="true" Type="Edm.Int32"/>
               <Property Name="Owner" Nullable="true" Type="Edm.String"/>
               <Property Name="ShippingType" Nullable="true" Type="Edm.String"/>
             ......
               <Property Name="DueQuarter" Nullable="true" Type="Edm.String"/>
               <Property Name="DueMonth" Nullable="true" Type="Edm.String"/>
               <Property Name="GrossProfitLC" Nullable="true" Type="Edm.Double"/>
               <Property Name="LineTotalAmountLC" Nullable="true" Type="Edm.Double"/>
             <Property Name="id__" Nullable="false" Type="Edm.Int32"/>
            </EntityType>


          <EntitySet EntityType="SalesOrderDetailQuery"
Name="SalesOrderDetailQuery"/>
```

The entity type and entity set of view *SalesOrderDetailQuery* are both in the name of *SalesOrderDetailQuery*. No other metadata are needed for this view. However, not all views are as simple as that. Some views have placeholders, for example, *sap.sbodemous.fin.fi/BalanceSheetQuery*, such as below:

```
SELECT * FROM
"_SYS_BIC"."sap.sbodemous.fin.fi/BalanceSheetQuery"('PLACEHOLDER'=('$$P_AddVoucher$$',
'N'),'PLACEHOLDER'=('$$P_FinancialPeriod$$','2017'))"
```

To expose this sort of view, only one entity type and one entity set are not enough to express it. The corresponding placeholders must be exposed in an appropriate way as well. Another characteristic of this view is that it cannot be executed directly in the SAP HANA studio. Only with the placeholder parameters can this view be accessed.

To cope with this situation, it is sensible to separate this view into two entity types, expose them respectively and then associate them with `navigation`.

```
    <EntityType Name="BalanceSheetQuery">
        <Key>
            <PropertyRef Name="id__"/>
        </Key>
        <Property MaxLength="15" Name="AccountCode" Nullable="true"
Type="Edm.String"/>
        <Property MaxLength="20" Name="FinancialPeriodCode" Nullable="true"
Type="Edm.String"/>
        <Property Name="FiscalYear" Nullable="true" Type="Edm.Int16"/>
```

```
        <Property MaxLength="100" Name="AccountName" Nullable="true"
Type="Edm.String"/>
        <Property MaxLength="200" Name="SegmentationAccountCode" Nullable="true"
Type="Edm.String"/>
        <Property Name="FiscalYearOpeningBalanceLC" Nullable="true"
Type="Edm.Double"/>
        <Property Name="FiscalYearOpeningBalanceSC" Nullable="true"
Type="Edm.Double"/>
        <Property Name="FinancialPeriodClosingBalanceLC" Nullable="true"
Type="Edm.Double"/>
        <Property Name="FinancialPeriodClosingBalanceSC" Nullable="true"
Type="Edm.Double"/>
        <Property Name="id__" Nullable="false" Type="Edm.Int32"/>
    </EntityType>

    <EntitySet EntityType="BalanceSheetQuery" Name="BalanceSheetQuery"/>

    <EntityType Name="BalanceSheetQueryParameter">
        <Key>
            <PropertyRef Name="P_FinancialPeriod"/>
            <PropertyRef Name="P_AddVoucher"/>
        </Key>
        <Property MaxLength="20" Name="P_FinancialPeriod" Nullable="false"
Type="Edm.String"/>
        <Property DefaultValue="N" MaxLength="1" Name="P_AddVoucher" Nullable="false"
Type="Edm.String"/>
        <NavigationProperty Name="BalanceSheetQuery"
Partner="BalanceSheetQueryParameters" Type="Collection(SAPB1.BalanceSheetQuery)"/>
    </EntityType>

    <EntitySet EntityType="SAPB1.BalanceSheetQueryParameter"
Name="BalanceSheetQueryParameters">
        <NavigationPropertyBinding Path="BalanceSheetQuery"
Target="BalanceSheetQuery"/>
    </EntitySet>
```

In this way, *BalanceSheetQuery* can be navigated from *BalanceSheetQueryParameters* with placeholders in the following way.

```
GET
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
/BalanceSheetQuery
```

> **Note**
>
> `NavigationPropertyBinding`, `Path` and `Target` are three attributes introduced in OData version 4 to describe the navigation properties of an entity set.

Directly accessing *BalanceSheetQuery* and *BalanceSheetQueryParameters* without keys would result in error.

3.8.6 Semantic Layer View Authorization

For SAP Business One forms, only authorized users have the privilege to access the corresponding views.

By default, a normal user has no permission to access views. Attempting to access would result in failure.

For example, log in to Service Layer with a normal user (e.g. user1) and then send a request to retrieve *BalanceSheetQuery*.

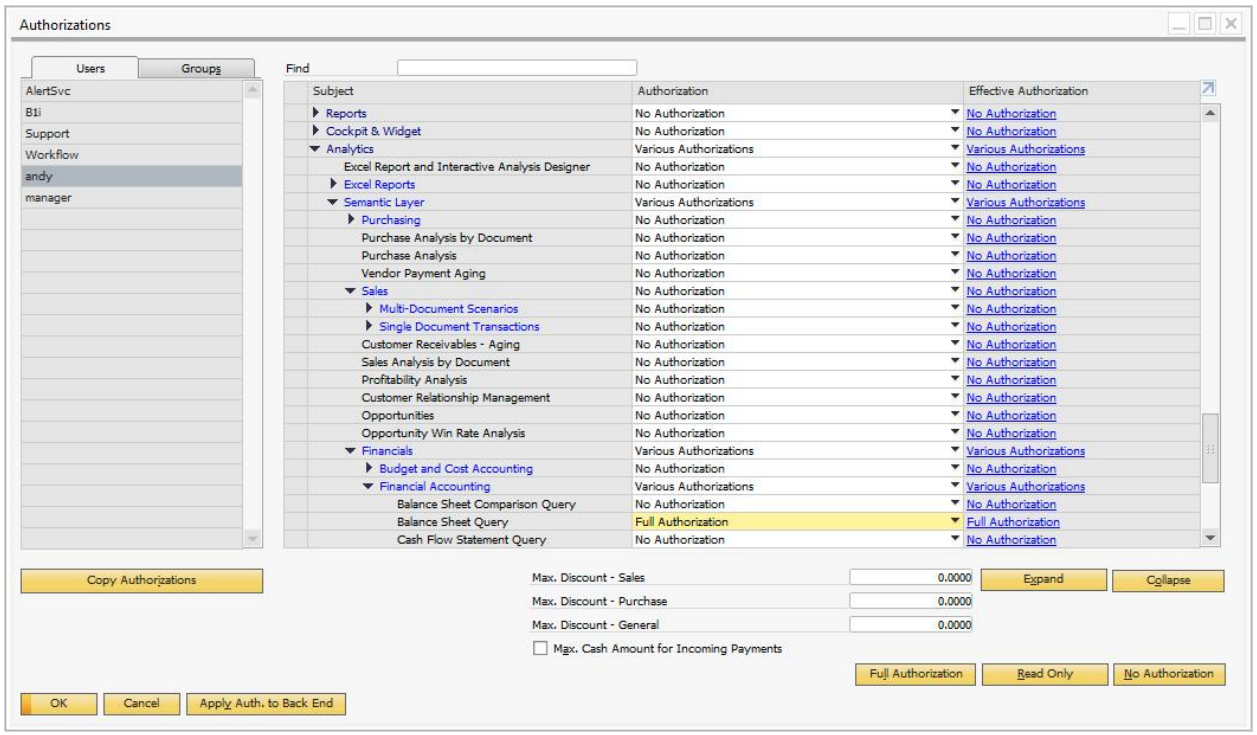
```
GET
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
/BalanceSheetQuery
```

Service returns:

```
HTTP/1.1 401 Unauthorized

{
    "error": {
        "code": -1,
        "message": {
            "lang": "en-us",
            "value": "No permission to access this view 'BalanceSheetQuery' for the
current user 'user1'"
        }
    }
}
```

To grant the view permission to a normal user, log on to the SAP Business One client with the superuser and then open the *General Authorizations* window from *System Initialization* -> *Authorizations*.



Note

Superusers have permission to access all exposed views.

The updated authorization for the normal user would not take effect immediately. To get the latest data, log off and log on to the service again or simply restart the service.

3.8.7 Semantic Layer View Query

Semantic Layer service enables you to perform the basic OData queries on exposed views, which makes it possible for it to be used in some flexible scenarios.

Assume all queries are performd on schema `SBODEMOUS`.

3.8.7.1 Getting All Records from View

```
GET /b1s/v1/sml.svc/AveragePurchasingPriceQuery
```

is equivalent to

```
SELECT * , row_number() over() as "id__" FROM
"_SYS_BIC"."sap.SBODEMOUS.ap.case/AveragePurchasingPriceQuery" T0
```

On success, the response is as follows:

```
HTTP/1.1 200 OK
```

```
{
    "@odata.context":
"https://databaseserver:50000/b1s/v1/sml.svc/$metadata#AveragePurchasingPriceQuery",
    "value": [
        {
           ...
            "PurchaseAmountLC": 5000,
            "PurchaseQuantityInInventoryUoM": 100,
            "AverageUnitPriceLC": 50,
            "id__": 1
        },
        {
           ...
            "PurchaseAmountLC": 2000,
            "PurchaseQuantityInInventoryUoM": 10,
            "AverageUnitPriceLC": 200,
            "id__": 2
        },
       ...
    ]
}
```

3.8.7.2 Querying View with Query Options

Semantic Layer service allows you to retrieve data with query option combinations.

For example, send the following request to service:

```
GET
/b1s/v1/sml.svc/AveragePurchasingPriceQuery?$select=PostingYear,BusinessPartnerCode&$s
kip=1&$filter=PostingYear eq '2017' and startswith(BusinessPartnerCode,
'1')&$orderby=PostingYear
```

On success, the response is as follows:

```
HTTP/1.1 200 OK

{
    "@odata.context":
"https://databaseserver:50000/b1s/v1/sml.svc/$metadata#AveragePurchasingPriceQuery",
    "value": [
        {
            "PostingYear": "2017",
            "BusinessPartnerCode": "1071287676"
```

```
        },
        {
            "PostingYear": "2017",
            "BusinessPartnerCode": "1100270398"
        },
        {
            "PostingYear": "2017",
            "BusinessPartnerCode": "124052273"
        },
        {
            "PostingYear": "2017",
            "BusinessPartnerCode": "1785286082"
        }
    ]
}
```

3.8.7.3 Querying View by Key

As mentioned above, the virtual property id__ was created just to comply with OData spec. Despite the fact that it functions similarly to that of a key, it is only allowed to do some simple queries.

Get one line by entity key

```
GET /b1s/v1/sml.svc/AveragePurchasingPriceQuery(1)
```

is equivalent to

```
SELECT * , row_number() over() as "__RowNum__" FROM
"_SYS_BIC"."sap.SBODEMOUS.adm/Item" T0 LIMIT 1 OFFSET 1
```

On success, the response is as follows:

```
HTTP/1.1 200 OK

{
    "@odata.context":
"https://databaseserver:50000/b1s/v1/sml.svc/$metadata#AveragePurchasingPriceQuery/$en
tity",
    "DueDateSQL": "2017-05-16",
    "DocumentDate": "2017-05-16",
    "DocumentYear": "2017",
       ...
    "DocumentQuarter": "Q2",
    "DocumentMonth": "05",
    "Manager": "-NULL-",
```

```
    "PurchaseAmountLC": 5000,
    "PurchaseQuantityInInventoryUoM": 100,
    "AverageUnitPriceLC": 50,
    "id__": 1
}
```

Get properties by entity key

```
GET
/b1s/v1/sml.svc/AveragePurchasingPriceQuery(1)?$select=PostingYear,BusinessPartnerCode
```

On success, the response is as follows:

```
HTTP/1.1 200 OK

{
    "@odata.context":
"https//databaseserver:50000/b1s/v1/sml.svc/$metadata#AveragePurchasingPriceQuery/$ent
ity",
    "PostingYear": "2017",
    "BusinessPartnerCode": "d3fb9f1c-72a0-4"
}
```

Get parameter entity by entity key

An entity ending with Parameter is only allowed to be accessed by key. For example, send a request as below:

```
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
```

On success, the response is as follows:

```
HTTP/1.1 200 OK

{
    "@odata.context":
"https://databaseserver:50000/b1s/v1/sml.svc/$metadata#BalanceSheetQueryParameters/$en
tity",
    "P_AddVoucher": "N",
    "P_FinancialPeriod": "2017"
}
```

Retrieving it without specifying an entity key results in the following failure message:

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": -1,
        "message": {
            "lang": "en-us",
            "value": "View parameters is not allowed to directly access."
        }
```

```
    }
}
```

> **Note**
>
> There are some innate query limitations on the virtual property id__. Do not depend on it excessively to perform complicated queries.

3.8.7.4 Querying View with Placeholders

A view with placeholders is queried by navigating from its corresponding parameter entity.

For example, to query *BalanceSheetQuery*, you should navigate from *BalanceSheetQueryParameters* as follows:

```
GET
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
/BalanceSheetQuery?$select=FiscalYear, AccountCode
```

which is equivalent to

```
SELECT FiscalYear, AccountCode  FROM
"_SYS_BIC"."sap.SBODEMOUS.fin.fi/BalanceSheetQuery"('PLACEHOLDER'=('$$P_AddVoucher$$',
'N'),'PLACEHOLDER'=('$$P_FinancialPeriod$$','2016'))
```

On success, the response is as follows:

```
HTTP/1.1 200 OK


{
    "@odata.context":
"https://databaseserver:50000/b1s/v1/sml.svc/$metadata#BalanceSheetQuery",
    "value": [
        {
            "FiscalYear": 2017,
            "AccountCode": "_SYS00000000049"
        },
        {
            "FiscalYear": 2017,
            "AccountCode": "_SYS00000000009"
        },
       ...
        {
            "FiscalYear": 2017,
            "AccountCode": "_SYS00000000029"
        }
    ],
```

```
    "@odata.nextLink":
"BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')/BalanceSheetQu
ery?$select=FiscalYear,%20AccountCode&$skip=20"
}
```

In addition, query by key is also supported for placeholder views.

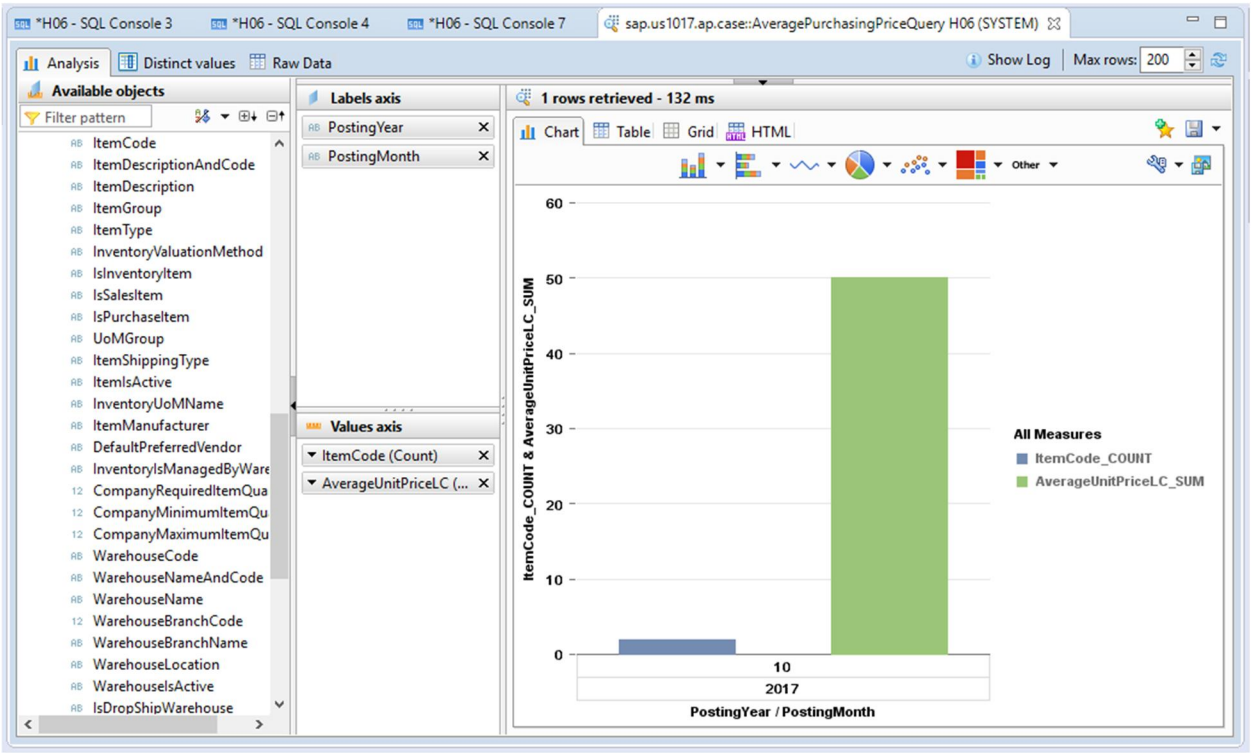
```
GET
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
/BalanceSheetQuery(1)
```

```
GET
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
/BalanceSheetQuery(1)?$select=FiscalYear, AccountCode
```

> **Note**
>
> `@odata.nextLink` is the paging annotation literal in OData version 4.

3.8.7.5 Querying View with Aggregation

In SAP HANA Studio, you can preview the query view data from multiple dimensions, such as displayed in the following screenshot:



The produced grid is as follows:

PostingYear	PostingMonth	ItemCode_COUNT	AverageUnitPriceLC_SUM
2017	10	2	50

The equivalent SQL for this is:

```
SELECT "PostingYear", "PostingMonth", COUNT(*) AS "ItemCode_COUNT",
SUM("AverageUnitPriceLC") AS "AverageUnitPriceLC_SUM"
FROM "_SYS_BIC"."sap.us1017.ap.case/AveragePurchasingPriceQuery"
GROUP BY "PostingYear", "PostingMonth"
ORDER BY "PostingYear" ASC, "PostingMonth" ASC
```

To simulate this in Semantic Layer service, send a request as follows:

```
GET /b1s/v1/sml.svc/AveragePurchasingPriceQuery?$apply=groupby((PostingYear,
PostingMonth), aggregate($count as ItemCode_COUNT, AverageUnitPriceLC with sum as
AverageUnitPriceLC_SUM))&$orderby=PostingYear asc,PostingMonth asc
```

On success, the response is as follows:

```
HTTP/1.1 200 OK

{
    "@odata.context":
"$metadata#AveragePurchasingPriceQuery(PostingYear,PostingMonth,ItemCode_COUNT,Average
UnitPriceLC_SUM)",
    "value": [
        {
            "PostingYear": "2017",
            "PostingMonth": "10",
            "ItemCode_COUNT": 2,
            "AverageUnitPriceLC_SUM": 50
        }
    ]
}
```

From this perspective, for a simple aggregation scenario, Semantic Layer service is somewhat capable of producing the similar result as SAP HANA Studio does. However, this does not mean the service is completely functionally equivalent to SAP HANA Studio. The service aggregation abilities could be gradually enhanced in subsequent patches.

Other Examples

To query the records number of one view (for example, *AveragePurchasingPriceQuery*), simply append the `$count` keyword as follows:

```
GET /b1s/v1/sml.svc/AveragePurchasingPriceQuery/$count
```

On success, the response is as follows:

```
HTTP/1.1 200 OK

14
```

The `$count` can also be applied to views with placeholders, seen from the following examples:

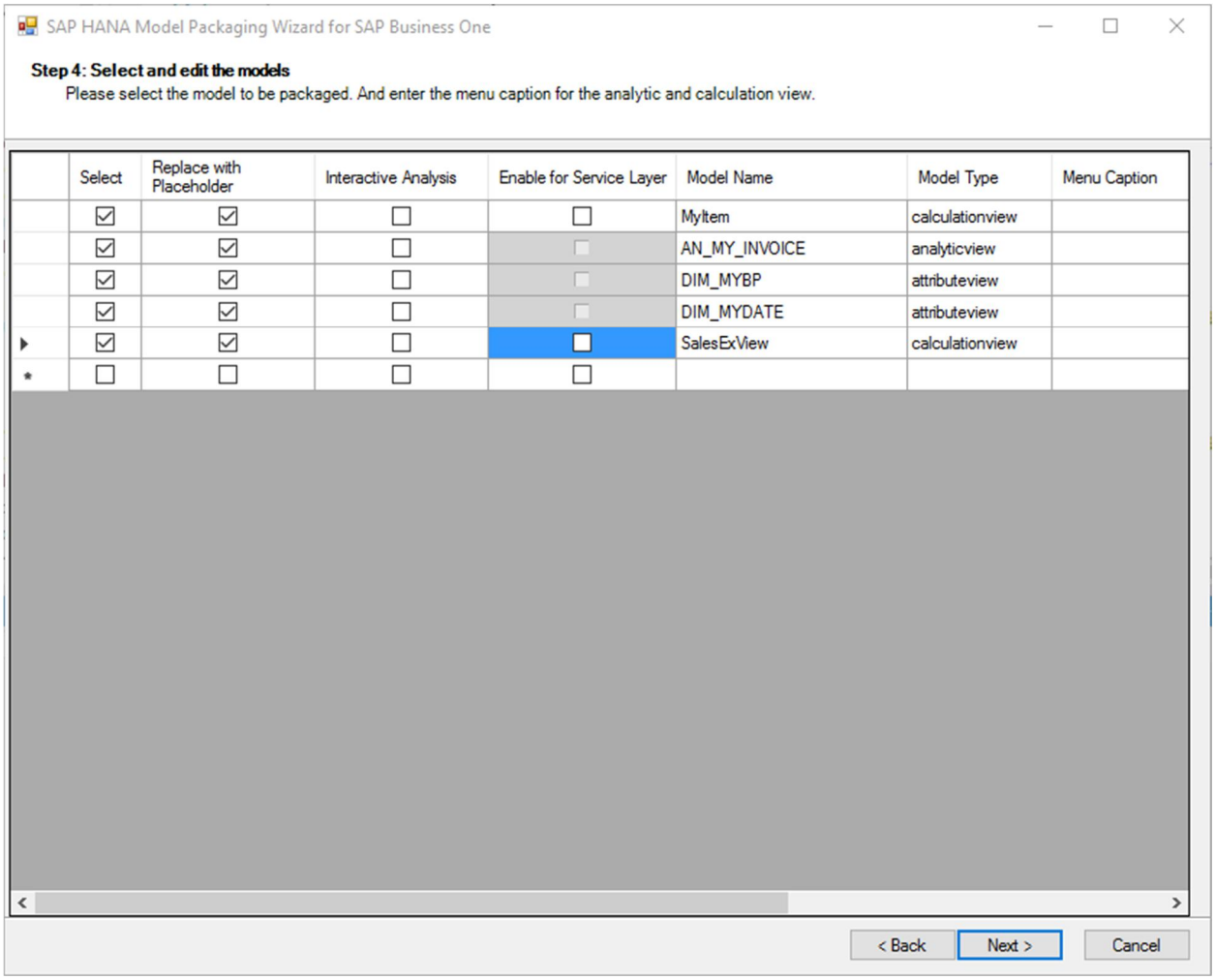
```
GET
/b1s/v1/sml.svc/ItemRecommendationQueryParameters(CurrentUserCode=1)/ItemRecommendatio
nQuery/$count
GET
/b1s/v1/sml.svc/BalanceSheetQueryParameters(P_FinancialPeriod='2017',P_AddVoucher='N')
/BalanceSheetQuery/$count
```

3.8.8 Customized Views Exposure

Besides the system built-in views, the customer designed views are also able to be exposed as OData service.

To achieve this, first use the latest SAP HANA Model Package Tool to generate a compressed model package after exporting the designed SAP HANA models from SAP HANA Studio. As for how to download and use this tool, see SAP Note 2008991 or this blog.

Compared to the old versions, in the *SAP HANA Model Packaging Wizard* for SAP Business One, a column named as `Enable for Service Layer` is added with checkbox type as below:

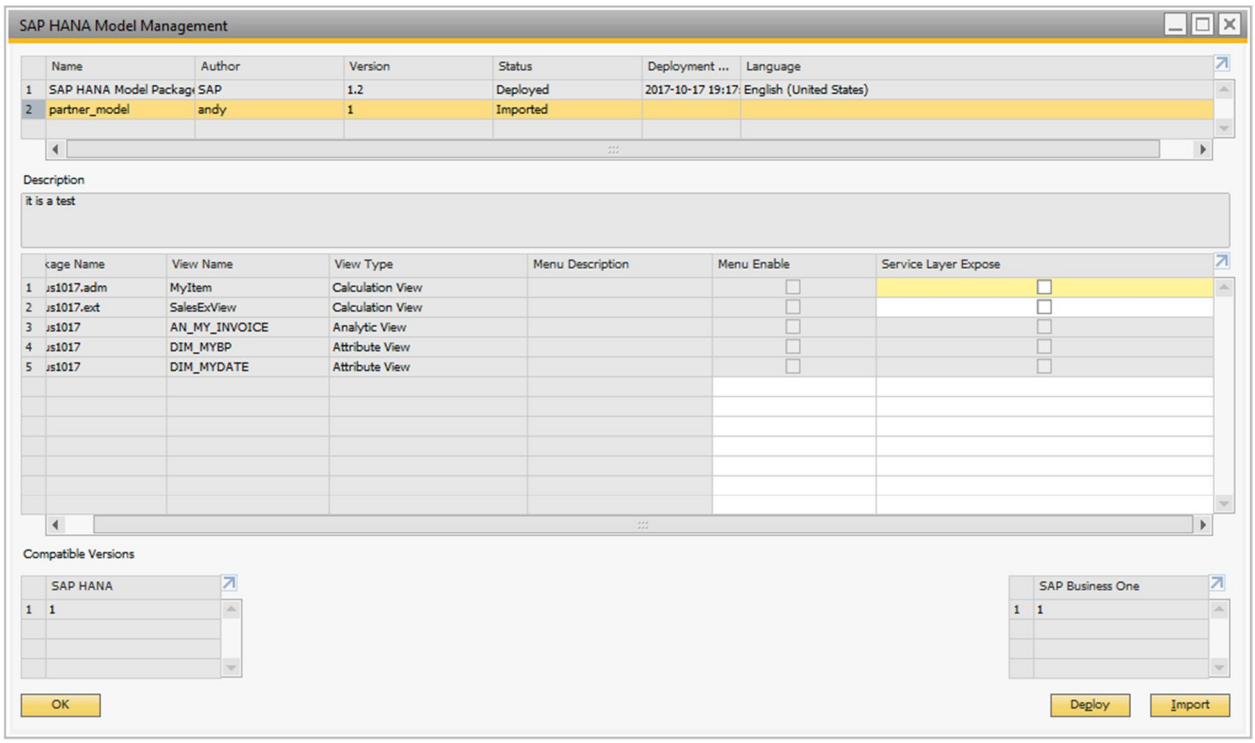


For this newly introduced column, the behaviors are described as follows:

- It is disabled for all views, except the views with calculation type.
- The default value is unchecked.
- If it is checked, a new parameter `SLEnable="Y"` is added to SAP HANA Model information file `Info.XML`.
- The new version packaging tool is compatible with the old version exported views.

  ```
  <Model name="xxxx" type="CalculationView" menu="mymenu" IAEnable="N" SLEnable="Y" SLExpose="Y" />
  ```
- If `SLEnable = "N"` or no `SLEnable` tag in `Info.XML`, the corresponding views should be disabled in the *SAP HANA Model Management* window.

Once the model package is ready, open the *SAP HANA Model Management* window in the SAP Business One client, import the package and click the *Deploy* button to start the deployment process as follows:



> **Note**
>
> - Customized views are not allowed to have the same name as the system built-in view, even with a different package path. Otherwise, the service will respond with an error.
> - Likewise, the authorization mechanism can also be applied to customized views.
> - One significant functionality of exposing customized views is to provide an alternative to the `RecordSet` in DI API, which is not allowed to expose in Service Layer for security and compatibility considerations.

