

User Manual
SAP Business One 9.1, version for SAP HANA
Document Version: 1.7 – 2016-05-04

CUSTOMER

Working with SAP Business One Service Layer

All Countries

Typographic Conventions

Type Style	Description
<i>Example</i>	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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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	<ul style="list-style-type: none">• 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 Appendix II.
1.6	2016-03-04	Minor update for OBServer.
1.7	2016-05-04	<ul style="list-style-type: none">• Add Create Entity with No Content• Add Aggregation• Add Attachments• Add Item Image• Add Cross Origin Resource Sharing

Table of Contents

1	Introduction.....	7
1.1	About This Document	7
1.2	Target Audience.....	7
1.3	About SAP Business One Service Layer	7
2	Getting Started	8
2.1	System Requirements.....	8
2.2	Architecture Overview.....	8
2.3	Installing SAP Business One Service Layer.....	9
3	Consuming SAP Business One Service Layer	14
3.1	Login and Logout	14
3.1.1	Session.....	15
3.2	Metadata Document.....	16
3.3	Service Document	18
3.4	Create/Retrieve/Update/Delete (CRUD) Operations.....	19
3.4.1	Creating Entities.....	19
3.4.2	Retrieving Entities	22
3.4.3	Updating Entities	23
3.4.4	Deleting Entities	23
3.4.5	Create Entity with No Content	24
3.5	Actions.....	24
3.6	Query Options.....	30
3.6.1	Get All Entities	31
3.6.2	Get Fields of an Entity	31
3.6.3	Query Properties of the Enumeration Type.....	31
3.6.4	Query Properties of the Datetime Type	32
3.6.5	Query Properties of the Time Type	32
3.6.6	Paginate the Selected Orders	32
3.6.7	Aggregation	33
3.7	Batch Operations.....	38
3.7.1	Batch Request Method and URI.....	38
3.7.2	Batch Request Headers.....	38
3.7.3	Batch Request Body	38
3.7.4	Change Sets.....	39
3.7.5	Batch Request Sample Codes	40
3.7.6	Batch Response	41
3.8	Retrieving Individual Properties	44
3.9	Associations.....	45
3.9.1	Metadata Definitions of Associations and Navigation Properties	46
3.9.2	Retrieving navigation properties as entity	47
3.9.3	Retrieving navigation properties via \$expand	48
3.10	User-Defined Schemas	48
3.10.1	Filter Fields.....	50

3.11	User-Defined Fields (UDFs)	51
3.11.1	Managing Metadata of UDFs	51
3.11.2	CRUD Operations	54
3.12	User-defined Tables (UDTs)	55
3.12.1	Managing Metadata of UDTs	55
3.12.2	CRUD Operations	56
3.13	User-Defined Objects (UDOs)	57
3.13.1	Managing Metadata of UDOs	58
3.13.2	CRUD Operations	61
3.14	Attachments	62
3.14.1	Setting up an Attachment Folder	63
3.14.2	Uploading an Attachment	63
3.14.3	Downloading Attachments	66
3.14.4	Updating Attachment	66
3.15	Item Image	68
3.15.1	Setting up an Item Image Folder	68
3.15.2	Getting an Item Image	68
3.15.3	Updating or Uploading an Item Image	69
3.15.4	Deleting an Item Image	70
3.16	Cross Origin Resource Sharing (CORS)	70
4	Configuring SAP Business One Service Layer	71
4.1	Configuration Options for Service Layer	71
4.2	Configuration by Request	73
5	Limitations	74
5.1	OData Protocol Implementation Limitations	74
5.2	Functional Limitations versus SAP Business One DI API	74
6	High Availability and Performance	75
6.1	High Availability and Load Balancing	75
6.2	Load Test Benchmarking	76
7	FAQ	77
8	Appendix I: Service Layer versus DI API	79
8.1	CRUD APIs	79
8.1.1	Creating Entities	79
8.1.2	Retrieving Entities	80
8.1.3	Updating Entities	81
8.1.4	Deleting Entities	81
8.2	Company Service APIs	82
8.3	Transaction APIs	83
8.4	Query APIs	85
8.5	UDO APIs	86
8.5.1	Creating UDOs	86
8.5.2	CRUD and Query Operations	91
8.6	UDF APIs	91
8.6.1	CRUD Operations	92
8.6.2	Performing Operations on Entities with UDFs	94

9	Appendix II: Metadata Naming Difference between Service Layer and DI API	97
9.1	Collection Object Naming Difference	97
9.2	Business Object Naming Difference	99
9.3	Property Naming Difference.....	99

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 HTTP header `OData-Version` to switch the OData versions. For example, to use OData version 4, send the following HTTP request:

```
GET /$metadata
OData-Version: 4.0
...
```

2 Getting Started

2.1 System Requirements

SAP Business One Service Layer runs on SUSE Linux Enterprise Release 11 SP2, 64-bit edition. It is an application server built on the Apache HTTP Web server. The required database backend is SAP HANA Platform Edition 1.0 SPS 07 Rev74.

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

- An integrated mode, installing on the same SAP HANA 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