3.8.9 Customized Views Query

Likewise, all queries supported on system built-in views can be performed on customized views as well. See the following examples:

Get all records from view

```
GET https://databaseserver:50000/b1s/v1/sml.svc/MyItem
```

Query one record from view

```
GET GET https://databaseserver:50000/b1s/v1/sml.svc/MyItem(2)
```

Get data with projection, filter and orderby

```
GET https://databaseserver:50000/b1s/v1/sml.svc/MyItem?$select=ItemGroup,
ItemCode&$filter=ItemCode ne 'FA10004'&$orderby=ItemCode desc,ItemGroup
```

Get data with aggregation

```
GET https://databaseserver:50000/b1s/v1/sml.svc/MyItem?$apply=aggregate(ItemCode with
countdistinct as CountDistinctItemCode)
```

```
GET https://databaseserver:50000/b1s/v1/sml.svc/MyItem?$apply=filter(IsPurchaseItem eq
'N')/groupby((ItemGroup), aggregate(ItemCode with max as MaxItemCode))
```

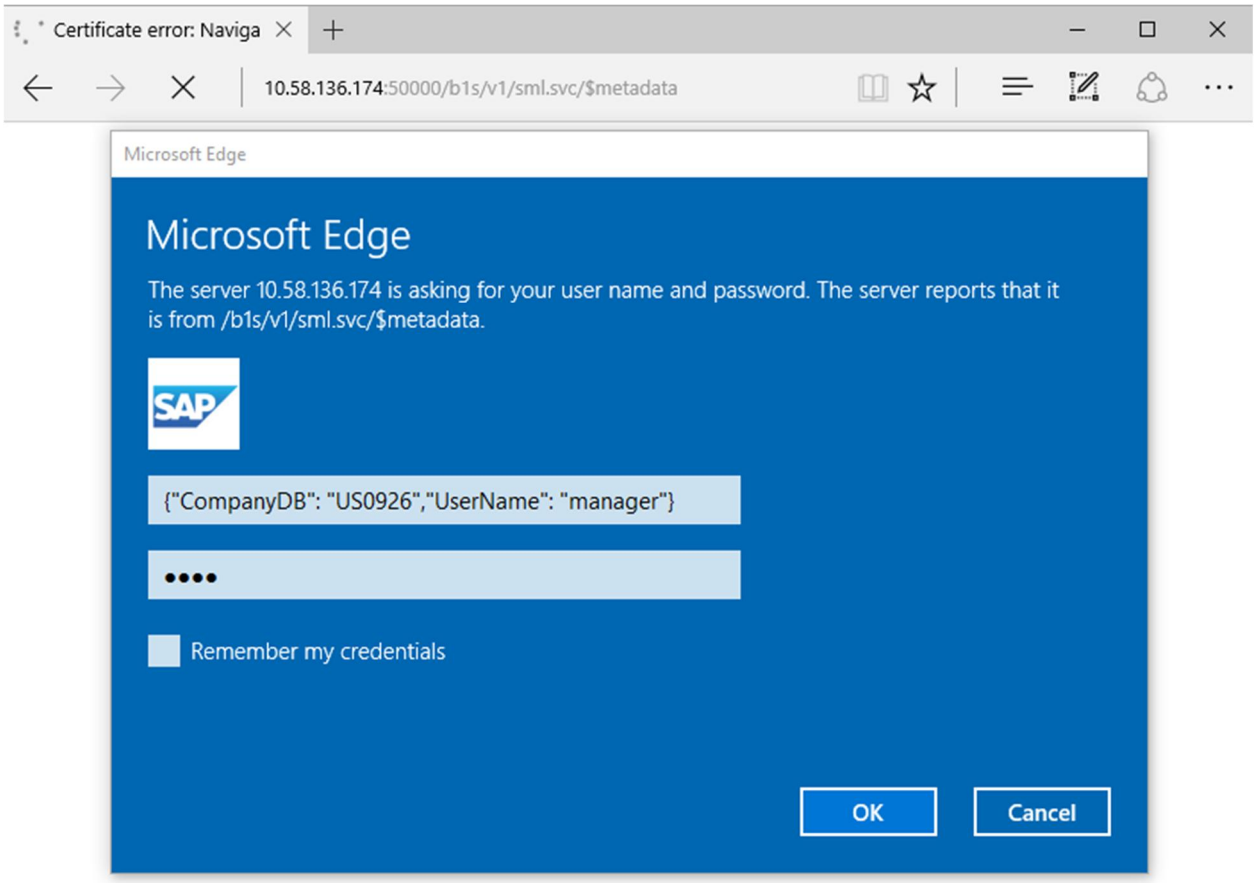
3.8.10 Semantic Layer Basic Authentication

Semantic Layer allows you to access views through basic authentication in browsers. For more information, see https://www.httpwatch.com/httpgallery/authentication/.

However, basic authentication only allows you to input user name and password. There is not a third input box for company database.

To address this issue, the solution combines the SAP Business One user name and company database together in a JSON format as the user name for basic authentication.

For example, for the browser Microsoft Edge, log on as follows:



Note

Basic authentication is another login mechanism just for Semantic Layer service. The authenticated session is not allowed to be reused to access the Service Layer resources (for example, BusinessPartners, Orders).

3.9 SQL View Exposure

As of SAP Business One 10.0 PL02, Service Layer on Microsoft SQL Server is able to automatically discover and expose the regular customized SQL views in OData V3/V4 protocol.

3.9.1 Create View

Open the SQL Server Management Studio, create a view (e.g. `B1_ItemPriceB1SLQuery`) in the company database (e.g. `SBODEMOUS`) in the following way:

```
USE [SBODEMOUS]
GO
```

```
/****** Object: View [dbo].[B1_ItemPriceB1SLQuery] Script Date:
12/13/2019 13:16:56 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[B1_ItemPriceB1SLQuery] AS
SELECT T0.[ItemCode], T0.[PriceList], T0.[UomEntry], T0.[Price], T0.[Currency],
T0.[PriceType]
FROM [dbo].[ITM1] T0
UNION ALL
SELECT T0.[ItemCode], T0.[PriceList], T0.[UomEntry], T0.[Price], T0.[Currency],
T0.[PriceType]
FROM [dbo].[ITM9] T0
GO
```

> **Note**
>
> To guarantee the view is recognized by Service Layer, it must fulfill the following requirements:
>
> - Make sure the view is under schema dbo.
> - Make sure the view name ends with B1SLQuery.

3.9.2 Expose View

By default, customized views are not exposed in Service Layer. To expose customized views, Service Layer brings a new entity SQLViews to help accomplish this.

For example, post the below request without any payload to expose B1_ItemPriceB1SLQuery:

```
POST /b1s/v1/SQLViews('B1_ItemPriceB1SLQuery')/Expose
```

On success, the service returns no content with response code 204.

To cancel the exposure, try the Unexpose command of SQLViews:

```
POST /b1s/v1/SQLViews('B1_ItemPriceB1SLQuery')/Unexpose
```

In a practical environment, exposing views one by one might be time consuming. To manipulate all views at once, use the asterisk * to represent all:

```
POST /b1s/v1/SQLViews('*')/Expose
POST /b1s/v1/SQLViews('*')/Unexpose
```

Since it is exposed as an entity, you are allowed to query the view's basic properties in the following way:

```
GET /b1s/v1/SQLViews('B1_ItemPriceB1SLQuery')
```

On success, the response is like:

```
HTTP/1.1 200 OK
{
"odata.metadata" :
"https://server:50000/b1s/v1/$metadata#SQLViews/@Element",
"Name" : "B1_ItemPriceB1SLQuery",
"DBType" : "MSSQL",
"SchemaName" : "dbo",
"CreateDate" : "2020-02-14"
}
```

And here is the relevant metadata:

```
<EntityType Name="SQLView">
<Key>
<PropertyRef Name="Name"/>
</Key>
<Property Name="Name" Nullable="false" Type="Edm.String"/>
<Property Name="DBType" Type="Edm.String"/>
<Property Name="SchemaName" Type="Edm.String"/>
<Property Name="CreateDate" Type="Edm.DateTime"/>
</EntityType>

<EntitySet EntityType="SAPB1.SQLView" Name="SQLViews"/>

<FunctionImport IsBindable="true" Name="Expose" m:HttpMethod="POST">
<Parameter Name="SQLViewParams" Type="SAPB1.SQLView"/>
</FunctionImport>

<FunctionImport IsBindable="true" Name="Unexpose" m:HttpMethod="POST">
<Parameter Name="SQLViewParams" Type="SAPB1.SQLView"/>
</FunctionImport>
```

3.9.3 View Service Endpoint

As in the Semantic Layer service, a unique endpoint is used for the view service, as below:

```
GET /b1s/v1/view.svc
```

To retrieve the metadata, append the `$metadata` to the endpoint:

```
GET /b1s/v1/view.svc/$metadata
```

On success, the service returns:

```
<?xml version="1.0" encoding="UTF-8"?>
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
<edmx:DataServices m:DataServiceVersion="3.0"
m:MaxDataServiceVersion="3.0"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
<Schema Namespace="SAPB1" xmlns="http://schemas.microsoft.com/ado/2009/11/edm">
<EntityType Name="B1_ItemPriceB1SLQuery">
<Key>
<PropertyRef Name="id__"/>
</Key>
<Property MaxLength="100" Name="ItemCode" Nullable="false" Type="Edm.String"/>
<Property Name="PriceList" Nullable="false" Type="Edm.Int16"/>
<Property Name="UomEntry" Nullable="true" Type="Edm.Int32"/>
<Property Name="Price" Nullable="true" Precision="19" Scale="6" Type="Edm.Decimal"/>
<Property MaxLength="6" Name="Currency" Nullable="true" Type="Edm.String"/>
<Property MaxLength="1" Name="PriceType" Nullable="true" Type="Edm.String"/>
<Property Name="id__" Nullable="false" Type="Edm.Int32"/>
</EntityType>
<EntityContainer Name="B1SView">
<EntitySet EntityType="SAPB1.B1_ItemPriceB1SLQuery" Name="B1_ItemPriceB1SLQuery"/>
</EntityContainer>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

The view service supports OData V4 as well. Here are the V4 URLs for the the endpoint and metadata, respectively.

```
GET /b1s/v2/view.svc
GET /b1s/v2/view.svc/$metadata
```

> **Note**
>
> - Please refer to OData-CSDL (Common Schema Definition Language) for more information on the metadata format.
> - All views are exposed as entities, as OData only allows to perform queries on entities. Due to the OData specification, each entity must at least have a primary key. However, this is contradictory to the fact that views do not have keys from the database perspective. To address this issue in a generic way, a virtual property id__ is defined as the entity key for the typical views, as seen from the last property of entity type in the metadata.

3.9.4 Query View

View service enables you to perform the basic OData queries on exposed views, which is useful in some flexible scenarios.

> **Note**
>
> The following examples are all using OData V3 to demonstrate how to do the view query. OData V4 can be applied as well.

3.9.4.1 Query View by Key

As mentioned above, the virtual property id__ is created just to comply with OData specification. Despite the fact that it functions similarly to a key, it is only allowed to perform some simple queries.

[Get one line by entity key]

The request below

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery(1)
```

is equivalent to

```
SELECT T0.[ItemCode] , T0.[PriceList] , T0.[UomEntry] , T0.[Price] , T0.
[Currency] , T0.[PriceType] , row_number() OVER (order by getdate()) as
[id__] FROM [dbo].[B1_ItemPriceB1SLQuery] T0 ORDER BY CURRENT_TIMESTAMP
OFFSET 0 ROWS FETCH NEXT 1 ROWS ONLY FOR BROWSE
```

On success, the response is like:

```
HTTP/1.1 200 OK
{
"odata.metadata":
"https://server:50000/b1s/v1/view.svc/$metadata#B1_ItemPriceB1SLQuery/@Element",
"ItemCode": "i001",
"PriceList": 1,
"UomEntry": -1,
"Price": 0,
"Currency": null,
"PriceType": "M",
"id__": 1
}
```

[Get properties by entity key]

Request as below:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery(1)?$select=ItemCode,PriceList
```

On success, the response is like:

```
HTTP/1.1 200 OK
```

```
{
"odata.metadata":
"https://server:50000/b1s/v1/view.svc/$metadata#B1_ItemPriceB1SLQuery/@Element",
"ItemCode": "i001",
"PriceList": 1
}
```

> **Note**
>
> - There are some innate query limitations on the virtual property id__. Do not excessively depend on it to perform complicated queries.
> - Query by key is just to comply with the OData specification. It is not a typical user case in a production environment.

3.9.4.2 Retrieve Records with Paging

Considering there might be hundreds of thousands of records one view would return in one roundtrip, the paging mechanism is enabled by default to avoid any potential resource racing issue on the service side.

For example, such a request as below

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery
```

results in the following response:

```
HTTP/1.1 200 OK
{
"odata.metadata":
"https://server:50000/b1s/v1/view.svc/$metadata#B1_ItemPriceB1SLQuery",
"value": [
{
"ItemCode": "i001",
"PriceList": 1,
"UomEntry": -1,
"Price": 0,
"Currency": null,
"PriceType": "M",
"id__": 1
},
...
{
"ItemCode": "i001",
"PriceList": 2,
"UomEntry": -1,
```

```
"Price": 0,
"Currency": null,
"PriceType": "M",
"id__": 20
}
],
"odata.nextLink": "B1_ItemPriceB1SLQuery?$skip=20"
}
```

The default page size is 20. To change it (for example, to 40) from the client side, append such a Prefer entry in the request header:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery
Prefer:odata.maxpagesize=40
```

3.9.4.3 Query Records with Query Options

View service allows you to retrieve data with query option combinations.

For example, send the below request to service:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery?
$select=ItemCode,PriceType&$filter=PriceType eq 'M'
```

On success, the response is like:

```
HTTP/1.1 200 OK
{
"odata.metadata":
"https://server:50000/b1s/v1/view.svc/$metadata#B1_ItemPriceB1SLQuery",
"value": [
{
"ItemCode": "i001",
"PriceType": "M"
},
{
"ItemCode": "i001",
"PriceType": "M"
},
{
"ItemCode": "i001",
"PriceType": "M"
},
{
```

```
"ItemCode": "i001",
"PriceType": "M"
},
...
{
"ItemCode": "i002",
"PriceType": "M"
}
]
}
```

3.9.4.4 Query View with Aggregation

To get the total record number of B1_ItemPriceB1SLQuery, send such a request:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery/$count
```

On success, the response is like:

```
HTTP/1.1 200 OK
Content-Type: text/plain;charset=utf-8;
40
```

To get the maximum ItemCode of the B1_ItemPriceB1SLQuery, send a request like:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery?$apply=aggregate(ItemCode with max as
MaxItemCode)
```

On success, the response is like:

```
HTTP/1.1 200 OK
{
"odata.metadata": "$metadata#B1_ItemPriceB1SLQuery(MaxItemCode)",
"value": [
{
"MaxItemCode": "i002"
}
]
}
```

The equivalent SQL on Microsoft SQL Server is:

```
SELECT MAX(T0.[ItemCode]) AS 'MaxItemCode' FROM [dbo].[B1_ItemPriceB1SLQuery] T0
```

To perform a complex aggregation with group by, send a request like:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery?$apply=groupby((ItemCode, PriceType),
aggregate($count as ItemCode_COUNT, Price with sum as SumPrice))&$orderby=ItemCode
asc,PriceType asc
```

On success, the response is like:

```
HTTP/1.1 200 OK
{
"odata.metadata":
"$metadata#B1_ItemPriceB1SLQuery(ItemCode,PriceType,ItemCode_COUNT,SumPrice)",
"value": [
{
"ItemCode": "i001",
"PriceType": "M",
"ItemCode_COUNT": 10,
"SumPrice": 0
},
{
"ItemCode": "i002",
"PriceType": "M",
"ItemCode_COUNT": 10,
"SumPrice": 0
}
]
}
```

The equivalent SQL on Microsoft SQL Server is:

```
SELECT T0.[ItemCode], T0.[PriceType], COUNT(*) AS 'ItemCode_COUNT', SUM(T0.[Price]) AS
'SumPrice' FROM [dbo].[B1_ItemPriceB1SLQuery] T0 GROUP BY T0.[ItemCode],
T0.[PriceType] ORDER BY T0.[ItemCode],T0.[PriceType] OFFSET 0 ROWS FETCH NEXT 20 ROWS
ONLY FOR BROWSE
```

3.9.5 Authorize View

As in SAP Business One, only authorized users have the privilege to access the corresponding views.

By default, a normal user has no permission to access views; attempting to do so would end in failure. For example, log in to Service Layer with a normal user and then send a request to retrieve B1_ItemPriceB1SLQuery:

```
GET /b1s/v1/view.svc/B1_ItemPriceB1SLQuery
```

As expected, the service returns:

```
HTTP/1.1 403 Forbidden
{
"error" : {
"code" : 804,
"message" : {
"lang" : "en-us",
"value" : "No permission to access this view
'B1_ItemPriceB1SLQuery' for the current user."
```

```
}
}
}
```

To grant the view permission to a normal user, perform the following:

1. Log on to SAP Business One client with a superuser.
2. Open the *Authorizations* window (*System Initialization* ->-*Authorizations*).
3. Change *No Authorization* to *Full Authorization*.

   Note

   - Superusers have full permission to access all exposed views.
   - The updated authorization for the normal user might not take effect immediately. To get the latest data, wait for a while (for example, one minute) to allow the internal permission cache mechanism to get refreshed.
   - The view's authorization is reset to *No Authorization* if the view is updated to unexposed status.

3.10 Batch Operations

Service Layer supports executing multiple operations sent in a single HTTP request through the use of batching. A batch request must be represented as a Multipart MIME (Multipurpose Internet Mail Extensions) v1.0 message.

3.10.1 Batch Request Method and URI

Always use the HTTP `POST` method to send a batch request. A batch request is submitted as a single HTTP `POST` request to the batch endpoint of a service, located at the URI `$batch` relative to the service root.

```
POST https://databaseserver:50000/b1s/v1/$batch
```

3.10.2 Batch Request Headers

The batch request must contain a `Content-Type` header that specifies a content type of multipart/mixed and a boundary specification as:

```
Content-Type: multipart/mixed;boundary=<Batch Boundary>
```

The boundary specification is used in the *Batch Request Body* section.

3.10.3 Batch Request Body

The body of a batch request is composed of a series of individual requests and change sets, each represented as a distinct MIME part, and separated by the boundary defined in the `Content-Type` header.

```
--<Batch Boundary>
<subrequest-1>
--<Batch Boundary>
<subrequest-2>
--<Batch Boundary>
Content-Type: multipart/mixed;boundary=<Changeset Boundary>

--<Changeset boundary>
<subchangeset-request-1>
--<Changeset boundary>
<subchangeset-request-2>
--<Changeset boundary>--
--<Batch Boundary>--
```

The service processes the requests within a batch request sequentially.

Each sub request must include a `Content-Type` header with value `application/http` and a `Content-Transfer-Encoding` header with value `binary`.

```
Content-Type:application/http
Content-Transfer-Encoding:binary

<sub request body>
```

The sub request body includes the real request content.

Example

```
POST /b1s/v1/Items

<Json format Items Content>
or
GET /b1s/v1/Item('i001')
```

Note that two empty lines are necessary after the `GET` request line. The first empty line is part of the `GET` request header, and the second one is the empty body of the `GET` request, followed by a `CRLF`.

3.10.4 Change Sets

A change set is an atomic unit of works. It means that any failed sub request in a change set will cause the whole change set to be rolled back. Change sets must not contain any `GET` requests or other change sets.

Sub change set requests basically have the same format as sub requests outside change sets, except for one additional feature: `Referencing Content ID`.

Referencing Content ID: New entities created by a POST request within a change set can be referenced by subsequent requests within the same change set by referring to the value of the `Content-ID` header prefixed with a $ character. When used in this way, `$<Content-ID>` acts as an alias for the URI that identifies the new entity.

Example

How to use change set with Content-ID

1. Create an order.
2. Use `$<Content-ID>` to modify the order you just created.

```
--<Batch Boundary>
Content-Type: multipart/mixed;boundary=<Changeset Boundary>

--<Changeset boundary>
Content-Type:application/http
Content-Transfer-Encoding:binary
Content-ID:1

POST /b1s/v1/Items

<Json format Items Content>
--<Changeset boundary>
Content-Type:application/http
Content-Transfer-Encoding:binary
Content-ID:2

PATCH /b1s/v1/$1

<Json format Item update content>
--<Changeset boundary>--
--<Batch Boundary>--
```

Note that `Content-ID` only exists in change set sub requests:

- For OData Version 3, it is not necessary to use `Content-ID` unless you need to use it for reference.
- For OData Version 4, this header is a mandatory field, whether you use it or not.

3.10.5 Batch Request Sample Codes

The sample codes in this section show a complete batch request that contains the following operations:

- A query request
- A change set that contains the following requests:
  - Insert entity (with `Content-ID` = 1)

- Update request (with `Content-ID` = 2)

Sample Codes

```
POST https://databaseserver:50000/b1s/v1/$batch
OData-Version: 4.0
Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding:binary

GET /b1s/v1/Items('i001')

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST /b1s/v1/Items('i002')
Content-Type: application/json

<Json format item(i002) body>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

PATCH /b1s/v1/$1
Content-Type: application/json

<Json format item(i002) update body>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

3.10.6 Batch Response

This section contains the batch responses after you execute the batch requests.

[Batch Request Format Invalid]

Service returns HTTP error code: `400 Bad Request` with error info in body if the request format is not valid.

Example

```
"error" : {
  "code" : -1000,
  "innererror" : {
        "context" : null,
        "trace" : null
  },
  "message" : {
        "lang" : "en-us",
        "value" : "Incomplete Batch Request Body!"
  }
}
```

[Batch Request Format Valid]

Service returns HTTP code: `202 Accept` (OData Version 3) or `200 OK` (OData Version 4) if the request format is valid, The response body that is returned to the client depends on the request execute result.

- If the batch request execution is successful, each sub request will have a corresponding sub response in the response body.

Example

```
--batchresponse_d878cedc-a0ad-4025-823e-5ee1aaffa288
Content-Type:application/http
Content-Transfer-Encoding:binary

HTTP/1.1 200 OK
Content-Type:application/json;odata=minimalmetadata;charset=utf-8
Content-Length:14729

<Json format Item(i001) Body>
--batchresponse_d878cedc-a0ad-4025-823e-5ee1aaffa288
Content-Type:multipart/mixed;boundary=changesetresponse_8bfb3c36-dbf7-46a0-bdfe-
670bbac86eb2

--changesetresponse_8bfb3c36-dbf7-46a0-bdfe-670bbac86eb2
Content-Type:application/http
Content-Transfer-Encoding:binary
```

```
Content-ID:1

HTTP/1.1 201 Created
Content-Type:application/json;odata=minimalmetadata;charset=utf-8
Content-Length:14641
Location:https://databaseserver:50000/b1s/v1/Items('i002')

<Json format Item(i002) Body>
--changesetresponse_8bfb3c36-dbf7-46a0-bdfe-670bbac86eb2
Content-Type:application/http
Content-Transfer-Encoding:binary
Content-ID:2

HTTP/1.1 204 No Content



--changesetresponse_8bfb3c36-dbf7-46a0-bdfe-670bbac86eb2--
--batchresponse_d878cedc-a0ad-4025-823e-5ee1aaffa288--
```

- If the batch request execution is not successful, the batch will stop executing once a sub request fails.

  Note that when there is a failure in the change set, only one response returns for this change set, no matter how many sub requests exist in this change set. For example, in the example below, the CREATE item operation fails because an item with the same item code already exists in the database.

  Example

```
--batchresponse_3aa0885d-245c-4164-b9a4-9c27f7a2c4d1
Content-Type:application/http
Content-Transfer-Encoding:binary

HTTP/1.1 200 OK
Content-Type:application/json;odata=minimalmetadata;charset=utf-8
Content-Length:14729

<Json format Item(i001) Body>
--batchresponse_3aa0885d-245c-4164-b9a4-9c27f7a2c4d1
Content-Type:application/http
Content-Transfer-Encoding:binary

HTTP/1.1 400 Bad Request
Content-Type:application/json;odata=minimalmetadata;charset=utf-8
Content-Length:233
```

```
{
   "error" : {
      "code" : -10,
      "innererror" : {
         "context" : null,
         "trace" : null
      },
      "message" : {
         "lang" : "en-us",
         "value" : "Item code 'i002' already exists"
      }
   }
}
--batchresponse_3aa0885d-245c-4164-b9a4-9c27f7a2c4d1--
```

Example

Create an order and cancel it in one transaction

```
POST https://databaseserver:50000/b1s/v1/$batch
content-type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-
56071383e77b


--batch_36522ad7-fc75-4b56-8c71-56071383e77b
content-type: multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-
9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
content-type: application/http
content-transfer-encoding:binary
content-id: 1

post orders
host:host

{
  "cardcode": "c1",
  "docduedate": "2017-07-20",
  "documentlines": [
 {
   "itemcode": "i1",
```

```
    "quantity": "1",
    "taxcode": "t1",
    "unitprice": "30"
  }
    ]
}

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
content-type: application/http
content-transfer-encoding:binary

post $1/cancel
host: host

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

3.11 Retrieving Individual Properties

> **Note**
>
> This feature is available in SAP Business One 9.1 patch level 04 and later.

> **Note**
>
> Retrieving the properties of a complex type is not supported. For example, the following request is not possible:
>
> `GET /Orders(1)/TaxExtension/TaxId0`

Retrieving Property Values

To retrieve the values of individual properties, send HTTP requests as follows:

`GET /Orders(1)/DocEntry`

The service returns either of the following:

- If `DocEntry` `1` exists:

  ```
  HTTP/1.1 200 OK

  {
      "value": 1
  }
  ```

- If DocEntry 1 does not exist:

```
HTTP/1.1 200 OK

{
    "odata.null": true
}
```

For OData version 4, an additional "@" is added before "`odata.null`" in the response.

Retrieving Property Raw Values

To retrieve the raw values of individual properties, send HTTP requests as follows:

```
GET /Orders(1)/DocEntry/$value
```

The service returns:

```
HTTP/1.1 200 OK

1
```

For null values, the service returns a `404 Not Found` error, as below:

```
HTTP/1.1 404 Not Found

{
    "error": {
        "code": -2028,
        "innererror": {
            "context": null,
            "trace": null
        },
        "message": {
            "lang": "en-us",
            "value": "Resource not found for the property: DocEntry"
        }
    }
}
```

3.12Associations

> **Note**
>
> This feature is available in SAP Business One 9.1 patch level 05 and later.

Two entities may be associated (independently related) in one way or another. The association is optionally represented in the navigation property of each association end (one of the two associated entities).

For example, if an association and corresponding navigation properties have been defined for order and customer entities in the metadata, you can send the following HTTP request to get the customer associated with a particular order:

```
GET Orders(1)/BusinessPartner
```

If you already knew that the `CardCode` property of the order is "c1", the above request is equal to `GET BusinessPartners('c1')`.

You can continue to operate on this entity as on `GET BusinessPartners('c1')`. For example, to get the foreign name of the customer, the following two requests are also equal:

- `GET Orders(1)/BusinessPartner/ForeignName`
- `GET BusinessPartners('c1')/ForeignName`

3.12.1 Metadata Definitions of Associations and Navigation Properties

Associations and navigation properties are defined in the service metadata. Take orders and business partners, for example:

```
<!-- section 1 -->
<Association Name="FK_Documents_BusinessPartners">
  <End Type="SAPB1.BusinessPartner" Role="BusinessPartners" Multiplicity="0..1" />
  <End Type="SAPB1.Document" Role="Documents" Multiplicity="*" />
  <ReferentialConstraint>
    <Principal Role="BusinessPartners">
      <PropertyRef Name="CardCode" />
    </Principal>
    <Dependent Role="Documents">
      <PropertyRef Name="CardCode" />
    </Dependent>
  </ReferentialConstraint>
</Association>

<!-- section 2 -->
<EntityType Name="BusinessPartner">
  <Key>
    <PropertyRef Name="CardCode"/>
  </Key>
  <Property Name="CardCode" Nullable="false" Type="Edm.String"/>
  <Property Name="CardName" Type="Edm.String"/>
  <Property Name="CardType" Type="SAPB1.BoCardTypes"/>
  ...
```

```
  <NavigationProperty Name="Orders" Relationship="SAPB1.FK_Documents_BusinessPartners"
FromRole="BusinessPartners" ToRole="Orders" />
  <NavigationProperty Name="Invoices"
Relationship="SAPB1.FK_Documents_BusinessPartners" FromRole="BusinessPartners"
ToRole="Invoices" />
  ...
</EntityType>

<!-- section 3 -->
<EntityType Name="Document">
  <Key>
    <PropertyRef Name="DocEntry"/>
  </Key>
  <Property Name="DocEntry" Nullable="false" Type="Edm.Int32"/>
  <Property Name="DocNum" Type="Edm.Int32"/>
  <Property Name="DocType" Type="SAPB1.BoDocumentTypes"/>
  ...
  <NavigationProperty Name="BusinessPartner"
Relationship="SAPB1.FK_Documents_BusinessPartners" FromRole="Documents"
ToRole="BusinessPartners" />
</EntityType>

<!-- section 4 -->
<AssociationSet Association="SAPB1.FK_Documents_BusinessPartners"
Name="FK_Orders_BusinessPartners">
  <End EntitySet="Orders" Role="Documents"/>
  <End EntitySet="BusinessPartners" Role="BusinessPartners"/>
</AssociationSet>
<AssociationSet Association="SAPB1.FK_Documents_BusinessPartners"
Name="FK_Invoices_BusinessPartners">
  <End EntitySet="Invoices" Role="Documents"/>
  <End EntitySet="BusinessPartners" Role="BusinessPartners"/>
</AssociationSet>
```

The metadata defines the association between `BusinessPartners` and `Orders` as follows:

- Section 1 defines a "1:*" (1:n) association between `BusinessPartners` and `Documents`, joined on the condition `BusinessPartners.CardCode = Documents.CardCode`.
- Section 2 defines two navigation properties `Orders` and `Invoices` on entity type `BusinessPartner`.
- Section 3 defines a navigation property `BusinessPartner` on entity type `Document`.
- Section 4 defines two association sets with the same association `FK_Documents_BusinessPartners`. The first association set is `Orders` and `BusinessPartners` and the second is `Invoices` and `BusinessPartners`.

3.12.2 Retrieving navigation properties as entity

As long as navigation properties are defined on association ends, you can navigate back and forth between the association ends. The navigation is not necessarily bidirectional; it can be unidirectional.

According to the metadata (section 2 in Metadata Definitions of Associations and Navigation Properties), a navigation property `Orders` has been defined for entity type `BusinessPartner` (the type of entity set `BusinessPartners`). To get the orders associated with business partner "c1", send the following request:

```
GET BusinessPartners('c1')/Orders
```

This request is equal to the following request:

```
GET Orders?$filter=CardCode eq 'c1'
```

According to the metadata (section 3 in Metadata Definitions of Associations and Navigation Properties), entity type `Document` has a navigation property `BusinessPartners`. To get the customer associated with the order (`DocEntry`: `1`), send the following request:

```
GET Orders(1)/BusinessPartner
```

You can extend your request chain even further in the URL. For example, to get all orders of the customer who is associated with order 1, send the following request:

```
GET Orders(1)/BusinessPartner/Orders
```

3.12.3 Retrieving navigation properties via $expand

With OData query option $select and $expand, the navigation properties can be retrieved just as other properties. For example, to retrieve the customer as a property of an order, send the following request:

```
GET Orders(1)?$select=*,BusinessPartner&$expand=BusinessPartner
```

You can get the customer code property from an order and the foreign name property from the associated customer by sending the following request:

```
GET Orders(1)?$select=CardCode,BusinessPartner/ForeignName&$expand=BusinessPartner
```

$expand can be applied to collections as well. For instance, you can send the following request to retrieve the `BusinessPartner` properties of all orders:

```
GET Orders?$select=*,BusinessPartner&$expand=BusinessPartner
```

> **Note**
>
> The following two requests have the same effect:
>
> ```
> GET Orders(1)?$select=CardCode,BusinessPartner/ForeignName
> ```
>
> ```
> GET Orders(1)?$select=CardCode
> ```
>
> For the former, `BusinessPartner/ForeignName` is ignored as the navigation property `BusinessPartner` is not expanded.

> **Note**
>
> $expand working with collections may have performance issues. We recommend that you not send such requests frequently.

3.13 User-Defined Schemas

> **Note**
>
> This feature is available in SAP Business One 9.1 patch level 03 and later.

A typical sales order returned by Service Layer:

```
{
    "DocEntry": 71,
    "DocNum": 51,
    "DocType": "dDocument_Items",
    "HandWritten": "tNO",
    "Printed": "psNo",
    ...
    "DocumentLines": [
        {
            "LineNum": 0,
            "ItemCode": "i1",
            "ItemDescription": "item 1",
            "Quantity": 10,
            "ShipDate": "2014-04-01",
            "Price": 100,
            ...
        },
        {
            "LineNum": 1,
            "ItemCode": "i2",
            "ItemDescription": "item 2",
            "Quantity": 8,
            "ShipDate": "2014-04-01",
            "Price": 120,
            ...
        }
    ],
    ...
}
```

If the existing data structure does not satisfy your needs or you want to restrict the field amount, you can create your own schemas. Note that user-defined schemas are based on entities.

Prerequisites

Before working with a user-defined schema, ensure the following:

- You have created a schema file under the `<Installation Directory>/ServiceLayer/conf` folder.

  > **Note**
  >
  > If you want to send requests directly through a load balancer member which is installed on a different machine from the load balancer, you must ensure a copy of the schema file exists also on the member machine.

- You have defined the schema in the JSON format, according to your needs.

  > **Note**
  >
  > Any change to the schema file takes effect immediately after you save the file. You do not have to restart the Service Layer service.

- If you want to use the default schema defined in the `b1s.conf` file instead of specifying the schema in requests, you have made the schema file name identical to the value of the schema configuration option. For more information, see Configuration Options for Service Layer.

3.13.1 Filter Fields

A company database usually contains many more fields than needed. You can define your own schemas with "trimmed" data structures.

> **Example**
>
> How to restrict data output to a limited number of fields
>
> In the `conf` folder, create a file named **`marketingDocument.schema`** and edit the file as below:

```
{
    "Document": [
        "DocEntry",
        "DocNum",
        "DocumentLines",
    ],
    "DocumentLine": [
        "LineNum",
        "ItemCode",
        "Quantity"
    ]
}
```

This schema restricts output fields as follows:

- For type (`EntityType` or `ComplexType` in the metadata) `Document` (including all marketing document entities): `DocEntry`, `DocNum`, and `DocumentLines`
- For type `DocumentLine`: `LineNum`, `ItemCode`, and `Quantity`

The service returns:

```
HTTP/1.1 200 OK
B1S-Schema: schema1.schema
...(other HTTP headers)...
{
    "DocEntry": 71,
    "DocNum": 51,
    "DocumentLines": [
        {
            "LineNum": 0,
            "ItemCode": "i1",
            "Quantity": 10
        },
        {
            "LineNum": 1,
            "ItemCode": "i2",
            "Quantity": 8
        }
    ]
}
```

Note

The schema file is often named `{xxx}.schema` but that is not mandatory. You can use any name, for example, `myschema`.

In SAP Business One 9.1 patch level 04 and later, a schema file named `demo.schema` is available after installation. You can directly use it as follows:

```
GET /Orders
B1S-Schema: demo.schema
```

3.14 User-Defined Fields (UDFs)

In SAP Business One 9.1 patch level 04 and earlier, user-defined fields (UDFs) are treated as dynamic properties of an OData entity. An entity that has a dynamic property is of "open type", that is, in the `EntityType` XML node in metadata, it has the attribute `OpenType=true`.

As of SAP Business One 9.1 patch level PL04, you can manage the metadata of UDFs and perform CRUD operations on UDFs as on regular entities.

As of SAP Business One 9.1 patch level PL05, UDFs appear in the entity definition in metadata.

Note

All UDFs in SAP Business One are prefixed with "U_".

3.14.1 Managing Metadata of UDFs

This feature is available in SAP Business One 9.1 patch level 04 and later.

You can perform CRUD operations on UDFs as on regular entities. However, you must be aware that DDL operations on tables can be expensive in the database if the tables are referenced by many objects (for example, procedures, functions, and views). It may take longer than expected to create or delete UDFs, especially in marketing documents.

Creating UDFs

Use the `POST` method to create a UDF. For example, to create a UDF named "u1" on table `OCRD` (business partner master data), send the following HTTP request:

```
POST /UserFieldsMD

{
    "Name": "u1",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "udf 1",
    "SubType": "st_None",
    "TableName": "OCRD"
}
```

The service returns:

```
HTTP/1.1 201 Created

{
    "Name": "u1",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "udf 1",
    "SubType": "st_None",
    "LinkedTable": null,
    "DefaultValue": null,
    "TableName": "OCRD",
    "FieldID": 0,
    "EditSize": 10,
    "Mandatory": "tNO",
    "LinkedUDO": null,
    "ValidValuesMD": []
}
```

For data consistency, when you create a UDF on a particular table, the UDF is automatically created on other related tables. In the example above, in addition to `OCRD`, the UDF "u1" is also created on table `ACRD` (the archive table for the Business Partner object). If you create a UDF on a sales order row table `RDR1`, the UDF is automatically created on all marketing document line tables (for example, purchase order rows -`POR1`, delivery rows - `DLN1`, invoice rows - `INV1`) as well as the archive table for document rows `ADO1`.

For more information, see Creating Entities.

Retrieving UDFs

To retrieve a UDF, send an HTTP request as the example below:

```
GET /UserFieldsMD(TableName='OCRD', FieldID=0)
```

For more information, see Retrieving Entities.

Querying UDFs

Standard OData query options are also supported for UDFs. For example, you've forgotten the table name and the UDF ID but still remember the UDF name; you can query the UDF as follows:

```
GET /UserFieldsMD?$filter=Name eq 'u1'
```

The service returns:

```
{
    "value": [
        {
            "Name": "u1",
            "TableName": "ACRD",
            "FieldID": 0,
            ...
            ...
        },
        {
            "Name": "u1",
            "TableName": "OCRD",
            "FieldID": 0,
            ...
        }
    ]
}
```

From the response above, you can see that in addition to table `OCRD`, UDF "u1" has also been created on table `ACRD`.

For more information, see Query Options.

Updating UDFs

To change the description and size of a UDF, send an HTTP request as the example below:

```
PATCH /UserFieldsMD(TableName='OCRD', FieldID=0)

{
    "EditSize": 20,
    "Description": "Internal Id",
}
```

> **Note**
>
> You cannot change such properties as the file type (`Type`).

For more information, see Updating Entities.

Deleting UDFs

To delete a UDF, send an HTTP request as the example below:

```
DELETE /UserFieldsMD(TableName='OCRD', FieldID=0)
```

3.14.2 CRUD Operations

As with regular entities, you can perform CRUD operations on UDFs, query UDFs, and so on.

> **Example**
>
> How to add Business Partners with a UDF "U_BPSpecRemarks"
>
> Send the HTTP request:
>
> ```
> POST /BusinessPartners
> {
>     "CardCode": "bpudf_004",
>     ...
>     "U_BPSpecRemarks": "First Business Partners with UDF remarks added by
> Chrome."
> }
> ```
>
> The service returns:
>
> ```
> HTTP/1.1 200 OK
> {
>     "CardCode": "bpudf_004",
>     ...
> ```

```
    "U_BPSpecRemarks": "First Business Partners with UDF remarks added by
Chrome.",
    ...
}
```

Example

How to query entities using UDFs

Send the HTTP request:

```
GET /BusinessPartners?$filter=startswith(U_BPSpecRemarks, 'First')
```

The service returns:

```
HTTP/1.1 200 OK
{
    "value": [
        {
            "CardCode": "bpudf_001",
            ...
            "U_BPSpecRemarks": "First Business Partners with UDF remarks.",
        },
        {
            "CardCode": "bpudf_003",
            ...
            "U_BPSpecRemarks": "First Business Partners with UDF remarks added
by Chrome.",
        },
        ...
    ]
}
```

3.15User-defined Tables (UDTs)

You can directly access user-defined tables (UDTs) of "no object" type in SAP Business One 9.1 patch level 05 and later. UDTs of "no object" type are treated as simple entities that only have one main table. UDTs of "no object" type cannot be used by UDOs (UDOs use UDTs of type "master data", "master data rows", "document" or "document rows").

In the following examples, we will add UDT "MYTBL" of type "no object", and then service layer will expose it as an entity named "U_MYTBL".

3.15.1 Managing Metadata of UDTs

You can manage UDT metadata via service layer in SAP Business One 9.1 patch level 04 and later.

Example

How to create UDT "MYTBL" as a "no object" table

Send the HTTP request:

```
POST /UserTablesMD
{
    "TableName": "MYTBL",
    "TableDescription": "My Table",
    "TableType": "bott_NoObject"
}
```

Note that:

- The table name uses capital letters. This name will be used when you get the metadata of this new table via `GET /UserTablesMD('MYTBL')`.
- The real table in the database is "@MYTBL". This name will be used when you add user-defined fields.

Example

How to add fields "F1" to table "MYTBL"

Send the HTTP request to add field "F1" with type "Alphanumeric":

```
POST /UserFieldsMD
{
    "Name": "F1",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "Customer name",
    "SubType": "st_None",
    "TableName": "@MYTBL"
}
```

3.15.2 CRUD Operations

As with regular entities, you can perform CRUD operations on UDTs.

Service layer maps UDTs to entities by adding the prefix "U_". For example, UDT "MYTBL" gets the entity name "U_MYTBL".

Example

How to create entities for UDT "MYTBL"

Send the HTTP request:

```
POST /U_MYTBL
{
 "Code":"C",
 "Name":"CName",
 "U_F1":"test data"
}
```

How to retrieve the records of UDT "MYTBL" (Code is the key field)

Send the HTTP request:

```
GET /U_MYTBL('C')
```

Or

```
GET /U_MYTBL(Code='C')
```

How to get a key list of all user orders

Send the HTTP request:

```
GET /U_MYTBL?$select=Code
```

How to get a record whose name equals 'CName'

Send the HTTP request:

```
GET /U_MYTBL?$filter=Name eq 'CName'
```

How to update the record value of "U_F1"

Send the HTTP request:

```
PATCH /U_MYTBL('C')
{
    "U_F1": "test data - updated"
}
```

How to delete the record

Send the HTTP request:

```
DELETE /U_MYTBL('C')
```

3.16 User-Defined Objects (UDOs)

Note

CRUD operations are possible for UDOs in SAP Business One 9.1 patch level 03 and later.

You can manage UDO metadata in Service Layer in SAP Business One 9.1 patch level 04 and later.

You can access UDTs via Service Layer directly in SAP Business One 9.1 patch level 05 and later.

In SAP Business One 9.1 patch level 05 and later, information from UDTs, UDOs and UDFs is included in OData metadata https://databaseserver:50000/b1s/v1/$metadata.

In SAP Business One 9.2 patch level 11 and later, UDO Cancel/Close function is supported.

Depending on your business needs, you can create your own objects for managing custom data and creating custom functionality. Each user-defined object must be registered with one main user-defined table and, optionally, with one or more child UDTs. Each UDT contains one or more user-defined fields (UDFs). The object type of a main UDT must be either Master Data or Document, while the object type of a child UDT must be either Master Data Rows or Document Rows.

3.16.1 Managing Metadata of UDOs

This feature is available in SAP Business One 9.1 patch level 04 and later.

Service Layer requires the same procedure to create a UDO as in the SAP Business One client application or via the DI API. The following procedure illustrates how to create a UDO "MyOrder" with the following definition:

Main Table / Child Table	UDT	UDF	Type
Main Table	**MyOrder**		Document
		CustomerName	Alphanumeric
		DocTotal	Units and Totals / Amount
Child Table	**MyOrderLines**		Document Rows
		ItemName	Alphanumeric
		Price	Units and Totals / Price
		Quantity	Units and Totals / Quantity

Procedure

1. To create the main table "MyOrder", send the following HTTP request:

```
POST /UserTablesMD
{
    "TableName": "MyOrder",
    "TableDescription": "My Orders",
    "TableType": "bott_Document"
}
```

The service returns:

```
HTTP/1.1 201 Created
{
    "TableName": "MYORDER",
    "TableDescription": "My Orders",
    "TableType": "bott_Document",
```

```
    "Archivable": "tNO",
    ...
}
```

> **Note**
>
> The name is the unique identifier for a UDT and undergoes the following automatic changes after its creation:
>
> - A prefix "@" is added to the name.
> - The name is converted to the upper case.
>
> For example, if you define a UDT name as "`MyOrder`", the actual UDT name in the database is "`@MYORDER`".
>
> To obtain the metadata of this table, send the HTTP request:
>
> `GET /UserTablesMD('MYORDER')`

2. Add UDFs to table "`MyOrder`".
   1. To create field "`CustomerName`", send the following HTTP request:

```
POST /UserFieldsMD
{
    "Name": "CustomerName",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "Customer Name",
    "SubType": "st_None",
    "TableName": "@MYORDER"
}
```

   2. To create field "`DocTotal`", send the following HTTP request:

```
POST /UserFieldsMD
{
    "Name": "DocTotal",
    "Type": "db_Float",
    "Description": "Total Amount",
    "SubType": "st_Sum",
    "TableName": "@MYORDER"
}
```

3. To create the child table "`MyOrderLines`", send the following HTTP request:

```
POST /UserTablesMD
{
    "TableName": "MyOrderLines",
    "TableDescription": "My Order Lines",
    "TableType": "bott_DocumentLines"
}
```

   As with the main table, the actual name of this table is "`@MYORDERLINES`".

4. Add UDFs to table "MyOrderLines".
    1. To create field "ItemName", send the following HTTP request:

```
POST /UserFieldsMD
{
    "Name": "ItemName",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "Item name",
    "SubType": "st_None",
    "TableName": "@MYORDERLINES"
}
```

    2. To create field "Price", send the following HTTP request:

```
POST /UserFieldsMD
{
    "Name": "Price",
    "Type": "db_Float",
    "Description": "Unit Price",
    "SubType": "st_Price",
    "TableName": "@MYORDERLINES"
}
```

    3. To create field "Quantity", send the following HTTP request:

```
POST /UserFieldsMD
{
    "Name": "Quantity",
    "Type": "db_Float",
    "Description": "Quantity",
    "SubType": "st_Quantity",
    "TableName": "@MYORDERLINES"
}
```

5. To register UDO "MyOrder", send the following HTTP request:

```
POST /UserObjectsMD
{
    "Code": "MyOrder",
    "Name": "My Orders",
    "TableName": "MyOrder",
    "ObjectType": "boud_Document",
    "UserObjectMD_ChildTables": [
        {
            "TableName": "MyOrderLines",
            "ObjectName": "MyOrderLines"
        }
```

```
        ]
    }
```

> **Note**
>
> The property name of a subobject collection is "<Subobject Code>Collection" and the UDT code is, by default, the same as the UDT name. Therefore, if you intend to use a UDT as a child table for a UDO and the UDT name (`TableName`) contains spaces, we recommend that you change the UDT code (`ObjectName`) during the registration. For example, if a UDT code is "`My Order Lines`", the corresponding property name would be "`My Order LineCollection`".

Adding User Keys

You can add user keys to user-defined tables. For example, to create a unique key on column `CustomerName`, send the following HTTP request:

```
POST /UserKeysMD
{
    "TableName": "@MYORDER",
    "KeyIndex": "1",
    "KeyName": "IX_0",
    "Unique": "tYES",
    "UserKeysMD_Elements": [
        {
            "ColumnAlias": "CustomerName"
        }
    ]
}
```

The service returns:

```
{
    "TableName": "@MYORDER",
    "KeyIndex": "0",
    "KeyName": "IX_0",
    "Unique": "tYES",
    "UserKeysMD_Elements": [
        {
            "SubKeyIndex": "0",
            "ColumnAlias": "CustomerName"
        },
    ]
}
```

To get the metadata of this key, send the following HTTP request:

```
GET /UserKeysMD(TableName='@MYORDER', KeyIndex=0)
```

3.16.2 Creating Entity for a UDO

As with regular entities, you can perform CRUD operations on UDOs, query UDOs, create user-defined schemas based on UDOs, and so on.

To create a UDO entity, send an HTTP request to add an order with 2 items, such as:

```
POST /MyOrder

{
    "U_CustomerName": "c1",
    "U_DocTotal": 620,
    "MyOrderLinesCollection": [
        {
            "U_ItemName": "item1",
            "U_Price": 100,
            "U_Quantity": 3
        },
        {
            "U_ItemName": "item2",
            "U_Price": 80,
            "U_Quantity": 4
        }
    ]
}
```

> **Note**
>
> The property name of the sub object collection is "{sub-object-code}Collection", for example, the sub object code is `MyOrderLines` (by default the same as the UDT name if you do not change it), so the property name is `MyOrderLinesCollection`.

On success, the server replies as the follows:

```
HTTP/1.1 201 Created

{
    "DocEntry": 10,
    "DocNum": 2,
    ...
    "U_CustomerName": "c1",
    "U_DocTotal": 620,
    "MyOrderLinesCollection": [
        {
            "LineNum": 1,
```

```
            "U_ItemName": "item1",
            "U_Price": 100,
            "U_Quantity": 3
        },
        {
            "LineNum": 2,
            "U_ItemName": "item2",
            "U_Price": 80,
            "U_Quantity": 4
        }
    ]
}
```

When defining a schema file based on a UDO, the object name is required instead of the ID (which is the unique identifier of a UDO). As the system does not prevent you from creating a UDO with a name identical to an existing UDO or a system object, you must pay special attention to maintain the uniqueness of the UDO name. Note that URL and request contents still require the UDO ID.

3.16.3 Retrieving Entity for UDO

You can get the order via a key field. `DocEntry` is the key field of the order.

```
GET /MyOrder(10)
```

Or

```
GET /MyOrder(DocEntry=10)
```

On success, HTTP code 200 is returned with the content of the object that is the same as the one returned by add-entity.

```
HTTP/1.1 200 OK
{
    "DocEntry": 10,
    ...
}
```

Query options are also supported. You can get a key list of all user orders.

```
GET /MyOrder?$select=DocEntry
```

Service returns:

```
HTTP/1.1 200 OK
{
    "value": [
        {"DocEntry": 1},
        {"DocEntry": 2},
```

```
        ...
        {"DocEntry": 13}
    ],
}
```

To get user orders with specified customer name and total money greater than 1000:

```
GET /MyOrder?$filter=U_CustomerName eq 'c1' and U_DocTotal gt 1000
```

User-defined schema is also supported for UDO.

For example, you created a file named `myobj.schema` in the `conf` folder:

```
{
    "MyOrder": ["DocEntry", "U_CustomerName"],
    "MyOrderLines": ["U_ItemName", "U_Quantity"]
}
```

> **Note**
>
> - Use UDO name (not UDO code) to define the schema if the name is not the same as the code.
> - By default, the object name is the same as the object code (i.e. the unique ID), and the sub-object name is the same as the sub-object code (i.e. the user table name). You can change the object name or sub-object name during registration. Code will be used in URL and request content, and name will be used in user-defined schema.
> - UDO name may contain a space, for example, "Order Lines".

Then send the request:

```
GET /MyOrder(1)
B1S-Schema: myobj.schema
```

You get:

```
HTTP/1.1 200 OK
B1S-Schema: myobj.schema

{
    "DocEntry": 10,
    "U_CustomerName": "c1",
    "MyOrderLinesCollection": [
        {
            "U_ItemName": "item1",
            "U_Quantity": 3
        },
        {
            "U_ItemName": "item2",
            "U_Quantity": 4
        }
```

```
    ]
}
```

3.16.4 Updating Entity for UDO

You can update the order, for example, change the quantity of item2 from 4 to 5 for this sales order. Note: U_DocTotal also needs to change as the quantity changes.

```
PATCH /MyOrder(10)

{
    "U_DocTotal": 700,
    "MyOrderLines": [
        {
            "LineNum": 2,
            "U_Quantity": 5
        }
    ]
}
```

On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

> **Note**
>
> PUT/PATCH/MERGE are all supported for updating. PATCH and MERGE are the same. Refer to chapter Updating Entities for differences between PATCH and PUT.

3.16.5 Deleting Entity for UDO

Use HTTP verb DELETE and the key fields to delete the entity.

```
DELETE /MyOrder(10)
```

On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

3.16.6 Canceling/Closing Entity for UDO

By default, you are not allowed to close or cancel the UDO. To enable this, first send a patch request to `UserObjectsMD` as below:

```
PATCH UserObjectsMD('MyOrder')
{ "CanClose": "tYES", "CanCancel": "tYES" }
```

To cancel the order,

```
POST /MyOrder(10)/Cancel
```

To close the order,

```
POST /MyOrder(10)/Close
```

On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

The related metadata is as follows:

```
<EntitySet EntityType="SAPB1.MYORDER" Name="MyOrder"/>

<FunctionImport IsBindable="true" Name="Cancel">
       <Parameter Name="MYORDERParams" Type="SAPB1.MYORDER"/>
</FunctionImport>

<FunctionImport IsBindable="true" Name="Close">
       <Parameter Name="MYORDERParams" Type="SAPB1.MYORDER"/>
</FunctionImport>
```

3.17 Attachments

As of SAP Business One 9.1 patch level 12, version for SAP HANA, attachment manipulation is supported through the Service Layer. The supported attachment type list is:

- pdf
- doc
- docx
- jpg
- jpeg
- png
- txt
- xls
- ppt

3.17.1 Setting up an Attachment Folder

An attachment folder is generally a shared folder on the Windows platform for the SAP Business One client.

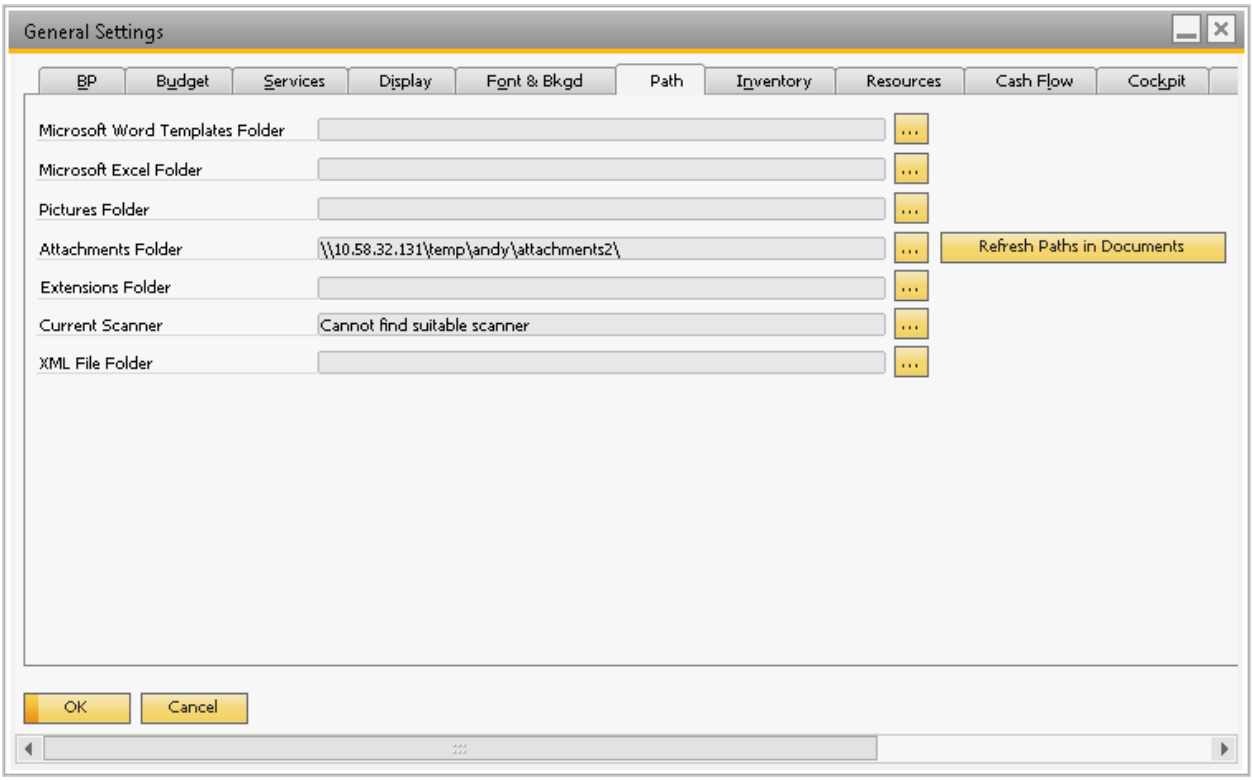
For the Service Layer running on SAP HANA on Linux, it is not allowed to directly access this shared folder. In order to make the attachment folder accessible for Service Layer as well, the Common Internet File System (CIFS) is required. For more information about CIFS, you can visit:

https://technet.microsoft.com/en-us/library/cc939973.aspx

https://www.samba.org/cifs/

Take the following steps to set up:

1. Create a network shared folder with read and write permissions on Windows (for example, \\<databaseserver>\temp\SL\attachments2) and configure it as the attachment folder in *General Settings* in the SAP Business One client.



2. Create a corresponding attachment directory on Linux, (for example, /mnt/attachments2).
3. Mount the Linux folder to the Windows folder by running a command such as this:

```
mount -t cifs -o username=xxxxx,password=******,file_mode=0777,dir_mode=0777
"//<databaseserver>/temp/SL/attachments2/" /mnt/attachments2
```

> **Note**
>
> Remember to escape in case the user name or password contains any special characters (e.g. \, $). For example:

```
mount -t cifs -o
username=global\\i066088,password=1234\$4321,file_mode=0777,dir_mode=0777
"//<databaseserver>/temp/andy/attachments2/" /mnt/attachments2
```

Remember the shared folder is a network folder rather than a local folder on Windows. Otherwise, Service Layer will not be able to access it.

 Example

How to auto mount when Linux server starts

To facilitate the configuration convenience for customers, `/etc/fstab` can be leveraged to automatically mount to the Windows shared folder once the Linux server reboots. One approach to achieve this is as follows:

1. Log in as a root user and create a credentials file (for example, `/etc/mycifspass`) with the following content:

   ```
   username=xxxxx
   password=******
   file_mode=0777
   dir_mode=0777
   ```

2. Open the system configuration file `/etc/fstab` and append one line, as follows:

   ```
   //<databaseserver>/temp/SL/attachments2/ /mnt/attachments2 cifs
   credentials=/etc/mycifspass 0 0
   ```

3. Reboot the Linux server; the Windows shared folder is automatically mounted.

3.17.2 Uploading an Attachment

Considering that the source file to upload may be on the same machine as Service Layer or on a separate machine, Service Layer has to support both of these two cases.

3.17.2.1 Uploading source file to the local Service Layer

[Applicable for Service Layer on both SAP HANA and SQL Server]

This case is similar as the attachment handling by the SAP Business One client, as the source file and the SAP Business One client are always in the same machine.

 Note

The file to upload in this case is on Linux. It can also be applied to the Windows files.

For this case, upload the source file (for example, `/home/builder/src_attachment/my_attach_1.dat`) as an attachment by sending a request such as:

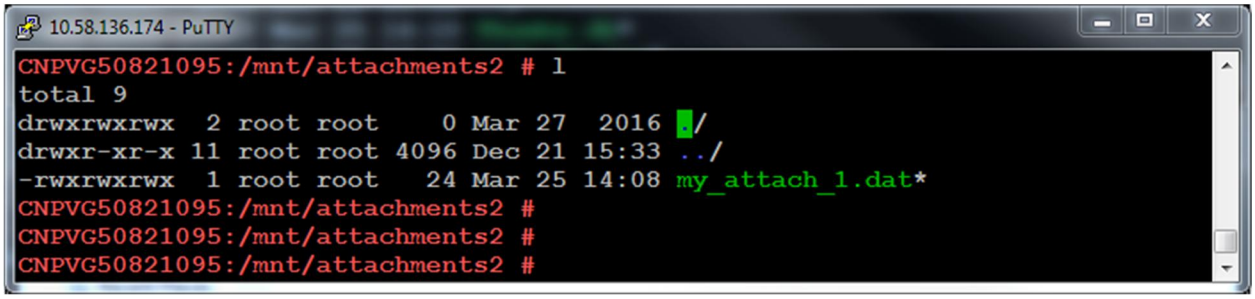
```
POST /b1s/v1/Attachments2
{
```

```
    "Attachments2_Lines": [
        {
            "SourcePath": "/home/builder/src_attachment",
            "FileName": "my_attach_1",
            "FileExtension": "dat"
        }
    ]
}
```

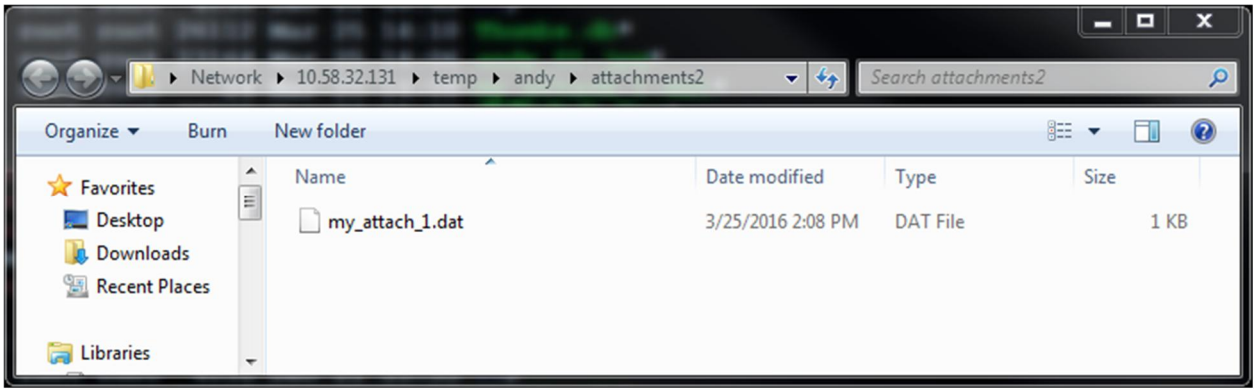
On success, the response is as follows:

```
HTTP/1.1 201 Created
{
    "AbsoluteEntry": "1",
    "Attachments2_Lines": [
        {
            "SourcePath": "/home/builder/src_attachment",
            "FileName": "my_attach_1",
            "FileExtension": "dat",
            "AttachmentDate": "2016-03-25",
            "UserID": "1",
            "Override": "tNO"
        }
    ]
}
```

The source file is saved in the destination attachment folder on Linux (`/mnt/attachments2`).



Open the Windows folder (`\\<databaseserver>\temp\SL\attachments`); the source file is saved there as well.



3.17.2.2 Upload source file to a remote Service Layer

[Applicable for Service Layer on both SAP HANA and SQL Server]

As a Web service, most times Service Layer and the source file to upload may be on separate machines, which is quite different than the attachment case in the SAP Business One client.

One way to add an attachment for this case is to use the HTTP POST method. The request must contain a `Content-Type` header specifying a content type of `multipart/form-data` and a boundary specification as:

```
Content-Type: multipart/form-data;boundary=<Boundary>
```

The body is separated by the boundary defined in the Content-Type header, such as:

```
--<Boundary>
Content-Disposition: form-data; name="files"; filename="<file1>"
Content-Type: <content type of file1>

<file1 content>
--<Boundary>
Content-Disposition: form-data; name="files"; filename="<file2>"
Content-Type: <content type of file2>

<file2 content>
--<Boundary>--
```

For example, if you want to pack two files into one attachment to post, send the request as follows:

```
POST /b1s/v1/Attachments2 HTTP/1.1
Content-Type: multipart/form-data; boundary=WebKitFormBoundaryUmZoXOtOBNCTLyxT

--WebKitFormBoundaryUmZoXOtOBNCTLyxT
Content-Disposition: form-data; name="files"; filename="line1.txt"
Content-Type: text/plain
```

```
Introduction
B1 Service Layer (SL) is a new generation of extension API for consuming B1 objects
and services
via web service with high scalability and high availability.
--WebKitFormBoundaryUmZoXOtOBNCTLyxT
Content-Disposition: form-data; name="files"; filename="line2.jpg"
Content-Type: image/jpeg

<image binary data>
--WebKitFormBoundaryUmZoXOtOBNCTLyxT--
```

On success, the response is as follows:

```
HTTP/1.1 201 Created

{
   "odata.metadata" :
"https://databaseserver:50000/b1s/v1/$metadata#Attachments2/@Element",
   "AbsoluteEntry" : "3",
   "Attachments2_Lines" : [
      {
         "SourcePath" : "/tmp/sap_b1_i066088/ServiceLayer/Attachments2/",
         "FileName" : "line1",
         "FileExtension" : "txt",
         "AttachmentDate" : "2016-04-06",
         "UserID" : "1",
         "Override" : "tNO"
      },
      {
         "SourcePath" : "/tmp/sap_b1_i066088/ServiceLayer/Attachments2/",
         "FileName" : "line2",
         "FileExtension" : "png",
         "AttachmentDate" : "2016-04-06",
         "UserID" : "1",
         "Override" : "tNO"
      }
   ]
}
```

Note

- The boundary MUST be prepended with two dashes (--) in the request body.
- The last boundary in the request body MUST be appended with two extra dashes (--).

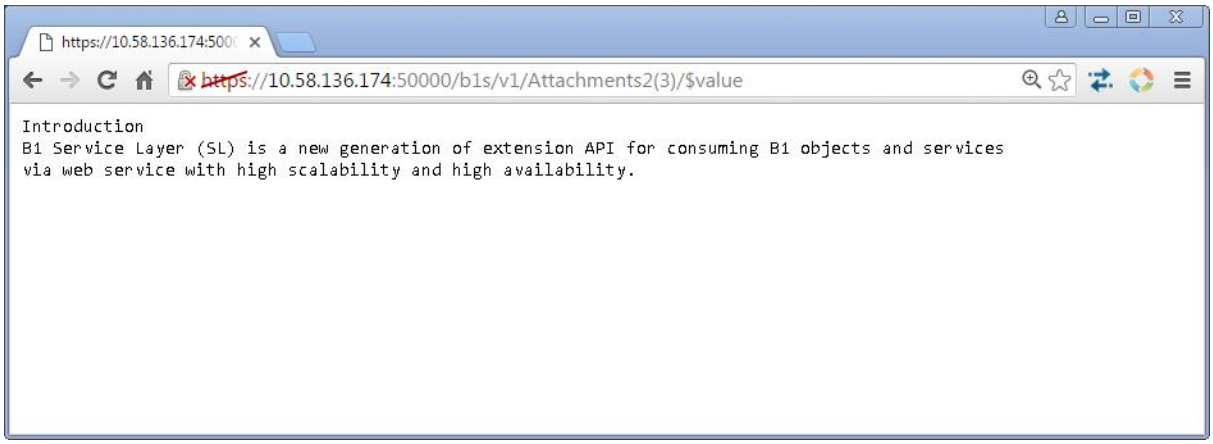
- If Service Layer returns a message about creating a file error on Linux, it indicates the permission of temporary attachment directory has been changed by someone accidentally. For this case, open a Linux terminal with root user privilege and run the below commands to recover the permission.
  ```
  sudo chown -R b1service0:b1service0 /tmp/sap_b1_b1service0
  sudo chmod -R 755  /tmp/sap_b1_b1service0
  ```

3.17.3 Downloading Attachments

By default, the first attachment line is downloaded if there are multiple attachment lines in one attachment. To download it, `$value` is required to be appended to the end of the attachment retrieval URL. For example:

```
GET /b1s/v1/Attachments2(3)/$value
```

On success, the response in browser is as follows:



If you want to download an attachment line other than the first attachment line, you need to specify the full file name (including the file extension) in the request URL. For example:

```
GET /b1s/v1/Attachments2(3)/$value?filename='line2.png'
```

On success, the response in browser is as follows:

3.17.4 Updating Attachment

Service Layer allows you to update an attachment via PATCH and there are two typical cases for this operation.

Example

How to update an existing attachment line

If the attachment line to update already exists, it is simply replaced by the new attachment line. For example:

```
PATCH /b1s/v1/Attachments2(3) HTTP/1.1
Content-Type: multipart/form-data; boundary=WebKitFormBoundaryUmZoXOtOBNCTLyxT

--WebKitFormBoundaryUmZoXOtOBNCTLyxT
Content-Disposition: form-data; name="files"; filename="line1.txt"
Content-Type: text/plain

Introduction (Updated)
B1 Service Layer (SL) is a new generation of extension API for consuming B1
objects and services via web service with high scalability and high
availability.
--WebKitFormBoundaryUmZoXOtOBNCTLyxT--
```

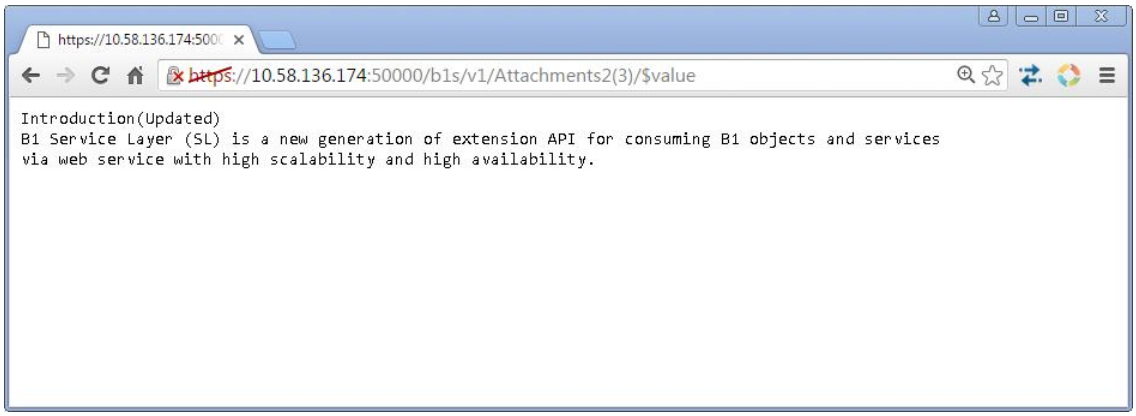
On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

To check the updated attachment line, send a request such as:

```
GET /b1s/v1/Attachments2(3)/$value?filename='line1.txt'
```

On success, the response in browser is as follows:



Example

How to add one attachment line if not existing

If the attachment line to update doesn't exist, the new attachment line is appended to the last existing attachment line. For example:

```
PATCH /b1s/v1/Attachments2(3) HTTP/1.1
Content-Type: multipart/form-data; boundary=WebKitFormBoundaryUmZoXOtOBNCTLyxT

--WebKitFormBoundaryUmZoXOtOBNCTLyxT
Content-Disposition: form-data; name="files"; filename="line3.png"
Content-Type: image/jpeg

<binary data>
--WebKitFormBoundaryUmZoXOtOBNCTLyxT--
```

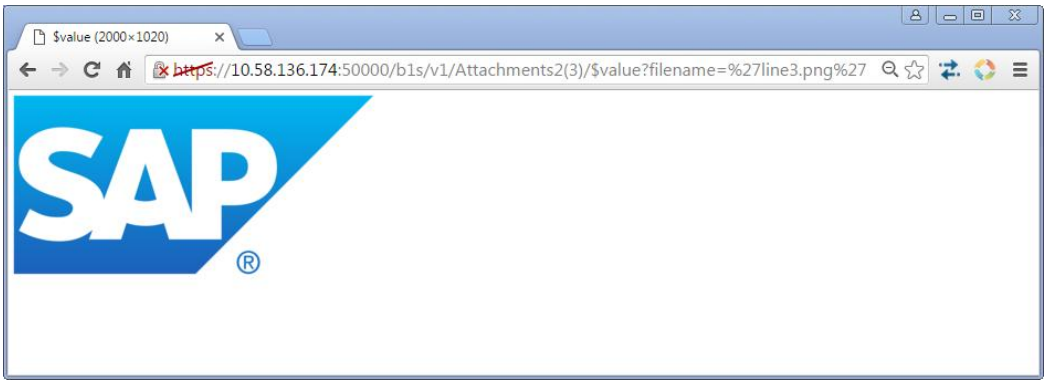
On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

To check the newly created attachment line, send a request as follows:

```
GET /b1s/v1/Attachments2(3)/$value?filename='line3.png'
```

On success, the response in browser is as follows:



Note

- From the business logic perspective, it is not allowed to delete an attachment or attachment line.
- Due to security considerations, the attachment to upload MUST be less than 50M. If not, SL responds with an error message as below:

  ```
  413 Request Entity Too Large

  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
  ```

```
<html>
    <head>
        <title>413 Request Entity Too Large</title>
    </head>
    <body>
        <h1>Request Entity Too Large</h1>
            The requested resource
        <br />/b1s/v1/Attachments2
        <br />
        does not allow request data with POST requests, or the amount of data
provided in
        the request exceeds the capacity limit.

        <p>
            Additionally, a 413 Request Entity Too Large
            error was encountered while trying to use an ErrorDocument to
handle the request.
        </p>
    </body>
</html>
```

3.18Item Image and Employee Image

As of SAP Business One 9.1 patch level 12, version for SAP HANA, Service Layer introduces a new stream entity `ItemImages` to support the CRUD operations of entity `ItemImages`. The metadata of this entity is:

```
<EntityType Name="ItemImage" m:HasStream="true">
  <Key>
    <PropertyRef Name="ItemCode"/>
  </Key>
    <Property Name="ItemCode" Nullable="false" Type="Edm.String"/>
  <Property Name="Picture" Nullable="false" Type="Edm.String"/>
</EntityType>
```

As of SAP Business One 9.3 patch level 12, version for SAP HANA, `EmployeeImages` are availabe for you in the Service Layer.

3.18.1 Setting up an Item Image Folder

The item image folder is a shared folder on Windows platform for the SAP Business One client. To make it accessible for Service Layer on Linux, CIFS is required. The setup steps for the item image folder are similar to those of the attachment folder, as follows:

1. Create a shared folder with read and write permissions on Windows (for example, `\\<databaseserver>\temp\SL\itemimages`) and configure it as the item image folder in *General Settings* of the SAP Business One client. Make sure the folder path is a network path.
2. Create a folder on Linux, (for example, `/mnt/itemimages`).
3. Mount the Linux folder to the Windows folder by running a command such as:

   ```
   mount -t cifs -o username=xxxxx,password=******,file_mode=0777,dir_mode=0777
   "//<databaseserver>/temp/SL/itemimages" /mnt/itemimages
   ```

To auto mount when the Linux server starts, use the same steps as for the attachment folder.

3.18.2 Getting an Item Image or an Employee Image

From the SAP Business One client, you can specify an item image for an item or an employee image for an employee.

To get the item image via Service Layer, send a request such as:

```
GET /b1s/v1/ItemImages('i001')/$value
```

On success, the response in the browser is as follows:

> **Note**
>
> `$value` is required to be appended to the end of the `ItemImages` retrieval URL.
>
> If `$value` is omitted, the response is as follows:
>
> {

```
        "odata.metadata":
    "https://databaseserver:50000/b1s/v1/$metadata#ItemImages/@Element",
        "odata.mediaReadLink": "ItemImages('i001')/$value",
        "odata.mediaContentType": "image/jpeg",
        "ItemCode": "'i001'",
        "Picture": "sap_1.jpg"
    }
```

To get the employee image via Service Layer, send a request such as:

```
GET b1s/v1/EmployeeImages('EmployeeID')
```

3.18.3 Updating or Uploading an Item Image

Service Layer also allows you to upload or update an item image via `PATCH`. The request must contain a Content-Type header specifying a content type of multipart/mixed and a boundary specification as:

```
Content-Type: multipart/form-data;boundary=<Boundary>
```

The body is separated by the boundary defined in the Content-Type header, such as:

```
--<Boundary>
Content-Disposition: form-data; name="files"; filename="<file>"
Content-Type: <content type of file>

<file content>
--<Boundary>--
```

The prerequisite is the item must exist. If the item does not have an image, for example, the item with `ItemCode='i001'`, a `Patch` request such as the one below uploads an image. Otherwise, the request replaces the existing item image.

```
PATCH /b1s/v1/ItemImages('i001') HTTP/1.1
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryUmZoXOtOBNCTLyxT

------WebKitFormBoundaryUmZoXOtOBNCTLyxT
Content-Disposition: form-data; name="files"; filename="sap_2.jpg"
Content-Type: image/jpeg

<image binary data>
------WebKitFormBoundaryUmZoXOtOBNCTLyxT--
```

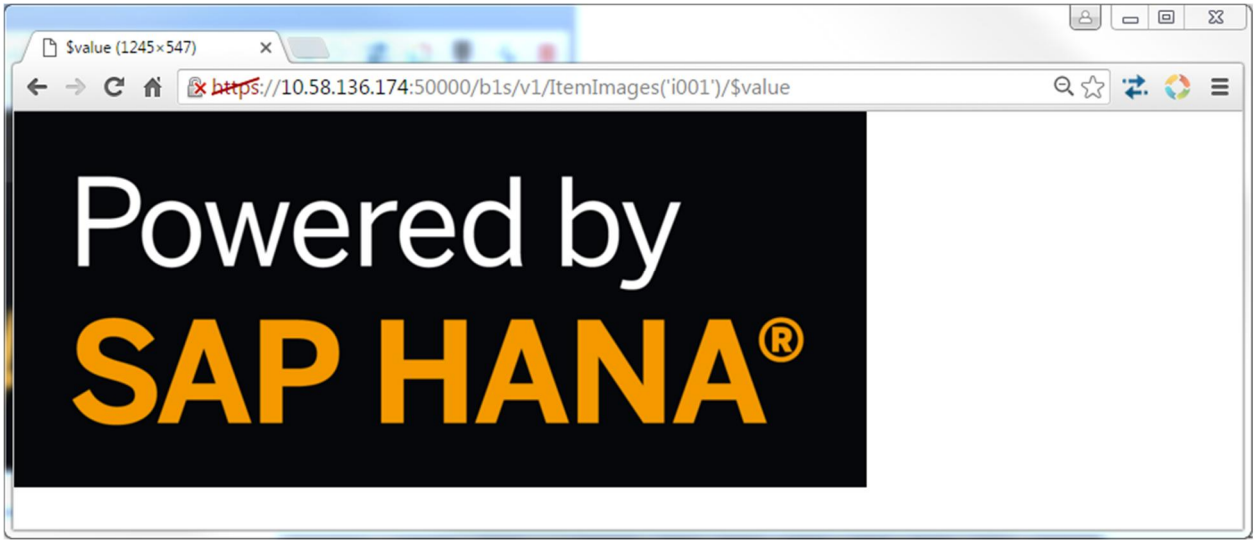
On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

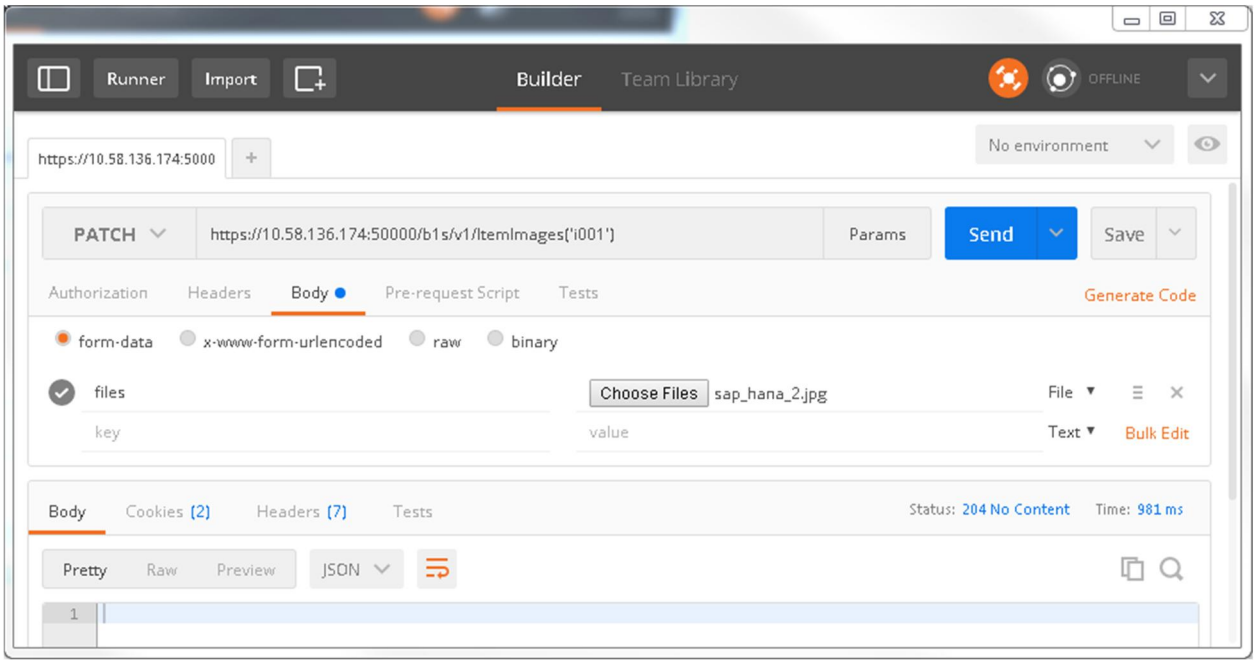
To check the updated one, send a request such as:

```
GET /b1s/v1/ItemImages('i001')/$value
```

On success, the response in the browser is as follows:



> **Note**
>
> For test purposes only, you can use the Chrome plug-in POSTMAN to update an item image.



3.18.4 Deleting an Item Image

To delete an item image, send a request such as:

```
DELETE /b1s/v1/ItemImages('i001')
```

On success, HTTP code 204 is returned without content.

```
HTTP/1.1 204 No Content
```

> **Note**
>
> It is not allowed to post an item image. You can work around that limitation by uploading an item image via PATCH.
>
> It is not allowed to query item images. To work around this issue, query the `ItemCode` and `Picure` of the entity Items instead.

3.19 JavaScript Extension

As of SAP Business One 9.2 PL04, Service Layer allows users to develop their own extension application by embedding JavaScript in the server side.

3.19.1 JavaScript Parsing Engine

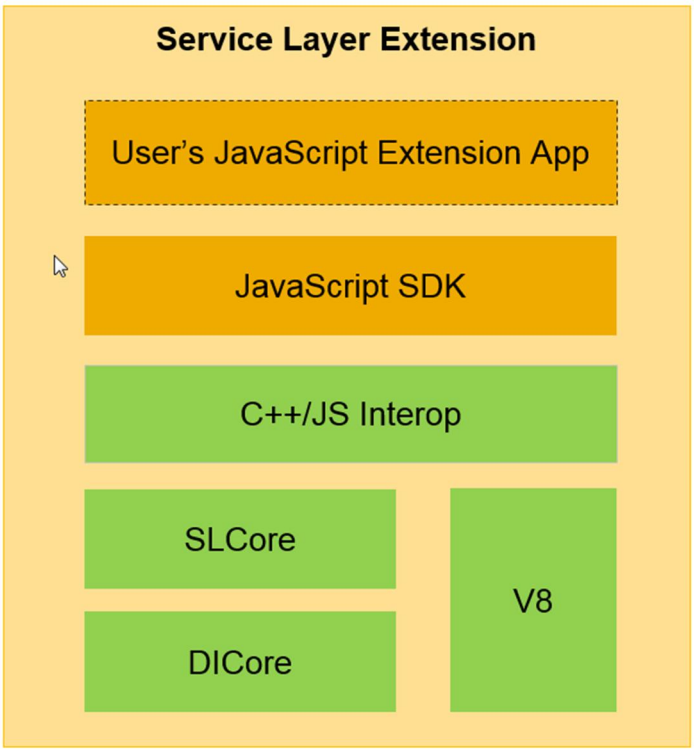
Service Layer uses Chrome V8 Engine (hereafter referred to as V8) as the JavaScript parsing engine due to the following considerations:

- Parsing performance is of significant importance for Service Layer, and V8 is a script engine known for its excellent performance.
- Service Layer and V8 are both written in C++. This would make the integration more seamless and easier.

The V8 JavaScript engine is an open source JavaScript engine developed by The Chromium Project for the Google Chrome Web browser. For more information about V8, see https://developers.google.com/v8.

3.19.2 JavaScript Extension Framework

To facilitate the development of an extension application, Service Layer provides a JavaScript framework for users to easily operate the business objects and services. The diagram below shows the basic structure of the framework.



Note

- Besides the DICore and SLCore, V8, as a new C++ component, is integrated into Service Layer.
- Service Layer adds the C++ / JavaScript interop layer to be responsible for the interaction between JavaScript and C++.
- On top of the interop layer, JavaScript SDK is designed to hide the interactive details and provide a high level and simplified API for the application layer.
- Considering the fact that switching the context between C++ and JavaScript stack is not good for performance, one target of providing the SDK is to decrease the frequency of context switching.
- Users' JavaScript Extension application is suggested to be developed based on the JavaScript SDK.

3.19.3 JavaScript Entry Function

As each executable file has a **`main`** entry function, each script file has to define entry functions. Conventionally, it is better to define four entry functions in each script file, corresponding to the CRUD operations on entities.

Each entry function has a same-name HTTP method. On receiving a request, the entry function having the same name as the http method of this request is triggered.

```
//The entry function for http request with the GET method
function GET(){
    ...
}
```

```
//The entry function for http request with the POST method
function POST(){
    ...
}

//The entry function for http request with the PATCH method
function PATCH(){
    ...
}

//The entry function for http request with the DELETE method
function DELETE(){
    ...
}
```

> **Note**
>
> Due to a keyword compatibility issue in JavaScript, each entry function should be in uppercase; otherwise the function will not be recognized.

3.19.4 JavaScript URL Mapping

Script files are triggered to run by sending requests to the specific script URL. To differentiate the script URL from a regular URL, Service Layer provides a specific URL resource path for scripts by appending `/script` to the original path `/b1s/v1` or `/b1s/v2` as below:

```
/b1s/v1/script/
/b1s/v2/script/
```

Considering the fact that different partners might define the script with the same name, Service Layer identifies which script to run by combining the partner name and the script name as the unique identifier. The mapping rule for the URL pattern is:

```
/b1s/v1/script/{{partner name}}/{{script name}}
```

Requests sent to URLs with the above pattern are dispatched to the corresponding script function defined by the corresponding partner.

> **Example**
>
> The following request will trigger the execution of the function `POST` defined in `item.js` provided by partner `mtcsys`.
>
> ```
> POST /b1s/v1/script/mtcsys/items
> ```

> **Note**
>
> A prerequisite is to ensure the script file with the corresponding `ard` file are deployed into SLD by the partner. For more details about how to deploy scripts, please refer to the chapter *JavaScript Deployment*.

3.19.5 JavaScript SDK

Similar to the DIAPI, the JavaScript SDK is intended to provide a group of APIs for programmers to easily operate on business services and business Objects. The APIs consist of entity CRUD, entity query, transactions, exceptions and http request/response.

JavaScript, as a weak-typed programming language, has many built-in favorable dynamic features. However, for the sake of programming experience and coding efficiency, the JavaScript SDK is designed like a static-language library, so as to make the most use of the auto-complete and IntelliSense functionalities provided by the modern IDE. The recommended one is the Visual Studio 2013/2015 with a `Node.js` plug-in (https://www.visualstudio.com/en-us/features/node-js-vs.aspx).

Of course, you can also choose to program dynamically and enjoy the flexible features built-in with JavaScript.

> **Note**
>
> This SDK is designed to purposely follow the Common JavaScript Specification and approximates the Node.js grammar, which is exactly the reason why the `Node.js` plug-in is recommended.

3.19.5.1 Http Request API

The http request functions listed below are packaged in the module `HttpModule.js`, which is an essential module, required to handle http requests.

API Name	API Description
`getContet()`	Returns the raw content from the request payload.
`getJsonObject()`	Returns the JSON format of the request payload.
`getMethod()`	Returns the http Verb, e.g. GET, POST, PATCH, DELETE.
`getContentType()`	Returns the MIME type of the request body (e.g. APPLICATION/JSON)
`getParameter(name)`	Returns the value of a request parameter as a String, or null if the parameter does not exist.
`getParameterNames()`	Returns an array of String objects containing the names of the parameters contained in this request.
`getEntityKey()`	Returns the entity key from the URL resource part.
getHeader(name)	Returns the value of the specified request header as a String.

3.19.5.2 Http Response API

The http response functions listed below are packaged in the module `HttpModule.js`, which is an essential module, required to handle http response.

API Name	API Description
`setHeader(name, value)`	Adds a response header with the given name and value.
`setContentType(contentType)`	Sets the content type of the response being sent to the client.
`setCharSet(charset)`	Sets the character encoding (MIME charset) of the response being sent to the client, for example, to UTF-8.
`setStatus(status)`	Sets the status code for this response.
`setContent(content)`	Sets the content in the response body
`send(status, content)`	Sends back the response to the client with the optional http status and content

Example

To handle a request such as the one below,

```
PATCH /b1s/v1/script/mtcsys/items('i001')?key1=val1 & key2=val2
DataServiceVersion:3.0

{
    "ItemName": "new name"
}
```

apply the following script:

```
var http = require('HttpModule.js');

function PATCH() {
    console.log("testing the http request and http response API...")

    var ret = {};
    ret.content = http.request.getJsonObj();
    ret.method = http.request.getMethod();
    ret.contentType = http.request.getContentType();
    ret.dataServiceVersion = http.request.getHeader("DataServiceVersion");

    ret.paramNames = http.request.getParameterNames();
    if (ret.paramNames && ret.paramNames.length) {
        ret.paramNames.forEach(function (param) {
            ret[param] = http.request.getParameter(param);
        });
    }

    ret.key = http.request.getEntityKey();
```

```
    http.response.setContentType(http.ContentType.APPLICATION_JSON);
    http.response.setStatus(http.HttpStatus.HTTP_OK);
    http.response.setContent(ret);
    http.response.send();
}
```

On success, Service Layer returns:

```
HTTP/1.1 200 OK

{
    "content": {
        "ItemName": "new name"
    },
    "method": "PATCH",
    "contentType": "text/plain;charset=UTF-8",
    "dataServiceVersion": "3.0",
    "paramNames": [
        "key1",
        "key2"
    ],
    "key1": "val1",
    "key2": "val2",
    "key": "'i001'"
}
```

Note

- Similar to `Node.js`, `require` is a global function to import a module and return a reference to that module. The above example indicates `http` is a reference of the module `HttpModule.js`.
- `request` and `response` are two members of `http`, representing a pre-created `HttpRequest` instance and an `HttpResponse` instance, respectively.
- To facilitate HTTP programming, module `HttpModule.js` also defines HTTP utility constants, for example, `HttpStatus`, `ContentType`.

3.19.5.3 Entity CRUD API

Each exposed entity supports CRUD operations by default. The relevant APIs are packaged in the module `ServiceLayerContext.js`.

- For most cases, to perform CRUD operations on an entity, you first must create an entity instance, if the entity name is known in advance. Then call the following group of APIs defined in the prototype of `EntitySet`:

  Prototype of EntitySet

API Name	API Description
`add(content, callback)`	Creates an entity by the content and the optional callback function on creation.
`get(key, callback)`	Retrieves an entity by the key and the optional callback function on retrieval.
`update(content, key, callback)`	Updates an entity by the content, key and the optional callback function on update.
`remove(key, callback)`	Removes an entity by the key and the optional callback function on removal.
`...`	...

- For the scenario where the entity name is not know in advance or the entity is a dynamically created UDO, you first must create a `ServiceLayerContext` instance. Then call the following group of APIs against this instance.

  Prototype of ServiceLayerContext

API Name	API Description
`add(name, content, callback)`	Creates an entity by the name, content and the optional callback function on creation.
`get(name, key, callback)`	Retrieves an entity by the name, key, and the optional callback function on retrieval.
`update(name, content, key, callback)`	Updates an entity by the name, content and key and the optional callback function on update.
`remove(name, key, callback)`	Removes an entity by the name, key and the optional callback function on removal.
`...`	...

  Example

  To handle a request such as the one below,

  ```
  POST /b1s/v1/script/mtcsys/test_items_more
  ```

  apply the following script:

  ```
  var ServiceLayerContext = require('ServiceLayerContext.js');
  var Item = require('EntityType/Item.js');
  var http = require('HttpModule.js');

  var test_item_code = "i001";
  function POST() {
      var slContext = new ServiceLayerContext();
      var ret = [];

      var item = new Item();
  ```

```
    item.ItemCode = test_item_code;

    var dataSrvRes = slContext.Items.add(item);
    if (!dataSrvRes.isOK()) {
        throw http.ScriptException(http.HttpStatus.HTTP_BAD_REQUEST, "create
entity failure")
    }
    ret.push({ "operation": dataSrvRes.operation, "status": dataSrvRes.status
});

    var key = test_item_code;
    var dataSrvRes = slContext.Items.get(key);
    if (!dataSrvRes.isOK()) {
        throw
http.ScriptException(http.HttpStatus.HTTP_INTERNAL_SERVER_ERROR, "retrieve
entity failure")
    }
    ret.push({ "operation": dataSrvRes.operation, "status": dataSrvRes.status
});

    item.ItemName = 'new_item_name';
    dataSrvRes = slContext.update("Items", item, key);//equivalent to
slContext.Items.update(item, key);
    if (!dataSrvRes.isOK()) {
        throw
http.ScriptException(http.HttpStatus.HTTP_INTERNAL_SERVER_ERROR, "update
entity failure")
    }
    ret.push({ "operation": dataSrvRes.operation, "status": dataSrvRes.status
});

    dataSrvRes = slContext.remove("Items", key);//equivalent to
slContext.Items.remove(key);
    if (!dataSrvRes.isOK()) {
        throw
http.ScriptException(http.HttpStatus.HTTP_INTERNAL_SERVER_ERROR, "delete
entity failure")
    }
    ret.push({ "operation": dataSrvRes.operation, "status": dataSrvRes.status
});

    http.response.send(http.HttpStatus.HTTP_OK, ret);
}
```

On success, Service Layer returns:

```
HTTP/1.1 200 OK
[
    {
        "operation": "add",
        "status": 201
    },
    {
        "operation": "get",
        "status": 200
    },
    {
        "operation": "update",
        "status": 204
    },
    {
        "operation": "remove",
        "status": 204
    }
]
```

3.19.5.4 Entity Query API

Query APIs are packaged in the module `ServiceLayerContext.js`, and similar to the CRUD API, they are defined both on the `EntitySet` and the `ServiceLayerContext` prototype.

Prototype of EntitySet

API Name	API Description
`query(queryOption, isCaseInsensitive)`	Performs a case-sensitive or case-insensitive query and return the entities satisfying the query options.
`count(queryOption, isCaseInsensitive)`	Performs a case-sensitive or case-insensitive query and return the number of the entities satisfying the query options.
`...`	...

Prototype of ServiceLayerContext

API Name	API Description
`query(name, queryOption, isCaseInsensitive)`	Performs a case-sensitive or case-insensitive query and return the entities with the given name and satisfying the query options.

API Name	API Description
`count(name, queryOption, isCaseInsensitive)`	Performs a case-sensitive or case-insensitive query and return the number of the entities with the given name and satisfying the query options.
`...`	...

Note

The paramenter `isCaseInsensitive` is for SAP HANA database only. If you are working with SAP HANA database, by default, the query is case-sensitive, due to the default Unicode collation for SAP HANA database. Specifying the flag `isCaseInsensitive` as true would issue a case insensitive query. However, the query performance would not be as efficient as with a case-insensitive query.

As of SAP Business One 9.2 PL07, version for SAP HANA, case-insensitive query is supported.

Example

To handle a request such as the one below,

```
GET /b1s/v1/script/mtcsys/test_query_businesspartner
```

apply the following script:

```
var ServiceLayerContext = require('ServiceLayerContext.js');
var http = require('HttpModule.js');
function GET() {
    var queryOption = "$select=CardName, CardCode &
$filter=contains(CardCode, 'c1') & $top=5 & $orderby=CardCode";
    var slContext = new ServiceLayerContext();
    var retCaseSensitive = slContext.BusinessPartners.query(queryOption);
    var retCaseInsensitive = slContext.query("BusinessPartners", queryOption,
true);

    http.response.setStatus(http.HttpStatus.HTTP_OK);
    http.response.setContent({ "CaseSensitive": retCaseSensitive.toArray(),
"CaseInsensitive": retCaseInsensitive.toArray() });
    http.response.send();
}
```

On Success, Service Layer returns:

```
HTTP/1.1 200 OK

{
    "CaseSensitive": [
        {
            "CardCode": "c1",
            "CardName": "customer c11"
        }
    ],
```

```
    "CaseInsensitive": [
        {
            "CardCode": "c1",
            "CardName": "customer c11"
        },
        {
            "CardCode": "C11",
            "CardName": null
        },
        {
            "CardCode": "C12",
            "CardName": null
        }
    ]
}
```

Example

To handle a case insensitive request:

```
GET /b1s/v1/BusinessPartners?$filter=contains(CardCode,
'c2')&$select=CardCode
B1S-CaseInsensitive: true
```

On success, Service Layer returns:

```
HTTP/1.1 200 OK

{
    "value": [
        {
            "CardCode": "C20000"
        },
        {
            "CardCode": "C23900"
        },
        {
            "CardCode": "c21"
        },
        {
            "CardCode": "c22"
        }
    ]
}
```

3.19.5.5 Transaction API

Transaction APIs, such as those listed below, are packaged in the module `ServiceLayerContext.js`, which is an essential module, required to control transactions.

API Name	API Description
`startTransaction`	Starts a transaction.
`commitTransaction`	Commits a transaction.
`rollbackTransaction`	Rollbacks a transaction
`isInTransaction`	Returns true if the current operation is in a transaction

Example

To handle a request such as the one below,

```
POST /b1s/v1/script/mtcsys/test_create_businesspartner


[
  {
    "CardCode": "c001",
    "CardName": "c001"
  },
  {
    "CardCode": "c002",
    "CardName": "c002"
  },
  {
    "CardCode": "c003",
    "CardName": "c003"
  },
  {
    "CardCode": "c004",
    "CardName": "c004"
  },
  {
    "CardCode": "c005",
    "CardName": "c005"
  }
]
```

apply the following script:

```
var ServiceLayerContext = require('ServiceLayerContext.js');
var http = require('HttpModule.js');
```

```
var BusinessPartner = require('EntityType/BusinessPartner.js');

function POST() {

    var slContext = new ServiceLayerContext();
    var bpList = http.request.getJsonObj();
    if (!(bpList instanceof Array)) {
        throw http.ScriptException(http.HttpStatus.HTTP_BAD_REQUEST, "invalid
format of payload");
    }

    slContext.startTransaction();
    for (var i = 0; i < bpList.length; ++i) {
        var res = slContext.BusinessPartners.add(bpList[i]);
        if (!res.isOK()) {
            slContext.rollbackTransaction();
            throw http.ScriptException(http.HttpStatus.HTTP_BAD_REQUEST,
res.getErrMsg());
        }
    };
    slContext.commitTransaction();

    http.response.setContentType(http.ContentType.TEXT_PLAIN);
    http.response.send(http.HttpStatus.HTTP_OK, "transaction committed");
}
```

On Success, Service Layer returns:

```
HTTP/1.1 200 OK

transaction committed
```

Send this request again, Service Layer returns:

```
HTTP/1.1 400 Bad Request

{
   "error" : {
      "code" : 600,
      "message" : {
         "lang" : "en-us",
         "value" : "1320000140 - Business partner code 'c001' already
assigned to a business partner; enter a unique business partner code"
      }
   }
```

```
}
```

Note

- Programmers should be aware that transaction operations are expensive and big transactions degrade Web service throughput. Thus, Service Layer imposes a limitation on the transaction size. The total operations in one transaction should be no more than 10.
- Please keep in mind that the following transactions should be called as pairs: `startTransaction/commitTransaction` or `startTransaction/rollbackTransaction`.

3.19.5.6 Exception API

3.19.5.6.1 Compile Exception

Service Layer responds with an error message to the client if there is a compilation error in a user's script.

Example

```
var Document = require('EntityType/Document.js');

//type mistake: ';' should be ','
var line = Document.DocumentLine.create({
    ItemCode: 'i001'; Quantity: 2, UnitPrice: 10
});

var lines = new Document.DocumentLineCollection();
lines.add(line);
```

The above code would result in an error message such as the one below:

```
{
  "error": {
    "code": 511,
    "message": {
      "lang": "en-us",
      "value": "Script error: compile error [SyntaxError: Unexpected token
;]."
    }
  }
}
```

3.19.5.6.2 Runtime Exception

Service Layer responds with an error message to the client if there is a runtime error in a user's script.

Example

```
var ServiceLayerContext = require('ServiceLayerContext.js');
//var Bank = require('EntityType/Bank.js');
var bank = new Bank();
bank.BankCode = 'bank01';
var res = new ServiceLayerContext().Banks.add(bank);
if (!res.isOK) {
}
```

The above code would result in an error message such as the one below:

```
HTTP/1.1 400 Bad Request

{
  "error": {
    "code": 512,
    "message": {
      "lang": "en-us",
      "value": "Script error: runtime error [ReferenceError: Bank is not
defined]."
    }
  }
}
```

3.19.5.6.3 User Exception

Service Layer also allows users to explicitly propagate exceptions by throwing `ScriptException` exported from the http module.

Example

```
var ServiceLayerContext = require('ServiceLayerContext.js');
var Order = require('EntityType/Document.js');
var http = require('HttpModule.js');

var slContext = new ServiceLayerContext();
var res = slContext.Orders.get(10000);
if (!res.isOK()) {
```

```
    throw new http.ScriptException(http.HttpStatus.HTTP_NOT_FOUND, "the given
order is not found");
}
```

The above code would result in an error message such as the one below:

```
HTTP/1.1 404 Not Found

{
  "error": {
    "code": 600,
    "message": {
      "lang": "en-us",
      "value": "the given order not found"
    }
  }
}
```

3.19.6 Logging

Currently, debugging script is not supported. However, users are allowed to log the key information during script programming by using the `API console.log`:

```
console.log('Hello, Service Layer Scripting!');
```

> **Note**
>
> `console` is a global object. Literally, the output of this object should be printed in the console. However, considering Service Layer is a backend service, the output is redirected to log files under `{SL Installation Path}/logs/script/`.

3.19.7 JavaScript SDK Generator Tool

Considering that in each patch there might be new business objects exposed or new changes made on the existing objects, the SDK would be adjusted accordingly to adapt to the changes.

To manually maintain the SDK would not only need huge efforts, but also would be error-prone. To automatically address this issue, a tool named `Metadata2JavaScript` is provided to generate the SDK according to the metadata, as metadata reflects all changes on the business objects.

This tool supports generating the SDK in two ways (let's take the Linux environment as an example):

- From a local metadata file:

  ```
  Metadata2JavaScript -a {local metadata file} -o {output folder, default is
  ./b1s_sdk}
  ```

  or

```
Metadata2JavaScript --addr {local metadata file} --output {output folder, default
is ./b1s_sdk}
```

For example:

```
Metadata2JavaScript -a metadata.xml -o ./output
```

- From a remote Service Layer instance:

```
Metadata2JavaScript -a {SL base url} -u {user} -p {password} -c {company} -o
{output folder, default is ./b1s_sdk}
```

or

```
Metadata2JavaScript --addr {SL base url} --user {user} --password {password} --
company {company} --output {output folder, default is ./b1s_sdk}
```

For example:

```
Metadata2JavaScript --addr https://databaseserver:50000/b1s/v1/ --user manager --
password 1234 --company SBODEMOUS
```

> **Note**
>
> This tool is released together with Service Layer and is available in the `bin` folder of the Service Layer installation path.
>
> As this tool depends on JAVA JRE, before running it, make sure the relevant JAVA environment variables are correctly exported.
>
> In the Linux environment, set `JAVA_HOME` as below:
>
> ```
> export JAVA_HOME=/usr/sap/SAPBusinessOne/Common/sapjvm_8/jre
> export PATH=$JAVA_HOME/bin:$PATH
> ```

Previously, the Metadata2Javascript tool is available in the Linux environment only. As of SAP Business One 10.0 FP 2011, this tool is also available in the Microsoft Windows environment. There are slight differences when you use the tool:

- In the Linux environment, you enter the command starting with `Metadata2JavaScript`.
- In the Microsoft Windows environment, you enter the command starting with `java -jar Metadata2JavaScript.jar`.

In the above examples, for the Microsoft Windows environment, change the command to the following:

```
java -jar Metadata2JavaScript.jar -a metadata.xml -o ./output
java -jar Metadata2JavaScript.jar --addr https://databaseserver:50000/b1s/v1/ --user
manager --password 1234 --company SBODEMOUS
```

3.19.8 JavaScript Deployment

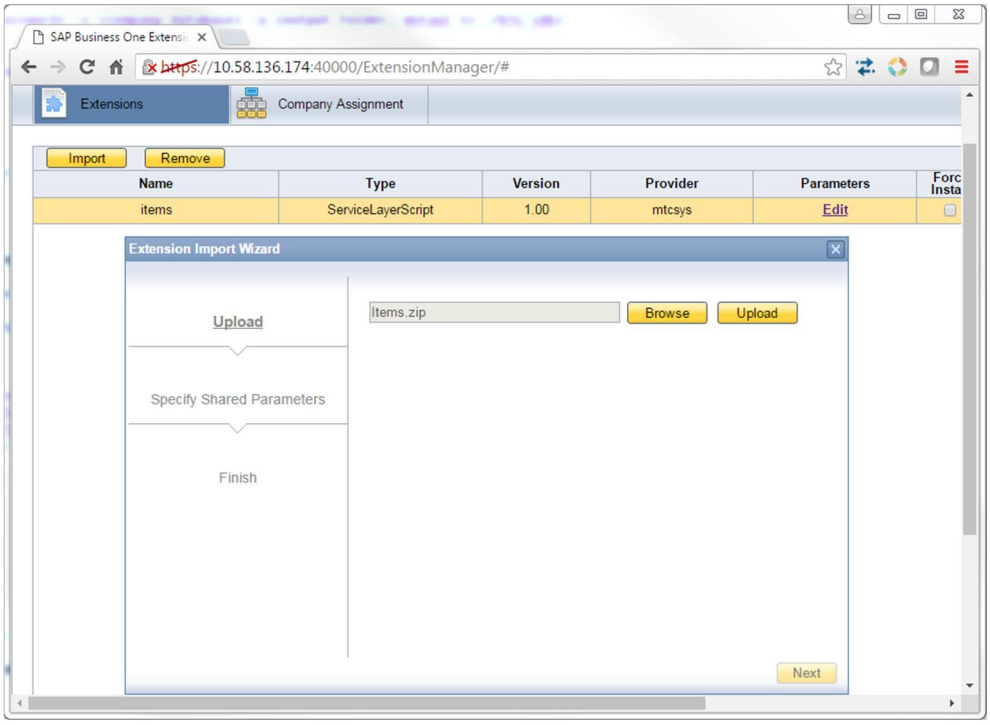
Service Layer reuses the extension manager to manage the life cycle of script files. Similar to the DIAPI add-on, extension applications developed by Service Layer are deployed to SLD as well.

Assume you have a script file `Items.js`; take the following steps to deploy it:

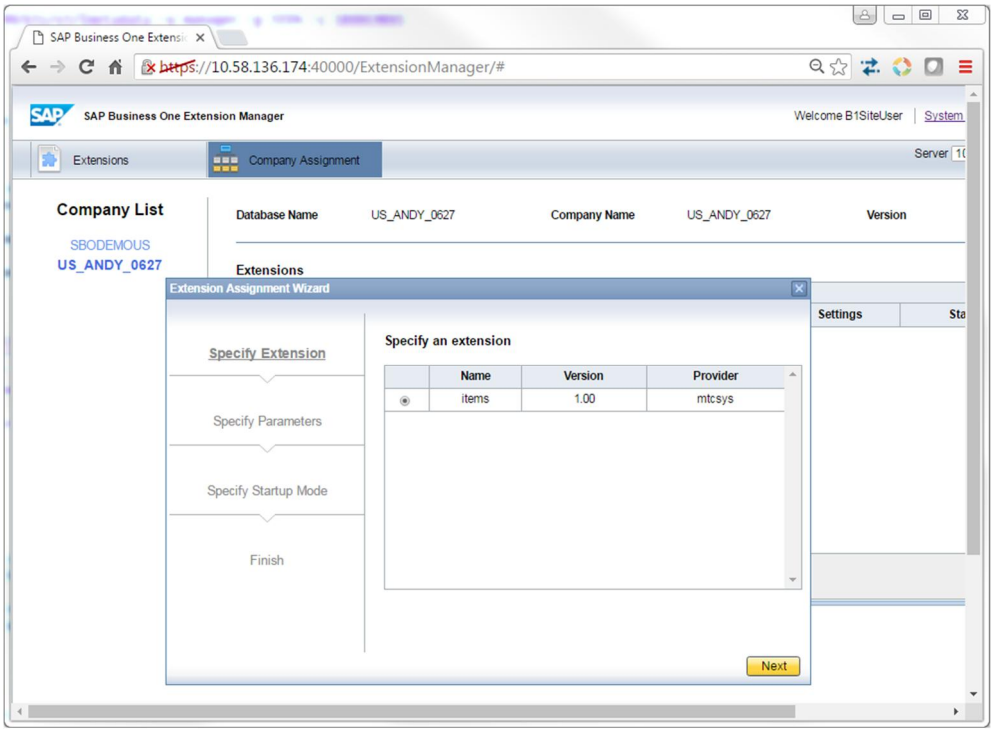
1. Create an `ard` file named `Items.ard` in the below format to describe the meta of this script file. Meanwhile, the `ard` file can also be used to determine the script URL path.

```
<?xml version="1.0" encoding="utf-8"?>
<AddOnRegData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              SlientInstallation="" SlientUpgrade="" Partnernmsp="mtcsysnm"
SchemaVersion="3.0"
              Type="ServiceLayerScript" OnDemand="" OnPremise=""
ExtName="ItemsExt"
              ExtVersion="1.00" Contdata="sa" Partner="mtcsys" DBType="HANA"
ClientType="S">
  <ServiceLayerScripts>
      <Script Name="items" FileName="Items.js"></Script>
  </ServiceLayerScripts>
  <XApps>
    <XApp Name="" Path="" FileName="" />
  </XApps>
</AddOnRegData>
```

2. Compress the `ard` file and script file into a `zip` file (e.g. `Items.zip`).
3. Upload Items.zip to the extension manager from the *Extension Import Wizard*.



4. From the *Extension Assignment Wizard*, assign the extension application to one company.



5. Log in to the company with Service Layer and access the script with the following URL:

   `/b1s/v1/script/mtcsys/items`

   Note

   - The script URL is a combination of partner name and script name separated by a '/' appended to the Service Layer base URL `/b1s/v1/`.
   - Currently, Service Layer does not support compressing multiple script files into one `ard` file.
   - For more details about how to deploy extension applications, please reference the guide *How to Package and Deploy SAP Business One Extensions for Lightweight Deployment*.
   - In the `ard` file, do not name the value of the attribute `Partner` as `test`, as `test` is a reserved word for internal testing.

3.19.9 Typical User Cases of Applying Script

3.19.9.1 Complex Transactions

Scripting can be used in transaction scenarios, which is an important complement to the OData Batch operations. The following is an example for adding an order and a delivery based on the order in one transaction, which would be impossible without scripting.

Example

```
var ServiceLayerContext = require('ServiceLayerContext.js');
var http = require('HttpModule.js');
var Order = require('EntityType/Document.js');
var DeliveryNote = require('EntityType/Document.js');

/*
 * Entry function for the POST http request.
 *
 */
function POST() {
    var order = new Order();

    order.CardCode = 'c1';
    order.DocDate = new Date();
    order.DocDueDate = new Date();

    var line = new Order.DocumentLine();
    line.ItemCode = 'i1';
    line.Quantity = 1;
    line.UnitPrice = 10;

    var line2 = new Order.DocumentLine();
    line2.ItemCode = 'i2';
    line2.Quantity = 1;
    line2.UnitPrice = 10;

    order.DocumentLines = new Order.DocumentLineCollection();
    order.DocumentLines.add(line);
    order.DocumentLines.add(line2);

    var slContext = new ServiceLayerContext();

    //start the transaction
    slContext.startTransaction();
    var res = slContext.Orders.add(order);
    if (!res.isOK()) {
        slContext.rollbackTransaction();
        return http.response.send(http.HttpStatus.HTTP_BAD_REQUEST, res.body);
    }
```

```
    //get the newly created order from the response body.
    var newOrder = Order.create(res.body);

    //create a delivery based on the order
    var deliveryNote = new DeliveryNote();
    deliveryNote.DocDate = newOrder.DocDate;
    deliveryNote.DocDueDate = newOrder.DocDueDate;
    deliveryNote.CardCode = newOrder.CardCode;
    deliveryNote.DocumentLines = new DeliveryNote.DocumentLineCollection();
    for (var lineNum = 0; lineNum < order.DocumentLines.length; ++lineNum) {
        var line = new DeliveryNote.DocumentLine();
        line.BaseType = 17;
        line.BaseEntry = newOrder.DocEntry;
        line.BaseLine = lineNum;
        deliveryNote.DocumentLines.add(line);
    }

    res = slContext.DeliveryNotes.add(deliveryNote);
    if (!res.isOK()) {
        slContext.rollbackTransaction();
        return http.response.send(http.HttpStatus.HTTP_BAD_REQUEST, res.body);
    }else{
        slContext.commitTransaction();
        return http.response.send(http.HttpStatus.HTTP_CREATED, res.body);
    }
}
```

3.19.9.2 Customized Business Logic (e.g. UDO)

Another typical case for scripting is to add customized business logic during the process of operating user-defined objects (UDO). The following is an example for performing some validations and calculating the `DocTotal` when creating the UDO named `MyOrder`.

Example

```
POST /b1s/v1/script/mtcsys/test_myorder

{
  "U_CustomerName": "c1",
  "U_DocTotal": 0,
```

```
  "MyOrderLinesCollection": [
    {
      "U_ItemName": "i1",
      "U_Price": 100,
      "U_Quantity": 3
    },
    {
      "U_ItemName": "i2",
      "U_Price": 80,
      "U_Quantity": 4
    }
  ]
}
```

Apply the following script to handle the above request:

```
function POST() {
//Before creating the UDO, users are allowed to add extra logic.
    var myOrder = http.request.getJsonObj();
    var slContext = new ServiceLayerContext();


//Example 1 : added logic to validate if each item exists and the item stock is
enough.
    myOrder.MyOrderLinesCollection.forEach(function (line) {
        var dataSvcRes = slContext.Items.get(line.U_ItemName);
        if (!dataSvcRes.isOK()) {
            throw new http.ScriptException(http.HttpStatus.HTTP_NOT_FOUND, "item
not found");
        } else {
            //Convert weak type to strong type by calling Item.create. The
conversion is not a must.
            //You can also use dataSvcRes.body.QuantityOnStock
            var item = Item.create(dataSvcRes.body);
            if (item.QuantityOnStock < line.U_Quantity) {
                throw new http.ScriptException(http.HttpStatus.HTTP_BAD_REQUEST,
"not enough items on stock");
            }
        }
    });

//Example 2 : added logic to calculate the DocTotal
    myOrder.U_DocTotal = 0;
```

```
        myOrder.MyOrderLinesCollection.forEach(function (line) {
            myOrder.U_DocTotal += (line.U_Price * line.U_Quantity);
        });

    //Add this UDO
        var res = slContext.add("MyOrder", myOrder);
        if (res.isOK()) {
            http.response.send(http.HttpStatus.HTTP_CREATED, res.body);
        } else {
            http.response.send(http.HttpStatus.HTTP_BAD_REQUEST, res.body);
        }
    }
```

3.19.10 Consume Script Service from .Net Application

For purposes of flexibility, SL allows the response from the script to be highly-customized. It is not appropriate to define the fixed metadata for the scripting, and as such, using the single WCF framework is not possible to consume the script service. As an alternative, it is suggested to program with the .NET Web Http library mixed with WCF, illustrated with the below code snippet.

```
[TestFixture]
    class ScriptOrdersTest : AppCommon.GeneralTestGroup
    {
        [SetUp]
        public void setup()
        {
            ServicePointManager.ServerCertificateValidationCallback += delegate(object
sender, X509Certificate cert, X509Chain chain, SslPolicyErrors ssl) { return true; };
            ServicePointManager.Expect100Continue = false;
            ServicePointManager.MaxServicePointIdleTime = 2000;
        }

        private string m_cookie = AppCommon.WebConnection.Instance.SessionID;
        private Uri m_baseUri = new Uri(AppCommon.ConfigInfo.Instance().SL_URL);
        private int m_docEntry = 0;
        [Test]
        public void test01_create()
        {
            Document order = new Document();
            order.CardCode = "c1";
            order.DocDate = DateTime.Now;
```

```
            order.DocDueDate = DateTime.Now;
            {
                DocumentLine line = new DocumentLine();
                line.LineNum = 1;
                line.ItemCode = "i1";
                line.Quantity = 1;
                line.UnitPrice = 10;
                order.DocumentLines.Add(line);
            }
            {
                DocumentLine line = new DocumentLine();
                line.LineNum = 2;
                line.ItemCode = "i2";
                line.Quantity = 1;
                line.UnitPrice = 10;
                order.DocumentLines.Add(line);
            }

            try
            {
                var setting = new JsonSerializerSettings() { NullValueHandling =
NullValueHandling.Ignore };
                string json = JsonConvert.SerializeObject(order, setting);
                var data = Encoding.ASCII.GetBytes(json);
                HttpWebRequest request = (HttpWebRequest)WebRequest.Create(new
Uri(m_baseUri, "script/test/test_orders"));
                request.CachePolicy = new
System.Net.Cache.RequestCachePolicy(System.Net.Cache.RequestCacheLevel.NoCacheNoStore)
;
                request.Method = "POST";
                request.KeepAlive = false;
                request.Headers["Cookie"] = m_cookie;
                request.ContentType = "application/json;odata=minimalmetadata";
                request.ContentLength = data.Length;

                using (var stream = request.GetRequestStream())
                {
                    stream.Write(data, 0, data.Length);
                }
                HttpWebResponse response = (HttpWebResponse)request.GetResponse();
                Assert.AreEqual(response.StatusCode, HttpStatusCode.Created);
```

```
                var responseString = new
StreamReader(response.GetResponseStream()).ReadToEnd();
                Document newEntity =
JsonConvert.DeserializeObject<Document>(responseString);
                Assert.IsTrue(newEntity.DocEntry > 0);
                Assert.AreEqual(newEntity.DocumentLines.Count(),
order.DocumentLines.Count());
                response.Close();
                m_docEntry = newEntity.DocEntry;
            }
            catch (WebException ex)
            {
                WebResponse response = ex.Response;
                if (response == null)
                {
                    throw SetResultMessage(ex);
                }
                var responseString = new
StreamReader(response.GetResponseStream()).ReadToEnd();
                throw SetResultMessage(new Exception(responseString));
            }
            catch (Exception ex)
            {
                throw SetResultMessage(ex);
            }
        }
```

3.20 Cross Origin Resource Sharing (CORS)

As of SAP Business One 9.1 patch level 08, version for SAP HANA, CORS is supported to allow trusted origins to access the resource of Service Layer. For more information about CORS, please check the links below:

http://enable-cors.org/

http://www.html5rocks.com/en/tutorials/cors/#toc-withcredentials

3.20.1 Enabling CORS

By default, a cross domain request is rejected due to the security settings of the browser. To enable CORS, open `b1s.conf` and append two configuration items. For example:

```
"CorsEnable": true,
"CorsAllowedOrigins": "http://host1:8080;https://host2:8443"
```

You can refer to Configuration Options for Service Layer for more details about the CORS configurations.

3.20.2 Enable to Configure Allowed Headers

As of SAP Business One 9.2, version for SAP HANA patch level 07, request headers are allowed to configure in `b1s.conf`.

By default, only `content-type` and `accept` are allowed in the CORS process. However, under some conditions, other headers are needed, e.g. `B1S-CaseInsensitive`. To satisfy this requirement, append the configuration option `CorsAllowedHeaders` in `b1s.conf`. For example:

```
"CorsAllowedHeaders":"content-type, accept, B1S-CaseInsensitive"
```

> **Note**
>
> You can refer to Configuration Options for Service Layer for more details about the CORS configurations.

3.20.3 CORS process

Once CORS is enabled, browsers first issue an `OPTIONS` request (a preflight request), which is like asking the server for permission to make the actual request. Once permissions have been granted, the browser makes the actual request. The browser handles the details of these two requests transparently. The preflight response can also be cached so that it is not issued on every request. Take the requests received by Service Layer as an example:

```
[11936] 5- b1s_handler: OPTIONS /b1s/v1/Login from 10.58.81.2
[11936] 6- b1s_handler: POST /b1s/v1/Login from 10.58.81.2
[11936] 7- b1s_handler: OPTIONS /b1s/v1/Items from 10.58.81.2
[11936] 8- b1s_handler: POST /b1s/v1/Items from 10.58.81.2
```

3.21Ping Pong API

As of SAP Business One 9.3 patch level 10, version for SAP HANA, Service Layer provides a new Ping Pong API method which can improve debugging, support, network testing and component monitoring. The purpose of this API is to provide a direct response from the Apache server so that you can eliminate SAP Business One internal processing time from any network performance debugging. (This is different from all other Service Layer APIs

which are passed through to SAP Business One core for processing before returning a result). In response to a PING request, the Apache server (load balancer or node) will respond directly with a simple PONG response.

This API could be used to fulfill the following scenarios:

- Isolate network latency from SAP Business One processing latency
- Check server time accuracy
- Monitor or debug Service Layer API availability
- Monitor load balancer and nodes separately (important for multi-server deployment)

Example

The following scenarios are some examples of how to use the Ping Pong API:

- Scenario 1 - No endpoint specified, load balancer will respond

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  ```

  ```
  {    "message": "pong",    "sender": "load balancer",    "timestamp":
  "1555998764.740"}
  ```

- Scenario 2 - Ping Service Layer load balancer

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/load-balancer
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  ```

  ```
  {    "message": "pong",    "sender": "load balancer",    "timestamp":
  "1555998785.080"}
  ```

- Scenario 3 - Specified node will respond

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/node/1
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  ```

  ```
  {    "message": "pong",    "sender": "node1",    "timestamp":
  "1554363811.386"}
  ```

- Scenario 4 - Specified node will respond

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/node/2
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  ```

  ```
  {    "message": "pong", "sender":"node2", "timestamp": "1552263107.648"}
  ```

- Scenario 5 - No node specified, node 1 will respond

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/node
  ```

  Response:

  ```
  HTTP/1.1 200 OK
  {    "message": "pong",    "sender": "node1",    "timestamp":
  "1555998837.832"}
  ```

- Scenario 6 - Node 4 is down

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/node/4
  ```

  Response:

  ```
  HTTP/1.1 503 Service Unavailable
  {    "message": "Service Unavailable"}
  ```

- Scenario 7 - Node 5 does not exist

  Request:

  ```
  https://<ServerName/IP>:<Port>/ping/node/5
  ```

  Response:

  ```
  HTTP/1.1 503 Service Unavailable
  {    "message": "node 5 does not exist",    "sender": "load balancer",
  "timestamp": "1554363934.431"}
  ```

4 SQL Query

As of SAP Business One 10.0 FP 2011, the Service Layer on Microsoft SQL Server and SAP HANA supports a highly flexible SQL Query.

Via the Service Layer, database views are allowed to be consumed to retrieve unexposed data, which is a good complement to the OData Query. However, it is impossible to deploy these views automatically, so various manual steps are needed to require full access to the SAP HANA-box.

To reduce the manual effort to deploy views, a solution is provided to further enhance service layer's query capability, with the aim to:

- Provide a more dynamic way to do a query in a secure and controllable manner
- Provide a more lightweight way to do a query than the Semantic/SQL view deployment
- Provide a more straightforward way to do a query using a limited subset of SQL, without the need to learn a new query language similar to LINQ, HQL or DBQI/DBD

4.1 Business Object Metadata

The entity `SQLQuery` is exposed in the Service Layer, with the following metadata:

```
<EntityType Name="SQLQuery">
    <Key>
        <PropertyRef Name="SqlCode"/>
    </Key>
    <Property Name="SqlCode" Nullable="false" Type="Edm.String"/>
    <Property Name="SqlName" Type="Edm.String"/>
    <Property Name="SqlText" Type="Edm.String"/>
    <Property Name="ParamList" Type="Edm.String"/>
    <Property Name="CreateDate" Type="Edm.DateTime"/>
    <Property Name="UpdateDate" Type="Edm.DateTime"/>
</EntityType>
<EntitySet EntityType="SAPB1.SQLQuery" Name="SQLQueries"/>
```

Besides the ordinary CRUD methods, an additional bounded function `List` is exposed as below, for the purpose of performing the SQL statement execution represented by this entity.

```
<FunctionImport IsBindable="true" Name="List" ReturnType="SAPB1.SQLQueryResult">
    <Parameter Name="SQLQueryParams" Type="SAPB1.SQLQuery"/>
    <Parameter Name="ParamList" Type="Edm.String"/>
</FunctionImport>

<ComplexType Name="SQLQueryParams">
```

```
    <Property Name="SqlCode" Type="Edm.String"/>
</ComplexType>
<ComplexType Name="SQLQueryResult" OpenType="true">
    <Property Name="SqlText" Type="Edm.String"/>
</ComplexType>
```

4.2 CRUD Operations

SQLQuery is allowed to perform the basic CRUD operations. The following are some examples of how to do these operations on the Microsoft SQL Server, which are very similar as on SAP HANA.

4.2.1 Create

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql04",
    "SqlName": "queryOnItem",
    "SqlText": "select ItemCode, ItemName, ItmsGrpCod from oitm"
}
```

Response

```
HTTP/1.1 201 Created
{
    "odata.metadata" : "https://server:50000/b1s/v1/$metadata#SQLQueries/@Element",
    "odata.etag" : "W/\"44486B13CDA82E54A31194A3588857803F9D1E57\"",
    "SqlCode" : "sql04",
    "SqlName" : "queryOnItem",
    "SqlText" : "select [ItemCode], [ItemName], [ItmsGrpCod] from [OITM]",
    "ParamList" : null,
    "CreateDate" : "2020-10-08",
    "UpdateDate" : "2020-10-08"
}
```

4.2.2 Retrieve by Key

Request

```
GET https://server:50000/b1s/v1/SQLQueries('sql04') HTTP/1.1
```

Response

```
HTTP/1.1 200 OK
{
    "odata.metadata" : "https://server:50000/b1s/v1/$metadata#SQLQueries/@Element",
    "odata.etag" : "W/\"44486B13CDA82E54A31194A3588857803F9D1E57\"",
    "SqlCode" : "sql04",
    "SqlName" : "queryOnItem",
    "SqlText" : "select [ItemCode], [ItemName], [ItmsGrpCod] from [OITM]",
    "ParamList" : null,
    "CreateDate" : "2020-10-08",
    "UpdateDate" : "2020-10-08"
}
```

4.2.3 Patch

Request

```
PATCH https://server:50000/b1s/v1/SQLQueries('sql04') HTTP/1.1
{
    "SqlName": "queryOnItem",
    "SqlText": "select ItemCode, ItemName from oitm"
}
```

Response

```
HTTP/1.1 204 No Content
```

4.2.4 Delete

Request

```
DELETE https://server:50000/b1s/v1/SQLQueries('sql04') HTTP/1.1
```

Response

```
HTTP/1.1 204 No Content
```

4.2.5 Retrieve All

As with the other entities, the paging mechanism takes effect in this case to avoid potential resource issues on the server side.

Request

```
GET https://server:50000/b1s/v1/SQLQueries HTTP/1.1
Prefer: odata.maxpagesize=5
Content-Type: application/json; charset=UTF-8
```

Response

```
HTTP/1.1 200 OK
Preference-Applied: odata.maxpagesize=5
Content-Type: application/json;odata=minimalmetadata;charset=utf-8
{
    "odata.metadata": "https://server:50001/b1s/v1/$metadata#SQLQueries",
    "value": [
        {
            "SqlCode": "sql01",
            "SqlName": "queryOnOrder",
            "SqlText": "select [DocEntry], [DocTotal], [DocDate], [Comments] from
[ORDR] where [DocTotal] > :docTotal",
            "ParamList": "docTotal",
            "CreateDate": "2020-10-08",
            "UpdateDate": "2020-10-08"
        },
        {
            "SqlCode": "sql02",
            "SqlName": "queryOnOrder",
            "SqlText": "select [DocEntry], [DocType] from [ORDR] t1 where not
exists(select t2.[DocEntry] from [RDR1] t2 where t1.[DocEntry] = t2.[DocEntry])",
            "ParamList": null,
            "CreateDate": "2020-10-08",
            "UpdateDate": "2020-10-08"
        },
        {
            "SqlCode": "sql03",
            "SqlName": "queryOnOrder",
            "SqlText": "select [DocEntry] from [ORDR] t1 where [DocEntry] in (select
t2.[DocEntry] from [RDR1] t2) or [DocEntry] not in (select t2.[DocEntry] from [INV1]
t2)",
            "ParamList": null,
```

```
            "CreateDate": "2020-10-08",
            "UpdateDate": "2020-10-08"
        },
        {
            "SqlCode": "sql04",
            "SqlName": "queryOnOrder",
            "SqlText": "select [DocEntry], [DocType] from [ORDR] t1 where [DocEntry]
is not null and t1.[Comments] is null",
            "ParamList": null,
            "CreateDate": "2020-10-08",
            "UpdateDate": "2020-10-08"
        },
        {
            "SqlCode": "sql06",
            "SqlName": "queryOnItem",
            "SqlText": "select [DocEntry] from [ORDR] where [CreateDate] != '2020-10-
03'",
            "ParamList": null,
            "CreateDate": "2020-10-08",
            "UpdateDate": "2020-10-08"
        }
    ],
    "odata.nextLink": "SQLQueries?$skip=5"
}
```

4.3 List Operation

The `List` function is a bounded function to run the query represented by a specific `SQLQuery`. Once a `SQLQuery` entity is created, the `List` function can be invoked in the following way with verb `GET` or `POST`:

```
GET https://server:50000/b1s/v1/SQLQueries('sql04')/List HTTP/1.1
```

```
POST https://server:50000/b1s/v1/SQLQueries('sql04')/List HTTP/1.1
```

Upon success, the service returns a JSON payload, containing the exact columns in the SQL select clause.

```
HTTP/1.1 200 OK
{
    "odata.metadata" : "https://server:50000/b1s/v1/$metadata#SAPB1.SQLQueryResult",
    "SqlText" : "select [ItemCode], [ItemName], [ItmsGrpCod] from [OITM]",
    "value" : [
        {
            "ItemCode" : "i001",
            "ItemName" : "i001",
```

```
            "ItmsGrpCod" : 100
        },
        {
            "ItemCode" : "i002",
            "ItemName" : "i002",
            "ItmsGrpCod" : 100
        },
        {
            "ItemCode" : "i003",
            "ItemName" : "i003",
            "ItmsGrpCod" : 100
        },
        {
            "ItemCode" : "i004",
            "ItemName" : "i004",
            "ItmsGrpCod" : 100
        },
        {
            "ItemCode" : "i005",
            "ItemName" : "i005",
            "ItmsGrpCod" : 100
        },
        ...
    ]
}
```

4.4 List with Paging

The paging mechanism on the server side is a MUST for the `List` function of entity `SQLQueries`, as it can protect the server resource from exhausting in case there are millions of records returned in one roundtrip, or in the case of a careless user joining multiple big tables without applying filtering conditions.

However, for the purpose of flexibility, the Service Layer allows clients to change the default paging size by specifying the following request header:

`Prefer: odata.maxpagesize=<Your preferred page size>`

For example, such a request with page size = 5,

```
GET https://server:50000/b1s/v1/SQLQueries('sql0001')/List HTTP/1.1
Prefer: odata.maxpagesize=5
Content-Type: application/json; charset=UTF-8
```

would result in the following response, in which the next page link is indicated by the field `odata.nextLink`.

```
HTTP/1.1 200 OK
Preference-Applied: odata.maxpagesize=5
Content-Type: application/json;odata=minimalmetadata;charset=utf-8
{
    "odata.metadata" : "https://server:50000/b1s/v1/$metadata#SAPB1.SQLQueryResult",
    "SqlText" : "SELECT [ItemCode], lower([ItemCode]) as [lowerItemCode],
upper([ItemCode]) as [upperItemCode] FROM [OITM]",
    "value" : [
        {
            "ItemCode" : "i001",
            "lowerItemCode" : "i001",
            "upperItemCode" : "I001"
        },
        {
            "ItemCode" : "i002",
            "lowerItemCode" : "i002",
            "upperItemCode" : "I002"
        },
        {
            "ItemCode" : "i003",
            "lowerItemCode" : "i003",
            "upperItemCode" : "I003"
        },
        {
            "ItemCode" : "i004",
            "lowerItemCode" : "i004",
            "upperItemCode" : "I004"
        },
        {
            "ItemCode" : "i005",
            "lowerItemCode" : "i005",
            "upperItemCode" : "I005"
        }
    ],
"odata.nextLink" : "SQLQueries('sql0001')/List?&$skip=5"
}
```

The underlying SQL on Microsoft SQL and SAP HANA is:

```
SELECT * FROM (SELECT [ItemCode], lower([ItemCode]) as [lowerItemCode],
upper([ItemCode]) as [upperItemCode] FROM [OITM])B1S_DUMMY_TABLE ORDER BY
CURRENT_TIMESTAMP OFFSET 0 ROWS FETCH NEXT 5 ROWS ONLY --for Microsoft SQL
```

```
SELECT "ItemCode", lower("ItemCode") as "lowerItemCode", upper("ItemCode") as
"upperItemCode" FROM "OITM" LIMIT 5 OFFSET 0 --for SAP HANA
```

4.5 Query Allowlist

Due to data sensitivity and security considerations, not all tables and columns in the company database are allowed to query.

4.5.1 Table Allowlist

Currently, only the tables in the following allowlist are accessible:

- Company Info (`CINF`)
- Administration (`OADM`)
- Administration Extension (`ADM1`)
- Print Preferences (`OADP`)
- Currencies (`OCRN`)
- Business Partner Master Data (`OCRD, OCRP, CRD1`)
- Sales Documents (`OQUT+ QUT1-14, ORDR+RDR1-14, ODLN+DLN1-14, ORRR+RRR1-14, ORDN+RDN1-14, ODPI+DPI1-14, OINV+INV1-14, ORIN+RIN1-14`)
- Purchase Documents (`OPRQ+PRQ1-14, OPQT+PQT1-14, OPOR+POR1-14, OPDN+PDN1-14, OPRR+PRR1-14, ORPD+RPD1-14, ODPO+DPO1-14, OPCH+PCH1-14, ORPC+RPC1-14`)
- Drafts (`ODRF+DRF1-14`)
- Payments (`ORCT, RCT1, RCT2, RCT3, RCT4 + OVPM, VPM1, VPM2, VPM3, VPM4`)
- Bank Codes (`ODSC`)
- House Bank Accounts (`DSC1`)
- Hidden Features (`OHFC`)
- Posting Period (`OFPR`)
- Periods Category (`OACP`)
- Branch (`OBPL`)
- Holidays (`OHLD + HLD1`)
- Warehouse and stock tables (`OITM, OITW, OIBQ, OBIN, OBTQ, OBBQ, OBTN, OSRQ, OSBQ, OSRN`)
- Activity Related Tables (`OCLG, OCLT, OCLS`)
- Attachments (`ATC1`)
- Exchange Rates (`ORTT`)
- Resource master data (`ORSC, RSC1, RSC2, RSC3, RSC4, RSC5, RSC6`)
- Route stage master data (`OSRT`)
- Bill of materials (`OITT, ITT1, ITT2`)

- Production order (`OWOR, WOR1, WOR4`)
- Resource capacity (`ORCJ`)
- Order Recommendations (`ORCM`)
- Price List and Prices (`OPLN, ITM1`)

4.5.2 Column List

By default, all columns in the allowlist tables are accessible. To further accurately control the accessibility, the column list for the above tables can also be defined.

For example, assume the list is:

```
{
    "TableList": [
        "ADM1",
        "ORDR",
        "CINF"
    ],
    "ColumnExcludeList": {
        "ORDR": [
            "CreateDate",
            "UpdateDate"
        ],
        "CINF": [
            "Algo",
            "AliasUpd",
            "TrailDays"
        ]
    },
    "ColumnIncludeList": {
        "ADM1": [
            "CurrPeriod",
            "Street"
        ]
    }
}
```

Attempt to access the columns defined in `ColumnExcludeList` in the following request,

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql06",
    "SqlName": "queryOnItem",
```

```
    "SqlText": "select Version, Algo from CINF"
}
```

would result in the following error:

```
HTTP/1.1 400 Bad Request
{
    "error" : {
        "code" : 703,
        "message" : {
            "lang" : "en-us",
            "value" : "Column 'Algo' from table 'CINF' not accessible"
        }
    }
}
```

> **Note**
>
> Internally, the table allowlist is maintained in a configuration file named `b1s_sqltable.conf` in the following folder: `<service layer installation folder>/conf/b1s_sqltable.conf`.
>
> The content is in JSON format. Partners can change the table allowlist and define the column list according to their own specific requirement. Please ensure the table names or column names are in exactly the same letter-case as they are defined in the database.

4.6 SQL Keywords

Considering the gap between the set of query keywords on SAP HANA and the Microsoft SQL Server, not all keywords are supported in the Service Layer. From the practical usage perspective, the Service Layer is designed to support the following subset of SQL keywords, which are most commonly used by partners.

SQL Keywords	Example
Select ... From ... Where	`select ItemCode, ItemName, ItmsGrpCod from oitm where 1 ItemCode > 'i01'`
Alias	`select t1.DocEntry as Col1 , t1.DocNum as Col2 from ORDR t1 where t1.DocEntry > 0`
And, Or, Not	`select t1.DocEntry, t1.DocNum from ORDR t1 where not t1.DocEntry = 1 and t1.DocNum = 1 or t1.Comments <> '1234'`
Parenthesis	`select t1.DocEntry, t1.DocNum from ORDR t1 where not (t1.DocEntry = 1 or or t1.Comments <> '1234') and t1.DocNum = 1`
Between ... And ...	`select "DocEntry" from ordr where "DocEntry" BETWEEN 1 AND 10`
Order By	`select t1.DocEntry, t1.DocNum, t1.DocTotal from ORDR t1 order by t1.DocEntry`

SQL Keywords	Example
Group By	`select DocStatus, DocType, count(*) as GroupCount from ordr group by DocStatus, DocType having count(*) > 0`
Is (Not) Null	`select DocEntry, DocType from ordr t1 where DocEntry is not null and t1.Comments is null`
Constants	`SELECT 1 as c1, 'string' 1 as c2 FROM OITM`
Like	`select CardCode from ordr where CardCode like 'c%'`
Top	`select top 2 DocStatus, DocType from ordr`
Union (All)	`select t1.LineNum from rdr1 t1 union all select t1.LineNum from inv1 t1`
In	`select DocEntry from ordr t1 where DocEntry in (select t2.DocEntry from rdr1 t2) or DocEntry not in (select t2.DocEntry from inv1 t2)`
Exists	`select DocEntry, DocType from ordr t1 where exists(select 1 from rdr1 t2 where t1.DocEntry = t2.DocEntry)`
Inner Join	`select t1.DocEntry, t2.LineNum from ordr t1 inner join rdr1 t2 on t1.DocEntry = t2.DocEntry`
Left (Outer) Join	`select t1.DocEntry, t2.LineNum from ordr t1 left join rdr1 t2 on t1.DocEntry = t2.DocEntry`
Right (Outer) Join	`select t1.DocEntry, t2.LineNum from ordr t1 right join rdr1 t2 on t1.DocEntry = t2.DocEntry`
Full (Outer) Join	`select t1.DocEntry, t2.LineNum from ordr t1 FULL OUTER JOIN rdr1 t2 on t1.DocEntry = t2.DocEntry`
Mixed Join	`select t1.DocEntry, t2.LineNum, t3.CardCode from ordr t1 inner join rdr1 t2 on t1.DocEntry = t2.DocEntry left join ocrd t3 on t1.CardCode = t3.CardCode`

4.7 SQL Functions

Currently, only the aggregation functions and a limited set of other functions are supported. More functions would be considered for support in future according to customers' requirements.

SQL Functions	
Sum, Avg	`select sum("DocTotal") as sumDocTotal, avg("DocTotal") as avgDocTotal from ordr`
Max, Min	`select min("DocTotal") as minDocTotal, max("DocTotal") as maxDocTotal from ordr`

SQL Functions	
Distinct, Count	`select count(distinct docEntry) as countDistinct, count(*)as cnt from ordr`
IfNull, IsNull	`select DocEntry, isnull(comments, 'null comments') as mssqlComments, ifnull(Comments, 'null comments') as hanaComments from ordr`
Lower, Upper	`SELECT ItemCode, lower(ItemCode) as lowerItemCode, upper(ItemCode) as upperItemCode FROM [OITM] where lower(ItemCode) = 'i001'`
Left, Right	`SELECT ItemCode, right(ItemCode, 1) as rightItemCode, LEFT(ItemCode, 1) as leftItemCode FROM oitm`

4.8 SQL Normalization

Considering the gap of SQL grammar between SAP HANA and the Microsoft SQL Server, the SQL statements running on the Microsoft SQL Server cannot necessarily work on SAP HANA, and vice versa. Consequently, in most cases, the end users are expected to compose two sets of SQL, one for the Microsoft SQL Server and the other for SAP HANA. This is not developer friendly.

To overcome this problem, the Service Layer would internally parse and normalize the raw SQL identifiers based on the underlying database, so that end users would have a unified development experience.

4.8.1 Table / Column Normalization

Due to database collation difference, in SAP Business One, SAP HANA is case-sensitive while the Microsoft SQL Server is not. This means that for an SQL statement on SAP HANA, any column needs to be double quoted to make a precise match to the predefined column. Otherwise, an invalid column name error will occur.

For example, the following SQL statements

```
select itemcode, itemName, itmsGrpCod from oitm where ItemCode > 'i01' -- raw SQL
```

would be converted to the below statements on the Microsoft SQL Server and SAP HANA respectively.

```
select [ItemCode], [ItemName], [ItmsGrpCod] from [OITM] where [ItemCode] > 'i01' -- normalized on Microsoft SQL Server
```

```
select "ItemCode", "ItemName", "ItmsGrpCod" from "OITM" where "ItemCode" > 'i01' -- normalized on SAP HANA
```

Even if the columns have already been enclosed with [] or "" in the raw SQL like below:

```
select [itemcode], [itemName], [itmsGrpCod] from [oitm] where [ItemCode] > 'i01' -- raw SQL
```

```
select "itemcode", "itemName", "itmsGrpCod" from "oitm" where "ItemCode" > 'i01' -- raw SQL
```

Service Layer is still able to do the corresponding normalization. This is mainly intended for the scenario where a SQL statement has already been normalized in the SAP HANA studio or the Microsoft SQL Server studio, and users just want to use exactly the same statement to create an SQL Query in the Service Layer.

4.8.2Alias Normalization

Both columns and tables are allowed to have an alias. While table alias normalization is not meaningful, the column alias normalization is a MUST in the SQL select clause, as it is the key name of the JSON object in the response body.

Therefore, column alias normalization is mainly to ensure that the Service Layer returns the consistent JSON content on both SAP HANA and the Microsoft SQL Server.

For example, the following SQL statements

```
select t1.DocEntry as Col1 , t1.DocNum as Col2 from ORDR t1 where t1.DocEntry > 0 -- raw SQL
```

would be converted to the following statements on the Microsoft SQL Server and SAP HANA respectively:

```
select t1.[DocEntry] as [Col1] , t1.[DocNum] as [Col2] from [ORDR] t1 where t1.[DocEntry] > 0 -- normalized on Microsoft SQL Server
```

```
select t1."DocEntry" as "Col1" , t1."DocNum" as "Col2" from "ORDR" t1 where t1."DocEntry" > 0 -- normalized on SAP HANA
```

Without alias normalization, on SAP HANA the column alias would be converted to uppercase by default. As a result, the actual generated SQL would be:

```
select t1."DocEntry" as COL1 , t1."DocNum" as COL1 from ORDR t1 where t1."DocEntry" > 0 -- uppercase alias on SAP HANA
```

Accordingly, the key name in the response JSON object would be in uppercase and thus result in inconsistent behavior compared to the output on the Microsoft SQL Server.

4.8.3Function Normalization

Sometimes, functions with different names between SAP HANA and Microsoft SQL Server have the same functionality. One typical example is `IsNull` on Microsoft SQL Server and `IfNull` on SAP HANA. This means that the Microsoft SQL Server does not support `IfNull` and SAP HANA does not support `IsNull`. However, `IfNull` is the functional equivalent to `IsNull`.

For this case, in order to provide a uniform interface for developers, the Service Layer is designed to support both, by normalizing `IsNull` to `IfNull` on SAP HANA, and `IfNull` to IsNull on Microsoft SQL Server.

```
select DocEntry, ISNULL(comments, 'null comments') as mssqlComments, IFNULL(Comments, 'null comments') as hanaComments from ordr -- raw SQL
```

```
select [DocEntry], ISNULL([Comments], 'null comments') as [mssqlComments], ISNULL([Comments], 'null comments') as [hanaComments] from [ORDR] -- normalized on Microsoft SQL Server
```

```
select "DocEntry", IFNULL("Comments", 'null comments') as "mssqlComments",
IFNULL("Comments", 'null comments') as "hanaComments" from "ORDR -- normalized on SAP
HANA
```

4.9 Query with Parameter

Like ODBC or JDBC, in the Service Layer, it is allowed to create a query with parameter placeholders, and then execute the query by specifying the parameter value. A colon mark (:) is used to define a named parameter, binding it to a specific column.

For example, send such a request to create a parametrized query:

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql01",
    "SqlName": "queryOnOrder",
    "SqlText": "select DocEntry, DocTotal, DocDate, comments from ordr where DocTotal
> :docTotal"
}
```

On success, service returns:

```
HTTP/1.1 201 Created
{
    "odata.metadata" : "https://server:50000/b1s/v1/$metadata#SQLQueries/@Element",
    "odata.etag" : "W/\"44486B13CDA82E54A31194A3588857803F9D1E57\"",
    "SqlCode" : "sql01",
    "SqlName" : "queryOnOrder",
    "SqlText" : "select [DocEntry], [DocTotal], [DocDate], [Comments] from [ORDR]
where [DocTotal] > :docTotal",
    "ParamList" : "docTotal",
    "CreateDate" : "2020-10-08",
    "UpdateDate" : "2020-10-08"
}
```

To run the query, there are two ways in the Service Layer: one is to set a payload using `POST` and the other is to specify a query parameter using `GET`.

- By POST

  ```
  POST https://server:50000/b1s/v1/SQLQueries('sql01')/List HTTP/1.1
  {
      "ParamList": "docTotal=10.1"
  }
  ```

- By GET

  ```
  GET https://server:50000/b1s/v1/SQLQueries('sql07')/List?docTotal=10.1 HTTP/1.1
  ```

The query result is:

```
HTTP/1.1 200 OK
{
    "odata.metadata" : "https://server:50000/b1s/v1/$metadata#SAPB1.SQLQueryResult",
    "SqlText" : "select [DocEntry], [DocTotal], [DocDate], [Comments] from [ORDR]
where [DocTotal] > :docTotal",
    "value" : [
        {
            "Comments" : "",
            "DocDate" : "20201007",
            "DocEntry" : 1,
            "DocTotal" : 2000.0
        },
        {
            "Comments" : "",
            "DocDate" : "20201007",
            "DocEntry" : 2,
            "DocTotal" : 2000.0
        },
        {
            "Comments" : "",
            "DocDate" : "20201007",
            "DocEntry" : 3,
            "DocTotal" : 2000.0
        },
        {
            "Comments" : "",
            "DocDate" : "20201007",
            "DocEntry" : 4,
            "DocTotal" : 2000.0
        },
        {
            "Comments" : "",
            "DocDate" : "20201007",
            "DocEntry" : 5,
            "DocTotal" : 2000.0
        }
    ],
    "odata.nextLink" : "SQLQueries('sql01')/List?docTotal=10.1&$skip=5"
}
```

If there is more than one parameter, the parameters should be separated by &, like below:

```
stringParam1='val1'&integerParam2=val2
```

4.10 Query Errors and Exceptions

To pursue the highest levels of flexibility and convenience, the Service Layer allows users to input the raw SQL, which is extremely error prone. As such, the Service Layer should be capable of handling various sorts of errors and exceptions.

The following is a summary of the error categories.

4.10.1 Table or Column not in Allowlist

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql06",
    "SqlName": "queryOnItem",
    "SqlText": "select CardCode from OCRD t1 where exists(select 1 from CRD2 t2 where
t1.CardCode = t2.CardCode)"
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error" : {
        "code" : 702,
        "message" : {
            "lang" : "en-us",
            "value" : "Table 'CRD2' not accessible"
        }
    }
}
```

4.10.2 SQL Grammar Error

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
```

```
{
    "SqlCode": "sql01",
    "SqlName": "queryOnOrder",
    "SqlText": "select DocEntry from ordr t1 order by DocEntry dsc"
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": 701,
        "message": {
            "lang": "en-us",
            "value": "Invalid SQL syntax:select DocEntry from ordr t1 order by
DocEntry dsc, line 1, character position 47, Incorrect syntax near 'dsc'."
        }
    }
}
```

4.10.3 SQL Keyword / Symbol Not Supported

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql01",
    "SqlName": "queryOnOrder",
    "SqlText": "select \"ItemCode\", length(\"ItemCode\")as lenItemCode from oitm"
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": 701,
        "message": {
            "lang": "en-us",
            "value": "Invalid SQL syntax:select \"ItemCode\", length(\"ItemCode\")as
lenItemCode from oitm, line 1, character position 19, Cannot support this function or
expression: 'length'"
```

```
        }
    }
}
```

4.10.4 Multiple Identical Aliases

As JSON does no allow multiple fields with the same name, two or more columns with the same alias in the select list are not supported.

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql0002",
    "SqlName": "query002",
    "SqlText": "select max(itemCode) as min_itemCode, min(itemCode) as min_itemCode
from OITM"
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error" : {
        "code" : 701,
        "message" : {
            "lang" : "en-us",
            "value" : "Invalid SQL syntax:select max(itemCode) as min_itemCode,
min(itemCode) as min_itemCode from OITM, line 1, character position 55, Multiple name
for column or column alias 'min_itemCode', please specify it with a different alias."
        }
    }
}
```

4.10.5 Computed Columns Without Alias

For this case, the SQL query author needs to specify a unique alias for the computed columns, so that the Service Layer would know how to format the output payload.

4.10.6 Select All Columns from One Table

In most cases, selecting all the columns from one table is not necessary, and in some situations, this could possibly compromise data sensitivity and result in performance issues. Therefore, by design, the Service Layer does not allow you to use select *.

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql01",
    "SqlName": "queryOnOrder",
    "SqlText": "select * from ORDR"
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": 701,
        "message": {
            "lang": "en-us",
            "value": "Invalid SQL syntax: select * from ORDR, line 1, character
position 7, Cannot support asterisk(*) in select list."
        }
    }
}
```

However, if the select * is in the sub query, like the below example, it is allowed.

```
select DocEntry, DocType from ordr t1 where exists (select * from rdr1 t2 where
t1.DocEntry = t2.DocEntry)
```

4.10.7 SQL DML

All the DML statements, for example, update, alter, insert, delete, etc, are not permitted to run.

Request

```
POST https://server:50000/b1s/v1/SQLQueries HTTP/1.1
{
    "SqlCode": "sql01",
    "SqlName": "queryOnOrder",
    "SqlText": "update ORDR set Comments = '1234'"
```

```
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": 701,
        "message": {
            "lang": "en-us",
            "value": "Invalid SQL syntax:, line 1, character position 0, mismatched
input 'update' expecting {SELECT, '('}"
        }
    }
}
```

4.11 Query with Permission Control

Like semantic layer views, only authorized users have the authorization to access the corresponding queries.

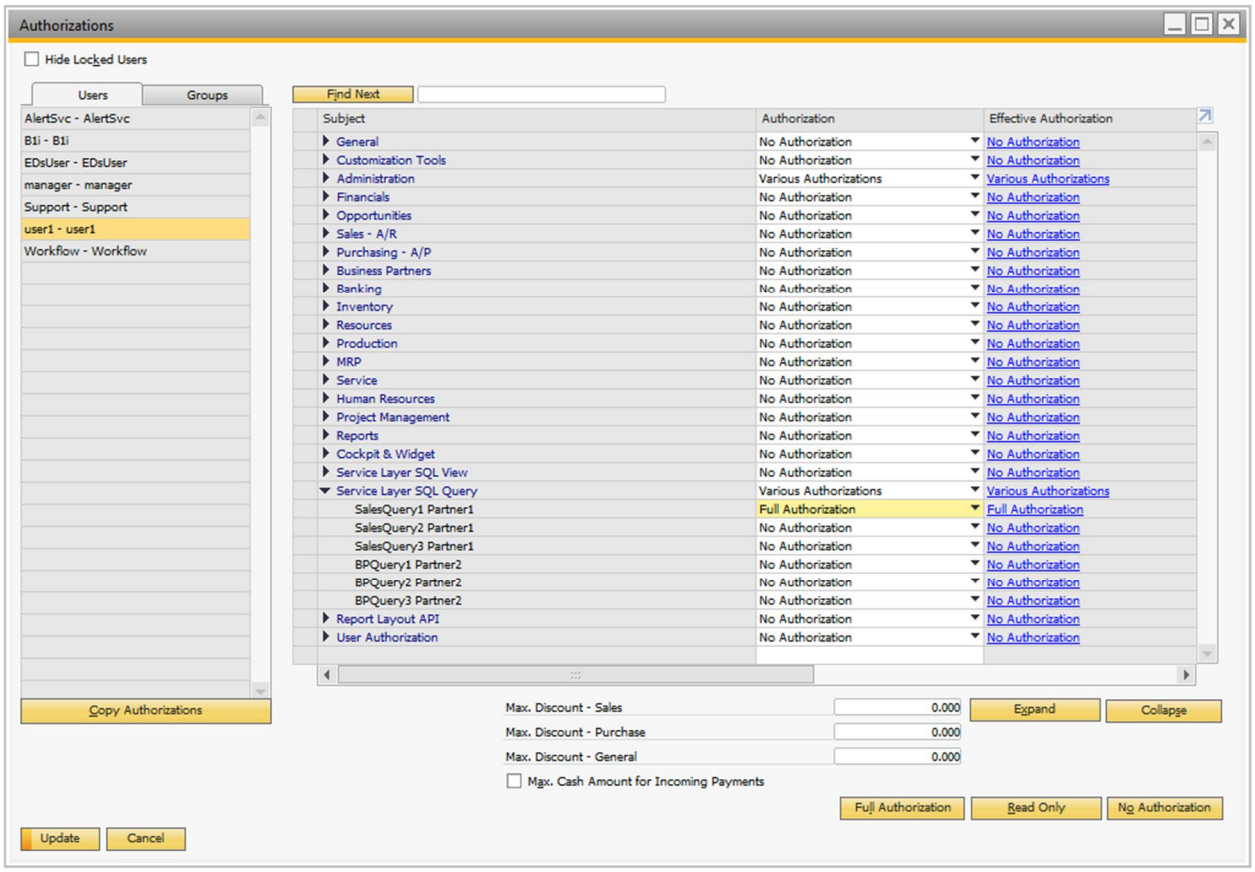
By default, a normal user has no permission to run queries. Attempting to do so would end in failure. For example, login to the Service Layer with a normal user (user1) and then send a request to invoke the `List` function on an existing SQLQueries (for example, with key = 'SalesQuery1'):

```
GET https://server:50000/b1s/v1/SQLQueries('SalesQuery1')/List HTTP/1.1
```

As expected, service returns:

```
HTTP/1.1 403 Forbidden
{
    "error": {
        "code": -6006,
        "message": {
            "lang": "en-us",
            "value": "You are not permitted to perform this action"
        }
    }
}
```

To grant permission to a normal user, log on the SAP Business One client with a super user, then open the *General Authorizations* window from the menu *Administration* -> *System Initialization* -> *Authorizations*, find the specific SQL Query from *Service Layer SQL Query* subject, and then change *No Authorization* to *Full Authorization*.



Note

- Superusers have full permission to carry out any operations on SQL queries.
- By design, the normal users do not have permission to create/delete/update SQL queries as well. However, the retrieval authorization is granted.
- For the sake of performance, the updated authorization for the normal user might not take effect immediately. To get the latest data, wait for a while (for example, one minute) to allow the internal permission cache mechanism to refresh.

4.12Security Considerations

4.12.1 SQL Injection

SQL injection would probably occur in case of parameter binding. To prevent this, internally the Service Layer validates the input parameter value by tokenizing or using the SQL statements preparation technique.

For example, there is an SQL Query with id = 'sql08' and has the following SQL text:

```
select [ItemCode], [ItemName], [ItmsGrpCod] from OITM where ItemCode = :itemCode
```

Such requests as below would results in error.

Request

```
POST https://server:50000/b1s/v1/SQLQueries('sql08')/List HTTP/1.1
{
    "ParamList": "itemCode='i001';truncate table OITM"
}
```

Response

```
HTTP/1.1 400 Bad Request
{
    "error": {
        "code": 704,
        "message": {
            "lang": "en-us",
            "value": "Parameter error."
        }
    }
}
```

4.12.2 Log SQL Query Modification

According to product standards, SQL is a type of programming code and therefore should have logs for the code change.

To track the SQL Query modification, the history can be found in table ASQL, which is the log table for the SQL Query business object.

4.12.3 Sensitive Data Accessibility

Besides business data, in the database, there is also some internally sensitive data we do not want to show to end users. To prohibit users from accessing the information, partners can define a column allowlist for each table to achieve this purpose.

4.13 Limitations or By Design

- Previously, user-defined objects (UDO) and user-defined tables (UDT) are not supported to query. As of SAP Business One 10.0 FP 2102, UDO/UDT is supported.
- LOG table (for example, AITM, ACRD) is not supported to query.

- SBOCOMMON or SBO-COMMON is not supported to query.
- Data ownership is not supported.

5 Service Layer ETag

5.1 ETag Introduction

Optimistic concurrency control is a concurrency methodology typically applied to transactional systems. It is generally used in relational database management systems with rare data update conflicts, in which transactions can complete without the expense of managing locks, leading to a higher throughput.

Although the stateless nature of HTTP makes locking infeasible for web user interfaces, HTTP does provide an alternative form of built-in optimistic concurrency control. The response to an initial `GET` request can include an ETag for subsequent `PATCH` requests to use in the `If-Match` header. Any `PATCH` requests with an out-of-date ETag in the `If-Match` header can then be rejected.

In HTTP protocol, ETag is short for entity tag and is used for identifying specific versions of a resource. It is an opaque identifier whose exact values are implementation dependent. ETag values occur in two varieties: strong and weak validation. For more information about ETag, see https://msdn.microsoft.com/library/dd541486.aspx.

In the Service Layer, we use weak validation, which guarantees the resource representation is semantically equivalent to the same ETag value. The format is as follows:

```
ETag: W/"hashed string"
```

As of SAP Business One 10.0 FP 2102, the Service Layer can support Etag mechanisms compliant with the OData spec, mainly for the purpose of avoiding blind concurrent updates on an object.

5.2 ETag Scenarios

5.2.1 ETag in Entity Creation

For this scenario, the ETag information is available in the response, reflecting the version of the newly created resource in a hashed string.

For example, send such a request to create an item:

```
POST https://hanaserver:50000/b1s/v2/Items HTTP/1.1
Content-Type: text/plain;charset=UTF-8
{"ItemCode": "i002"}
```

Upon success, the Service Layer responds with the ETag in the response header and body.

```
HTTP/1.1 201 Created
Location: https://hanaserver:50000/b1s/v2/Items('i002')
ETag: W/"356A192B7913B04C54574D18C28D46E6395428AB"

{
```

```
    "@odata.context" : "https://hanaserver:50000/b1s/v2/$metadata#Items/$entity",
    "@odata.etag" : "W/\"356A192B7913B04C54574D18C28D46E6395428AB\"",
    "ItemCode" : "i002",
    "ItemName" : null,
    ...
}
```

5.2.2 ETag in Entity Retrieval

The ETag can be applied to the entity retrieval scenario as well and allows you to get the latest resource version.

For example, send such a request to retrieve an item:

```
GET https://hanaserver:50000/b1s/v2/Items('i002') HTTP/1.1
```

Upon success, the Service Layer responds with the ETag in the response header and body.

```
HTTP/1.1 200 OK
ETag: W/"356A192B7913B04C54574D18C28D46E6395428AB"

{
    "@odata.context" : "https://hanaserver:50000/b1s/v2/$metadata#Items/$entity",
    "@odata.etag" : "W/\"356A192B7913B04C54574D18C28D46E6395428AB\"",
    "ItemCode" : "i002",
    "ItemName" : null,
    ...
}
```

5.2.3 ETag in Entity Update

The default behavior when you PATCH an entity is to directly change the entity, no matter whether the entity has already been changed by someone else. However, for most scenarios, just like the SAP Business One client, we do not want this override behavior to happen.

To achieve this, we need to specify the `If-Match` header with the value of the ETag in the request. In this way, the Service Layer would prevent the override from happening by returning an error message.

For example, assume the item with key 'i002' is changed by somebody just before you send a PATCH request:

```
PATCH https://hanaserver:50000/b1s/v2/Items('i002') HTTP/1.1
If-Match: W/"356A192B7913B04C54574D18C28D46E6395428AB"

{"ItemName": "name2"}
```

The Service Layer responds:

```
HTTP/1.1 412 Precondition Failed
Content-Type: application/json;charset=utf-8

{
    "error": {
        "code": "-2039",
        "message": "Another user or another operation modified data; to continue, open the window again (ODBC -2039)"
    }
}
```

If no changes happen, the Service Layer responds with the following, indicating that the update is successful:

```
HTTP/1.1 204 No Content
```

5.2.4 ETag in Entity Delete

The default behavior when you DELETE an entity is to directly delete the entity if the entity exists or to return a 404 error, indicating the entity is Not Found. For some scenarios, you just want to delete an entity with a specific version and prompt up an error message if the version does not exist. To achieve this, specify the `If-Match` header with the value of the ETag in the request.

For example, you send a request to get an item as follows:

```
GET https://hanaserver:50000/b1s/v2/Items('i002') HTTP/1.1
```

Upon success, the Service Layer responds:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8

{
    "@odata.context": "https://hanaserver:50000/b1s/v2/$metadata#Items/$entity",
    "@odata.etag": "W/\"356A192B7913B04C54574D18C28D46E6395428AB\"",
    "ItemCode": "i002",
    "ItemName": "i002",
    ...
}
```

Then another user happens to send a PATCH request to update this item:

```
PATCH https://hanaserver:50000/b1s/v2/Items('i002') HTTP/1.1

{
    "ItemName": "new name"
}
```

After you check some basic information with this item, you send a request to delete this item with ETag:

```
DELETE https://hanaserver:50000/b1s/v2/Items('i002') HTTP/1.1
If-Match: W/"356A192B7913B04C54574D18C28D46E6395428AB"
```

The Service Layer responds with the following error message as the entity you wish to delete has already been updated by others:

```
HTTP/1.1 412 Precondition Failed
Content-Type: application/json;charset=utf-8

{
    "error": {
        "code": "-2039",
        "message": "Another user or another operation modified data; to continue, open the window again (ODBC -2039)"
    }
}
```

5.2.5 ETag in Entity Action

The default behavior of a bindable action is to directly apply the action to the bound entity, no matter if the entity has already been changed by someone else.

However, for some scenarios, you just want to do the action on an entity with a specific version and expect the service to return an error message if the version does not exist or was updated by another. To achieve this, specify the `If-Match` header with the value of the ETag in the request.

For example, you send a request to get an order as follows:

```
GET https://hanaserver:50000/b1s/v2/Orders(8) HTTP/1.1
```

Upon success, the Service Layer responds:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8

{
    "@odata.context": "https://hanaserver:50000/b1s/v2/$metadata#Orders/$entity",
    "@odata.etag": "W/\"356A192B7913B04C54574D18C28D46E6395428AB\"",
    "DocEntry": 8,
    "DocNum": 8,
    ...

}
```

Then another user happens to send a PATCH request to update this order:

```
PATCH https://hanaserver:50000/b1s/v2/Orders(8) HTTP/1.1

{
    "Comments": "it is my order"
}
```

After you check some basic information with this order, you send a request to cancel this order with the ETag:

```
POST https://hanaserver:50000/b1s/v2/Orders(8)/Cancel HTTP/1.1
If-Match: W/"356A192B7913B04C54574D18C28D46E6395428AB"
```

The Service Layer responds with the following error message as the entity to cancel has already been updated by others:

```
HTTP/1.1 412 Precondition Failed
Content-Type: application/json;charset=utf-8

{
    "error": {
        "code": "-2039",
        "message": "Another user or another operation modified data; to continue, open the window again (ODBC -2039)"
    }
}
```

5.3 Entities with ETag

As of SAP Business One 10.0 FP 2102, the following business objects are enabled with the Etag mechanism:

- PurchaseDeliveryNotes
- CorrectionPurchaseInvoiceReversal
- Drafts
- CreditNotes
- Invoices
- GoodsReturnRequest
- PurchaseRequests
- InventoryGenEntries
- InventoryGenExits
- Orders
- DeliveryNotes
- PurchaseDownPayments

- Returns
- CorrectionInvoice
- CorrectionInvoiceReversal
- CorrectionPurchaseInvoice
- PurchaseInvoices
- PurchaseCreditNotes
- DownPayments
- PurchaseReturns
- PurchaseOrders
- ReturnRequest
- Quotations
- PurchaseQuotations
- Activities
- AdditionalExpenses
- Items
- BusinessPartners

5.4 ETag Metadata

In OData V4, the Etag is represented by an annotation in the entity set.

```
<EntitySet EntityType="SAPB1.BusinessPartner" Name="BusinessPartners">
    <Annotation Term="Org.OData.Core.V1.OptimisticConcurrency">
        <Collection>
            <PropertyPath>DataVersion</PropertyPath>
        </Collection>
    </Annotation>
    ...
</EntitySet>
```

The annotation is with the term attribute `Org.OData.Core.V1.OptimisticConcurrency`, indicating the internal concurrency mode is optimistic.

Inside the annotation, a collection of properties is presented to indicate the relevant properties used to achieve the optimistic concurrency.

However, OData V3 uses different metadata to reflect the ETag, as below.

```
<EntityType Name="BusinessPartner">
    <Key>
        <PropertyRef Name="CardCode"/>
    </Key>
    ...
    <Property ConcurrencyMode="Fixed" Name="DataVersion" Type="Edm.Int32"/>
</EntityType>
```

`ConcurrencyMode` specifies that the value of that declared property should be used for optimistic concurrency checks. Essentially, declared properties marked with a fixed `ConcurrencyMode` become part of a concurrency token.

As OData V4 is the prevalent protocol nowadays, it is strongly recommended that you use OData V4 to consume the Service Layer.

6 Configuring SAP Business One Service Layer

The installation wizard sets the common configuration options when you install the Service Layer load balancer or balancer members. The configuration options are in the configuration file `conf/b1s.conf`.

As of SAP Business One 10.0 (PL00 for SAP HANA version and PL02 for Microsoft SQL version), a configuration controller for Service Layer is available. It provides a user-friendly interface to update configuration parameters and have them take effect.

6.1 Configuration Controller for Service Layer

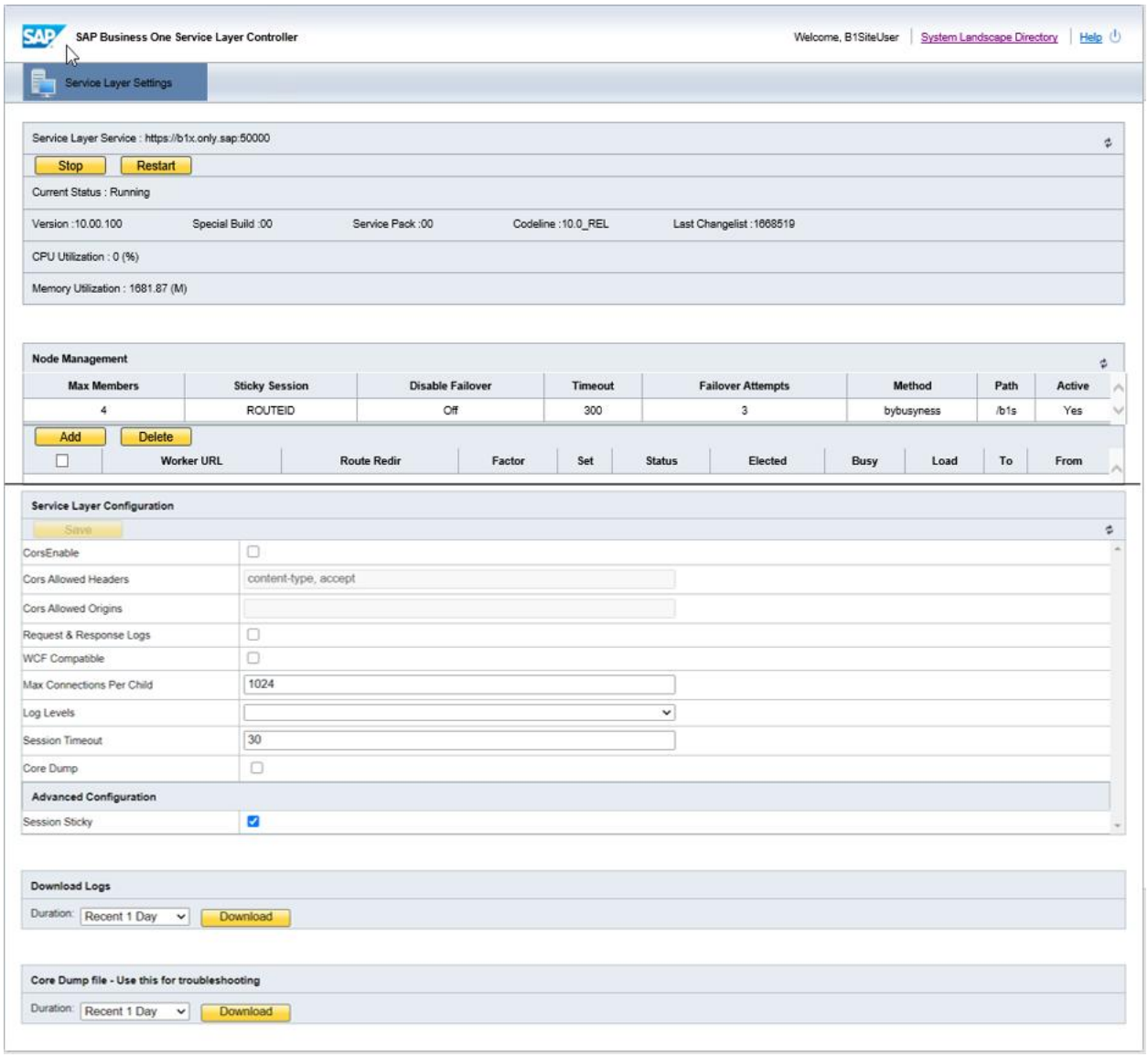
> **Note**
>
> In SAP Business One (Microsoft SQL version), before you use the controller, run the Windows PowerShell execution policy check, and make sure that the execution policy is "Unrestricted". To do so, perform the following:
>
> Open Windows PowerShell (64bit), and use the **`Get-ExecutionPolicy`** command to check the current execution policy. If it is not “Unrestricted”, use the **`Set-ExecutionPolicy Unrestricted`** command to set it.

You can access the SAP Business One Service Layer Controller after you install the Service Layer, using the URL **`https://<Server Name/IP>:<port>/ServiceLayerController`**.

In the SAP Business One Service Layer Controller, you can:

- Stop or force restart of the Service Layer service.
- Dynamically add and remove Service Layer nodes.
- Specify the configuration options to control the behavior of Service Layer. For details on the configuration options, refer to Configuration Options for Service Layer.
- Download Service Layer logs and dump files.

> **Note**
>
> Limitation: Service Layer Controller does not support managing all nodes in distribution installation mode (that is, some nodes are installed on one machine, while the load balancer is installed on another machine). This limitation applies for both the SAP HANA version and the Microsoft SQL version.



6.2 Configuration Options for Service Layer

You can also specify the configuration options to control the behavior of the service in the file <Installation Directory>/ServiceLayer/`conf/b1s.conf`. The file is in the JSON format and the options are case-sensitive. Once you save the changes, all configuration options take effect after you restart Service Layer.

> **Note**
>
> For 9.1 PL00-PL03, the file path is `<Installation Directory>/ServiceLayer/b1s/modules/b1s.conf`.
>
> The configuration file applies only to local Service Layer components. If you have installed some load balancer members on different machines from the load balancer, you must ensure a copy of the schema file exists also on each member machine.

Server Connection Options

Option	Type	Description and Default Values
`Server`	String	The database server.
`License Server`	String	This option takes effect as of 9.1 patch level 05 for working with the license server and the SLD server. The license server and the SLD server share the same address.
`SessionTimeout`	Integer	Measured by minutes. Defines the timeout period for each session. Default value is `30` (minutes).

Other Options

Option	Type	Description and Default Values
`WCFCompatible`	Boolean	Default value is `False`. If the value is set to `True`, the Microsoft WCF component can consume Service Layer. The application works around some limitations of WCF and the application behavior is as follows: · `EnumType` is replaced by `Edm.String`, since `EnumType` is not supported by WCF in metadata. · The property name cannot be the same as the type name. For example, `BatchNumber.BatchNumber` is automatically renamed to `BatchNumber.BatchNumberProperty`. · Use type `Edm.DateTime` instead of `Edm.Time`, as Microsoft .net does not have a `Time` type and uses `TimeSpan` instead, which is not compatible with SAP Business One. · As of SAP Business One 9.1, version for SAP HANA patch level 10, the `Navigation` and `Association` parts of the metadata are exposed to WCF.
`PageSize`	Integer	Defines the page size when paging is applied for a query. Default value is `20`.
`Schema`	String	Available as of 9.1 patch level 03. Default value is empty. The value is a file name under the `conf` folder, which defines the required properties for each type in metadata. For more information, see User-Defined Schemas.
`CorsEnable`	Boolean	Available as of 9.1 patch level 08. Default value is false. It functions as a switch to enable CORS (Cross Origin Resource Sharing). If this item is set to true, Service Layer will check the value of `CorsAllowedOrigins`.
`CorsAllowedOrigins`	String	Available as of 9.1 patch level 08. Default value is empty ("").

Option	Type	Description and Default Values
		This item takes effect only if `CorsEnable` is true. It is a semi-colon-separated string list where each string is a representation of a trusted origin. For example: `"CorsAllowedOrigins":"http://host1:8080;https://host2:8443"` `CorsAllowedOrigins` can also be configured as "*" to support requests from all origins. However, in production environments, it is not recommended due to security issues.
`CorsAllowedHeaders`	String	Available as of 9.2 patch level 07. Default value is ("content-type, accept"). This item takes effect only if `CorsEnable` is true. It is a comma-separated string list where each string is a representation of a request header name. For example: `"CorsAllowedHeaders":"content-type, accept, B1S-PageSize"`

6.3 Configuration by Request

Except for the connection options, Service Layer supports limiting all configuration options to the request level. You can set the Service Layer-customized HTTP header to overwrite the settings in `b1s.conf` only for the current request.

To configure your settings for the current request, you should use the following format:

`B1S-<configuration-item-name>: <value>`

For example:

- `B1S-WCFCompatible: True`
- `B1S-PageSize: 100`

> **Note**
>
> This feature is available in SAP Business One 9.1 patch level 01 and later.

7 Limitations

This section lists the limitations of SAP Business One Service Layer.

7.1 OData Protocol Implementation Limitations

In the OData protocol implementation perspective, the Service Layer has the following limitations:

- OData Version 1.0 and OData Version 2.0 are not supported.
- Request/Response of XML format is not supported for the general entity CRUD operations.
- Accessing the property of a complex type is not allowed (details in section Retrieving Individual Properties).
- Managing values and properties directly is not supported.
- OData-batch: rollback, an OData batch operation, is not supported.
- Metadata option `odata=fullmetadata` for OData version 3 is not supported.
- Metadata option `odata.metadata=full` for OData version 4 is not supported.
- `NavigationProperty` in metadata is enabled for OData Version 3 only (not enabled for OData Version 4).
- OData-query: arithmetic operators (for example, add/sub/mul/div/mod) in OData queries are not supported yet.
- OData-query: some OData query functions are not supported yet, for example, data functions, math functions, type case functions, string functions. For details of these functions, see http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html.

7.2 Functional Limitations versus SAP Business One DI API

Compared to the functionalities of SAP Business One DI API, Service Layer has the following limitations:

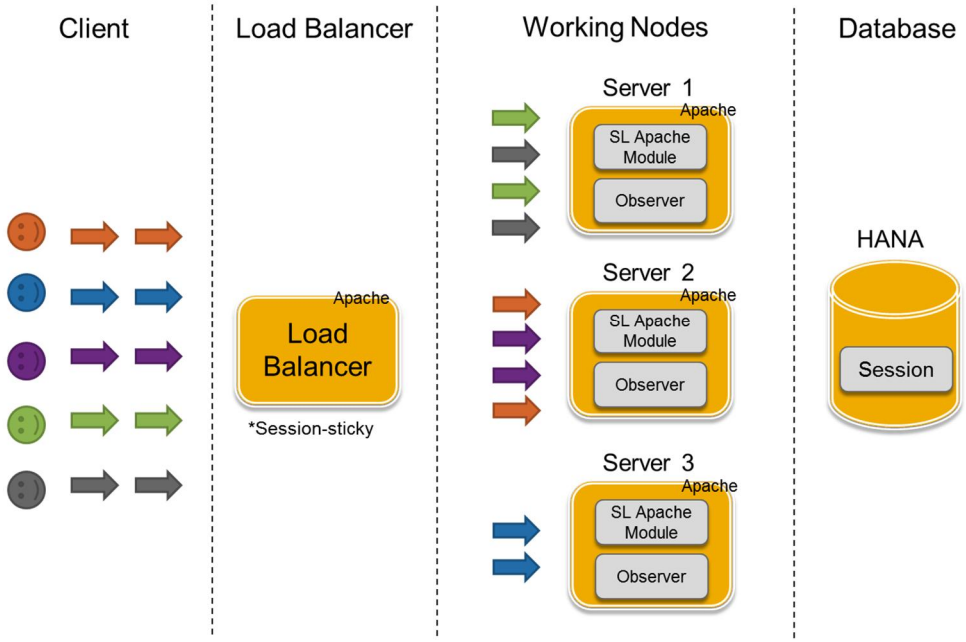
- Business object RecordSet (direct SQL) is not supported.
- Service Layer does not support the operation of ImportFromXML and ExportToXML.
- Newly created UDO/UDF/UDT is not accessible unless Service Layer is restarted.
- User transactions are not supported. There is no DI-like operation StartTransaction/EndTransaction. Transactions are internally used in each request (including OData batch request), but they cannot cross requests.

> **Note**
>
> Service layer does not support JSONP, since this feature is optional.

8 High Availability and Load Balancing

In the context of Web-based mobile-accessible applications, providing highly available services becomes increasingly important. Service Layer is well-designed and thoroughly tested to ensure that it will be continuously operational for a significant length of time in a production system.

By default, Service Layer installs Apache Multi-Processing Modules (MPMs) and is configured as a load-balancing cluster. A central load balancer distributes HTTP loads amongst its nodes according to the number of requests. In addition, Service Layer implements sticky sessions to avoid an unnecessary login, which is considered a heavy job in SAP Business One, because same session requests will always be forwarded to the same working node. Those working nodes can be deployed in a clustered system, where hardware and software redundancy helps to scale performance and provide high availability.



In an exceptional case, if the load balancer detects that one of its nodes has failed, it forwards subsequent requests to another valid node. The receiving node validates the session through the shared session info stored in the database. If valid, the receiving node automatically logs the user in, without interrupting the user actions or asking for user credentials. End-users will not notice the internal node failure, other than in a slight delay of the system response.

9 FAQ

1. What are the differences between Service Layer and other SAP Business One extension APIs, such as DI API and DI Server?

   Service Layer is built on the DI core technology, which is also the foundation of DI API and DI Server. Therefore, all three extension APIs share similar business object definitions. However, their differences are significant:

   - DI API derives from Microsoft COM technology and fits best in the Windows native environment;
   - DI Server targets SOAP-based data integration scenarios and prefers Web-services architecture;
   - Service Layer is an OData-compliant data service with a smoother learning curve, which enables easy Web Mashup, or effortless add-on development in various languages (Java, JavaScript, .NET) using 3rd-party libraries. For a list of all OData client libraries, refer to http://www.odata.org/libraries/. Service Layer is also a full-featured web application server with capabilities of high availability and scalable performance.

2. Service Layer is OData-compliant and RESTful, what are the other implications of moving to such Web-services architecture?

   Service Layer provides lightweight and faster results and simple transactions (for example, CRUD operations). Querying objects is just a matter of changing URI in a uniform fashion. Batch operation is to support advanced transaction scenarios where multiple requests need to be applied in an atomic way.

   Service Layer may not be a good choice to implement complex or distributed transactions where server-side state management is a must-have requirement.

3. Why does Service Layer support two types of update?

   The two types of update differ in the HTTP verb that is sent in the request:

   - A PUT request indicates a replacement update. All property values specified in the request body are replaced. Missing properties are set to their default values.
   - A PATCH request indicates a differential update. Only exactly those property values in the request body are replaced. Missing properties are not altered.

   In most cases, Patch request is the recommended approach to update object data.

4. What if my HTTP client does not support the `PATCH` method?

   As of 9.1 patch level 04, you can use the `POST` method to override the `PATCH` method. To do so, use the `POST` method and specify in the HTTP header `X-HTTP-Method-Override` the method to be overridden. For example, the following two requests are equal:

   - `PATCH /Orders(1)`
   - ```
     POST /Orders(1)
     X-HTTP-Method-Override: PATCH
     ```

   Note that the `POST` method can also override the `PUT`, `MERGE`, and `DELETE` methods.

5. Does the service auto start with system?

   Yes, for Service Layer running on SAP HANA. As Service Layer is installed as a series of Linux services named b1s and b1s<port>, you can check those services via this command (run as `root` user):

   ```
   # chkconfig | grep b1s
   ```

   On my environment it returns:
   ```

```
b1s                 off
b1s50000            on
b1s50001            on
b1s50002            on
b1s50003            on
sapb1servertools    on
```

b1s50000 is for the load balancer and others, such as 50001~50003, are for the 3 service nodes. "on" means the service is able to start automatically with the system.

You can turn it off for each node:

```
# chkconfig b1s50000 off
# chkconfig b1s50001 off
# chkconfig b1s50002 off
# chkconfig b1s50003 off
```

Of course you can turn on it again:

```
# chkconfig b1s50003 on
```

6. Does it work if SAP HANA client is not installed at the default location?

   By default, SAP HANA client x64 version is installed at `/usr/sap/hdbclient`. As of 9.1 patch level 04, Service layer can work even if SAP HANA client is not installed at the default location.

   Note: This is done by detecting SAP HANA client in this file: `/var/opt/.hdb/{hostname}/installations.client`

   In prior versions, please work around this issue by creating a symbol link at the default location to your new location, e.g.

   ```
   # mkdir /usr/sap
   # ln -s /your/path/hdbclient /usr/sap/hdbclient
   ```

   Or add the HANA client path in the system path: find file `/etc/ld.so.conf`, append your path at the end and run:

   ```
   # ldconfig
   ```

10 Appendix I: Service Layer versus DI API

This section explains the differences in how to invoke the APIs to finish the corresponding functionalities in SAP Business One Service Layer versus in SAP Business One DI API.

The APIs fall into different categories in terms of the operations for entity CRUD, Transaction, Query, Company Service and UDO.

10.1 CRUD APIs

We take the object Order as a typical example to illustrate how to invoke CRUD APIs.

10.1.1 Creating Entities

DI API

```
SAPbobsCOM.Company oCompany;
...
SAPbobsCOM.Documents order =
(Documents)oCompany.GetBusinessObject(BoObjectTypes.oOrders);

order.CardCode = "c001";
order.DocDate = DateTime.Today;
order.DocDueDate = DateTime.Today;

//Add items lines
order.Lines.ItemCode = "i001";
order.Lines.Quantity = 1;
order.Lines.TaxCode = "T1";
order.Lines.UnitPrice = 100;
order.Lines.Add();

//Add this newly created order
int retCode = order.Add();
```

Service Layer

```
POST /Orders
{
       "CardCode": "c001",
       "DocDate": "2014-04-01",
       "DocDueDate": "2014-04-01",
       "DocumentLines": [
              {
                     "ItemCode": "i001",
                     "UnitPrice": 100,
                     "Quantity": 1,
                     "TaxCode": "T1"
              }
       ]
}
```

10.1.2 Retrieving Entities

DI API

```
SAPbobsCOM.Documents order =
(Documents)oCompany.GetBusinessObject(BoObjectTypes.oOrders);
bool bRet = order.GetByKey(2);
if (bRet)
{
  Console.WriteLine(order.GetAsXML());
}
```

Service Layer

```
GET /Orders(2)
```

10.1.3 Updating Entities

DI API

```
SAPbobsCOM.Documents order =
(Documents)oCompany.GetBusinessObject(BoObjectTypes.oOrders);
order.GetByKey(2);
order.Comments = "New comments";
order.Update();
```

Service Layer

```
PATCH /Orders(2)
{
  "Comments":"New comments"
}
```

10.1.4 Deleting Entities

DI API

```
SAPbobsCOM.Documents order =
(Documents)oCompany.GetBusinessObject(BoObjectTypes.oOrders);
order.GetByKey(2);
order.Remove();
```

Service Layer

```
DELETE /Orders(2)
```

10.2 Company Service APIs

We take the objects GetCompanyInfo and UpdateCompanyInfo as examples to show how to invoke company service APIs.

DI API

GetCompanyInfo

```
SAPbobsCOM.CompanyService companyService = oCompany.GetCompanyService();

SAPbobsCOM.CompanyInfo companyInfo = companyService.GetCompanyInfo();
Console.WriteLine("initial: company version:{0}, company name: {1}, company name:
{2},",
   companyInfo.Version, companyInfo.CompanyName,
companyInfo.AutoCreateCustomerEqCard);
```

UpdateCompanyInfo

```
...//following the above code snippet
companyInfo.AutoCreateCustomerEqCard = BoYesNoEnum.tYES;
companyService.UpdateCompanyInfo(companyInfo);

companyInfo = companyService.GetCompanyInfo();
Console.WriteLine("updated: company version:{0}, company name: {1}, company name:
{2},",
   companyInfo.Version, companyInfo.CompanyName,
companyInfo.AutoCreateCustomerEqCard);
```

Service Layer

GetCompanyInfo

```
POST /CompanyService_GetCompanyInfo
```

UpdateCompanyInfo

```
POST /CompanyService_UpdateCompanyInfo

{
  "CompanyInfo": {
    "Version": 910160,
    "EnableExpensesManagement": "tYES",
       ...
```

```
    "AutoCreateCustomerEqCard": "tYES",
       ...
  }
}
```

> **Note**
>
> This kind of APIs in Service Layer is called `FunctionImport` or `Action` in OData terminology.

10.3 Transaction APIs

Service Layer does not explicitly provide APIs about transaction because OData protocol is stateless. However, the batch request can be posted to perform the comparable functionality.

DI API

```
oCompany.StartTransaction();

SAPbobsCOM.Items items = oCompany.GetBusinessObject(BoObjectTypes.oItems);
items.ItemCode = "item_001";
items.ItemName = "item_001_name";
items.Add();

if (items.GetByKey("item_001"))
{
       items.ItemName = "item_001_name new";
       items.Update();
       oCompany.EndTransaction(BoWfTransOpt.wf_Commit);
}
else
{
       oCompany.EndTransaction(BoWfTransOpt.wf_RollBack);
}
```

Service Layer

```
POST /$batch
```

```
Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST /b1s/v1/Items
Content-Type: application/json

{"ItemCode":"item_001", "ItemName":"item_001_name"}
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 2

PATCH /b1s/v1/Items('item_001')
Content-Type: application/json

{"ItemName":"item_001_name new"}
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

Note

The first two lines indicate that this request is a batch request and the request header should be set to 'Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b'.

The remaining part is the exact batch body.

- Multiple operations in the batch body should be enclosed in a change set so as to be treated as an atomic operation.
- The batch request does not provide a chance for clients to rollback transactions. If all operations are successful, the batch automatically performs this transaction.
- You can enclose complex business logic in one transaction via DI API, while you cannot do it via Service Layer.

For more information about batch specifications, see http://www.odata.org/documentation/odata-version-3-0/batch-processing/.

10.4 Query APIs

DI API exposes the object Recordset to execute native SQL to implement a query, while Service Layer makes use of OData query to finish the equivalent functionality.

DI API

```
SAPbobsCOM.Recordset oRecordSet =
oCompany.GetBusinessObject(BoObjectTypes.BoRecordset);
oRecordSet.DoQuery("Select \"CardCode\", \"CardName\" from OCRD where \"CardCode\" >=
'C001'");

while (!oRecordSet.EoF)
{
       Console.WriteLine("{0}={1},{2}={3}", oRecordSet.Fields.Item(0).Name,
oRecordSet.Fields.Item(0).Value,
              oRecordSet.Fields.Item(1).Name, oRecordSet.Fields.Item(1).Value);
       oRecordSet.MoveNext();
}
```

Service Layer

```
GET /BusinessPartners?$select=CardCode, CardName&$filter=CardCode ge 'C001'
{
    "odata.metadata":
"https://databaseserver:50000/b1s/v1/$metadata#BusinessPartners",
    "value": [
        {
            "CardCode": "ce7a456e-ead4-4",
            "CardName": "bb72965f-b076-4a81-859f-a2bde7b5b356"
        },
             ...
        {
            "CardCode": "ce8da8a1-d674-4",
            "CardName": "cbdb91d7-1a29-4e64-9ce8-0bd0a72704b8"
        }
    ]
}
```

Note

The response of Service Layer is in JSON format. If you want to iterate the result set as DI API does, you have to make use of OData client libraries (for example, WCF).

10.5 UDO APIs

10.5.1 Creating UDOs

You need to perform five steps to create an UDO. The creation process is very similar between DI API and Service Layer.

DI API

Step 1: Create UDT "MyOrder" as the main table

```
//Create UDT "MyOrder" as main table
SAPbobsCOM.UserTablesMD udtMyOrder =
(UserTablesMD)oCompany.GetBusinessObject(BoObjectTypes.oUserTables);
udtMyOrder.TableName = "MyOrder";
udtMyOrder.TableDescription = "MyOrderDesc";
udtMyOrder.TableType = BoUTBTableType.bott_Document;
udtMyOrder.Add();
```

Step 2: Add fields to table "MyOrder"

```
//Add UDF CustomerName to table "MyOrder"
SAPbobsCOM.UserFieldsMD udfCustomerName =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);

udfCustomerName.Name = "CustomerName";
udfCustomerName.Type = BoFieldTypes.db_Alpha;
udfCustomerName.Size = 10;
udfCustomerName.Description = "Customer name";
udfCustomerName.SubType = BoFldSubTypes.st_None;
udfCustomerName.TableName = "@MYOrder";

udfCustomerName.Add();

//Add UDF DocTotal to table "MyOrder"
```

```
SAPbobsCOM.UserFieldsMD udfDocTotal =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);
udfDocTotal.Name = "DocTotal";
udfDocTotal.Type = BoFieldTypes.db_Float;
udfDocTotal.Description = "Total amount";
udfDocTotal.SubType = BoFldSubTypes.st_Sum;
udfDocTotal.TableName = "@MYOrder";

udfDocTotal.Add();
```

Step 3: Create UDT "MyOrderLines" as the child table

```
//Create UDT "MyOrderLines" as child table
SAPbobsCOM.UserTablesMD udtMyOrderLines =
(UserTablesMD)oCompany.GetBusinessObject(BoObjectTypes.oUserTables);
udtMyOrderLines.TableName = "MyOrderLines";
udtMyOrderLines.TableDescription = "My Order lines";
udtMyOrderLines.TableType = BoUTBTableType.bott_DocumentLines;
udtMyOrderLines.Add();
```

Step 4: Add fields to table "MyOrderLines"

```
//Add UDF ItemName to table "MyOrderLines"
SAPbobsCOM.UserFieldsMD udfItemName =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);
udfItemName.Name = "ItemName";
udfItemName.Type = BoFieldTypes.db_Alpha;
udfItemName.Size = 10;
udfItemName.Description = "Item name";
udfItemName.SubType = BoFldSubTypes.st_None;
udfItemName.TableName = "@MYOrderLINES";

udfItemName.Add();

//Add UDF Price to table "MyOrderLines"
SAPbobsCOM.UserFieldsMD udfPrice =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);
udfPrice.Name = "Price";
udfPrice.Type = BoFieldTypes.db_Float;
udfPrice.Description = "Unit price";
udfPrice.SubType = BoFldSubTypes.st_Price;
udfPrice.TableName = "@MYOrderLINES";
```

```
udfPrice.Add();

//Add UDF Quantity to table "MyOrderLines"
SAPbobsCOM.UserFieldsMD udfQuantity =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);
udfQuantity.Name = "Quantity";
udfQuantity.Type = BoFieldTypes.db_Float;
udfQuantity.Description = "Quantity";
udfQuantity.SubType = BoFldSubTypes.st_Quantity;
udfQuantity.TableName = "@MYOrderLINES";

udfQuantity.Add();
```

Step 5: Register UDO "MyOrder"

```
SAPbobsCOM.UserObjectsMD udoMyOrder =
(UserObjectsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserObjectsMD);
udoMyOrder.Code = "MyOrders";
udoMyOrder.Name = "MyOrder";
udoMyOrder.TableName = "MyOrder";
udoMyOrder.ObjectType= BoUDOObjType.boud_Document;

udoMyOrder.ChildTables.TableName = "MyOrderLines";
udoMyOrder.ChildTables.ObjectName = "MyOrderLines";
udoMyOrder.ChildTables.Add();

udoMyOrder.Add();
```

Service Layer

Step 1: Create UDT "MyOrder" as the main table

```
POST /UserTablesMD

{
    "TableName": "MyOrder",
    "TableDescription": "My Orders",
    "TableType": "bott_Document"
}
```

Step 2: Add fields to table "MyOrder"

```
POST /UserFieldsMD
```

```
{
    "Name": "CustomerName",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "Customer name",
    "SubType": "st_None",
    "TableName": "@MYORDER"
}

POST /UserFieldsMD

{
    "Name": "DocTotal",
    "Type": "db_Float",
    "Description": "Total amount",
    "SubType": "st_Sum",
    "TableName": "@MYORDER"
}
```

Step 3: Create UDT "MyOrderLines" as the child table

```
POST /UserTablesMD

{
    "TableName": "MyOrderLines",
    "TableDescription": "My Order lines",
    "TableType": "bott_DocumentLines"
}
```

Step 4: Add fields to table "MyOrderLines"

```
POST /UserFieldsMD

{
    "Name": "ItemName",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "Item name",
    "SubType": "st_None",
    "TableName": "@MYORDERLINES"
}
```

```
POST /UserFieldsMD

{
    "Name": "Price",
    "Type": "db_Float",
    "Description": "Unit price",
    "SubType": "st_Price",
    "TableName": "@MYORDERLINES"
}

POST /UserFieldsMD

{
    "Name": "Quantity",
    "Type": "db_Float",
    "Description": "Quantity",
    "SubType": "st_Quantity",
    "TableName": "@MYORDERLINES"
}
```

Step 5: Register UDO "MyOrder"

```
POST /UserObjectsMD

{
    "Code": "MyOrders",
    "Name": "MyOrder",
    "TableName": "MyOrder",
    "ObjectType": "boud_Document",
    "UserObjectMD_ChildTables": [
        {
            "TableName": "MyOrderLines",
            "ObjectName": "MyOrderLines"
        }
    ]
}
```

10.5.2 CRUD and Query Operations

Once an UDO is created, you can treat it as an ordinary entity. The URL for UDO CRUD and Query operation are the same as the internal SAP Business One entities.

Creating UDO Entity

```
POST /MyOrders

{
    "U_CustomerName": "c1",
    "U_DocTotal": 620,
    "MyOrderLinesCollection": [
        {
            "U_ItemName": "item1",
            "U_Price": 100,
            "U_Quantity": 3
        },
        {
            "U_ItemName": "item2",
            "U_Price": 80,
            "U_Quantity": 4
        }
    ]
}
```

Retrieving UDO Entity

```
GET /MyOrders(10)
```

Querying on UDO Entity

```
GET /MyOrders?$select=U_CustomerName, U_DocTotal&$filter=U_CustomerName eq 'c1' and
U_DocTotal gt 1000
```

10.6 UDF APIs

This section demonstrates how to create, retrieve, update, and delete UDFs of an existing entity (for example, `BusinessPartners`), and perform relevant operations on entities with the added UDFs.

10.6.1 CRUD Operations

DI API

Creating UDFs

```
SAPbobsCOM.UserFieldsMD udf =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);

udf .Name = "u1";
udf .Type = BoFieldTypes.db_Alpha;
udf .Size = 10;
udf .Description = "udf 1";
udf .SubType = BoFldSubTypes.st_None;
udf .TableName = "OCRD";


int retCode = udf .Add();
```

> **Note**
>
> `OCRD` is the main table of `BusinessPartners`.

Retrieving UDFs

```
SAPbobsCOM.UserFieldsMD udf =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);
string tableName = "OCRD";
int fieldID = 0;//Assume the FieldID of the entity to retrieve is 0
if (udf.GetByKey(tableName, fieldID))
{
 Console.WriteLine("Name = {0}, Description = {1}", udf.Name, udf.Description);
}
```

> **Note**
>
> You can get the value of `fieldID` from the response of creating an UDF.
>
> The entity `UserFieldsMD` has to be retrieved with a multiple-field-composed key.

Updating UDFs

```
SAPbobsCOM.UserFieldsMD udf =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);
```

```
string tableName = "OCRD";
 int fieldID = 0;//Assume the FieldID of the entity to retrieve is 0
 if (udf.GetByKey(tableName, fieldID))
 {
 udf.Description = "New Description";
 udf.Update();
 }
```

Deleting UDFs

```
SAPbobsCOM.UserFieldsMD udf =
(UserFieldsMD)oCompany.GetBusinessObject(BoObjectTypes.oUserFields);

string tableName = "OCRD";
 int fieldID = 0;//Assume the FieldID of the entity to retrieve is 0.
 if (udf.GetByKey(tableName, fieldID))
 {
 udf.Remove();
 }
```

Service Layer

Creating UDFs

```
POST /UserFieldsMD
{
    "Name": "u1",
    "Type": "db_Alpha",
    "Size": 10,
    "Description": "udf 1",
    "SubType": "st_None",
    "TableName": "OCRD"
}
```

The response is:

```
HTTP/1.1 201 Created

{
    "Name": "u1",
    "Type": "db_Alpha",
    "Size": 10,
```

```
    "Description": "udf 1",
    "SubType": "st_None",
    "LinkedTable": null,
    "DefaultValue": null,
    "TableName": "OCRD",
    "FieldID": 0,
    "EditSize": 10,
    "Mandatory": "tNO",
    "LinkedUDO": null,
    "ValidValuesMD": []
}
```

Retrieving UDFs

```
GET /UserFieldsMD(TableName='OCRD', FieldID=0)
```

> **Note**
>
> You can get the value of `FieldID` from the response of creating UDF.
>
> The entity `UserFieldsMD` has to be retrieved with multiple-field-composed key.

Updating UDFs

```
PATCH /UserFieldsMD(TableName='OCRD', FieldID=0)

{
    "Description": "New Description",
}
```

Deleting UDFs

```
DELETE /UserFieldsMD(TableName='OCRD', FieldID=0)
```

10.6.2 Performing Operations on Entities with UDFs

This section shows how to perform operations on entities (`BusinessPartners`) with an added UDF (`U_u1`).

DI API

Creating entity with UDF

```
SAPbobsCOM.BusinessPartners bp =
(BusinessPartners)oCompany.GetBusinessObject(BoObjectTypes.oBusinessPartners);
bp.CardCode = "bp_001";
bp.CardName = "bp_001_name";
bp.UserFields.Fields.Item(0).Value = "udf value";

bp.Add();
```

> **Note**
>
> `bp.UserFields.Fields.Item(0)` refers to the UDF.

Retrieving entity with UDF

```
SAPbobsCOM.BusinessPartners bp =
(BusinessPartners)oCompany.GetBusinessObject(BoObjectTypes.oBusinessPartners);
bool bRet = bp.GetByKey("bp_001");
if (bRet)
{
 Console.WriteLine("CardCode = {0}, CardName = {1}", bp.CardCode, bp.CardName);
 string udfName = bp.UserFields.Fields.Item(0).Name;
 string udfValue = bp.UserFields.Fields.Item(0).Value;
 Console.WriteLine("udfName = {0}, udfValue = {1}", udfName, udfValue);
}
```

Updating entity with UDF

```
SAPbobsCOM.BusinessPartners bp =
(BusinessPartners)oCompany.GetBusinessObject(BoObjectTypes.oBusinessPartners);

bool bRet = bp.GetByKey("bp_001");
if (bRet)
{
 bp.UserFields.Fields.Item(0).Value = "new UDF value";

 bp.Update();
}
```

Service Layer

Once an UDF is created, you can treat it as an ordinary property of an entity.

Creating entity with UDF

```
POST /BusinessPartners

{
 "CardCode": "bp_001",
 "CardName": "bp_001_name",
 "U_u1": "udf value"
}
```

Querying entity with UDF

```
GET /BusinessPartners?$filter=startswith(U_u1, 'udf ')
```

Updating entity with UDF

```
PATCH /BusinessPartners('bp_001')

{
 "CardCode": "bp_001",
 "CardName": "bp_001_name",
 "U_u1": "udf value"
}
```

11 Appendix II: Metadata Naming Difference between Service Layer and DI API

Service Layer is built on top of DI Core and reuses its metadata. Actually, to follow OData protocol, Service Layer slightly modifies the metadata. The differences are reflected in the following aspects.

11.1 Collection Object Naming Difference

For Service Layer, the collection object metadata can be found by checking `/b1s/v1/$metadata` while for DI API, GetBusinessObjectXmlSchema can be invoked to retrieve the metadata.

From the table below, it can be inferred that Service Layer takes more sensible names.

Service Layer	DI API
AccountSegmentationsCategories	Categories
BPAccountReceivablePaybleCollection	BPAccountReceivablePayble
BPFiscalTaxIDCollection	BPFiscalTaxID
BPWithholdingTaxCollection	BPWithholdingTax
BillOfExchangeTransBankPages	BillOfExchangeTrans_BankPages
BillOfExchangeTransDeposits	BillOfExchangeTrans_Deposits
BillOfExchangeTransactionLines	BillOfExchangeTransaction_Lines
BudgetCostAccountingLines	BudgetCostAccounting_Lines
BudgetLines	Budget_Lines
BusinessPlaceIENumbers	IENumbers
BusinessPlaceTributaryInfos	TributaryInfos
CashFlowAssignments	PrimaryFormItems
CheckInListParams	CheckIns
DocFreightEBooksDetails	Doc_Freight_EBooks_Doc_Details
DocsInWTGroupsCollection	DocsInWTGroups
DocumentAdditionalExpenses	DocumentsAdditionalExpenses
DocumentInstallments	Document_Installments
DocumentLineAdditionalExpenses	Document_LinesAdditionalExpenses
DocumentLines	Document_Lines

Service Layer	DI API
DocumentReferences	DocumentReference
DocumentSpecialLines	Document_SpecialLines
EBooksDetails	Line_EBooks_Doc_Details
EmployeeAbsenceInfoLines	EmployeeAbsenceInfo
EmployeeEducationInfoLines	EmployeeEducationInfo
EmployeePreviousEmpoymentInfoLines	EmployeePrevEmpoymentInfo
EmployeeReviewsInfoLines	EmployeeReviewsInfo
EmployeeRolesInfoLines	EmployeeRolesInfo
EmployeeSavingsPaymentInfoLines	EmployeeSavingsPaymentInfo
FieldIDs	FormattedSearchFields
InventoryCountingDocumentReferencesCollection	InventoryCountingDocumentReferences
InventoryPostingDocumentReferencesCollection	InventoryPostingDocumentReferences
ItemBarCodeCollection	ItemBarCodes
ItemCycleCounts	ItemCycleCount
ItemDepreciationParameters	ItemDepreciationParam
ItemDistributionRules	ItemDistributionRule
ItemGroupsWarehouseInfos	ItemGroups_WarehouseInfo
ItemLocalizationInfos	LocalizationInfos
ItemPeriodControls	ItemPeriodControl
ItemPrices	Items_Prices
ItemUnitOfMeasurementCollection	ItemUnitOfMeasurement
ItemUoMPackageCollection	ItemUoMPackage
ItemWarehouseInfoCollection	ItemWarehouseInfo
JournalEntryLines	JournalEntries_Lines
LineFreightEBooksDetails	Line_Freight_EBooks_Doc_Details
MaterialRevaluationDocumentReferencesCollection	MaterialRevaluationDocumentReferences
MaterialRevaluationLines	MaterialRevaluation_lines
PaymentAccounts	Payments_Accounts
PaymentChecks	Payments_Checks
PaymentCreditCards	Payments_CreditCards
PaymentDocumentReferencesCollection	Payments_DocumentReferences

Service Layer	DI API
PaymentInvoices	Payments_Invoices
PickListsLines	PickLists_Lines
ProductTreeLines	ProductTrees_Lines
ProductTreeStages	ProductTrees_Stages
ProductionOrderLines	ProductionOrders_Lines
ProductionOrdersDocumentReferences	ProductionOrders_DocumentReferences
ProductionOrdersSalesOrderLines	ProductionOrders_SalesOrderLines
ProductionOrdersStages	ProductionOrders_Stages
ProgressiveTax_Lines	WithholdingTaxCodes_ProgressiveTax_Lines
SNBLinesCollection	SNBLines
SalesForecastLines	SalesForecast_Lines
SpecialPriceDataAreas	SpecialPricesDataAreas
SpecialPriceQuantityAreas	SpecialPricesQuantityAreas
StockTransferLines	StockTransfer_Lines
StockTransferTaxExtension	StockTransfer_TaxExtension
UserPermission	UserPermissionItem
WTGroupsCollection	WTGroups
WithholdingTaxCertificatesCollection	WithholdingTaxCertificates
WithholdingTaxDataCollection	WithholdingTaxData
WithholdingTaxDataWTXCollection	WithholdingTaxDataWTX

11.2 Business Object Naming Difference

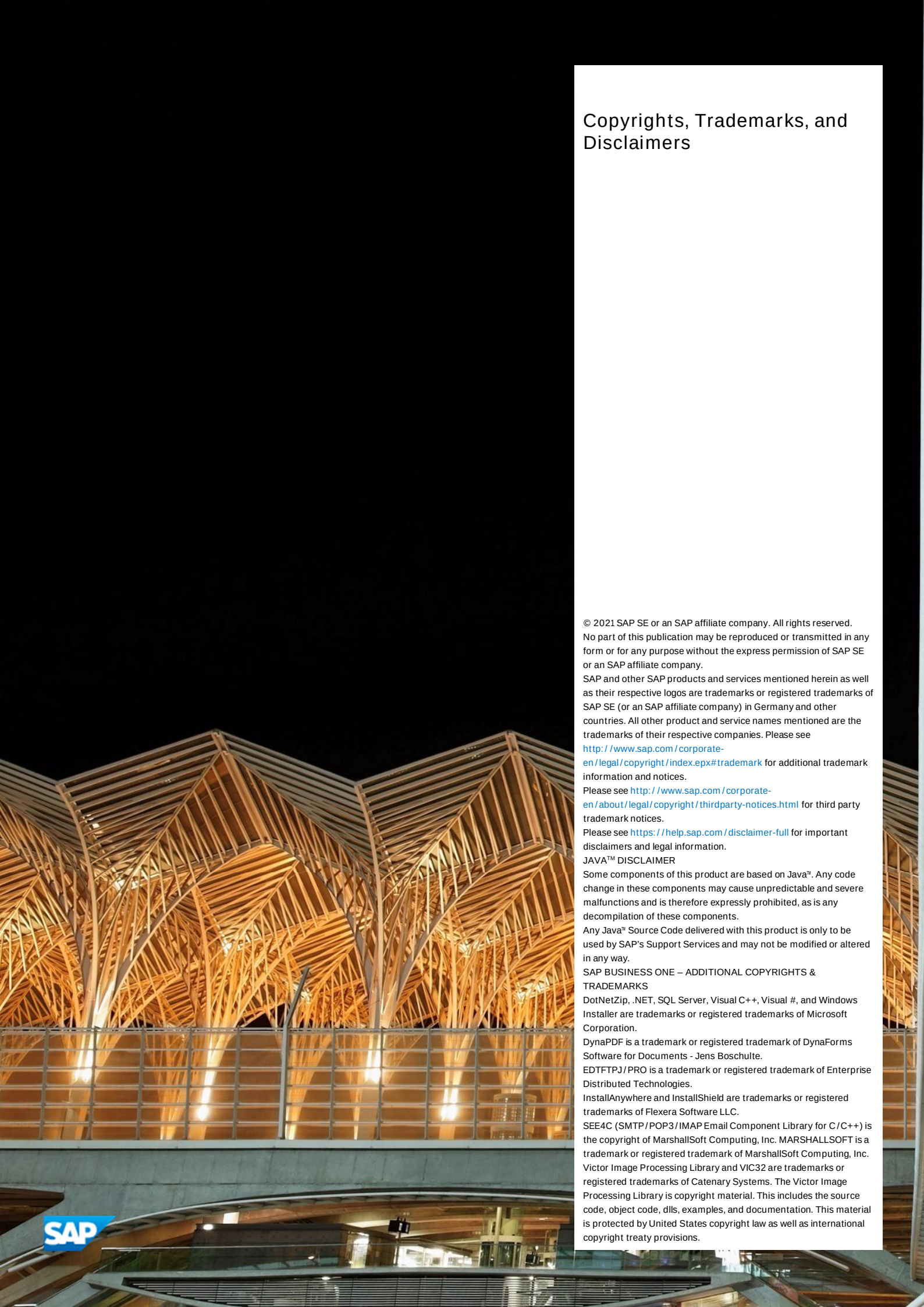
To conform to OData naming convention, Service Layer adopts plural format if the BO name is of singular format.

Service Layer	DI API
InventoryGenExits	InventoryGenExit
InventoryGenEntries	InventoryGenEntry

11.3 Property Naming Difference

To follow OData protocol, Service Layer changes a property name by appending 'Property' if it is the same as its residing object.

Service Layer	DI API
BatchNumber.BatchNumberProperty	BatchNumber.BatchNumber
Activity.ActivityProperty	Activity.Activity
PeriodCategory.PeriodCategoryProperty	PeriodCategory.PeriodCategory



Copyrights, Trademarks, and Disclaimers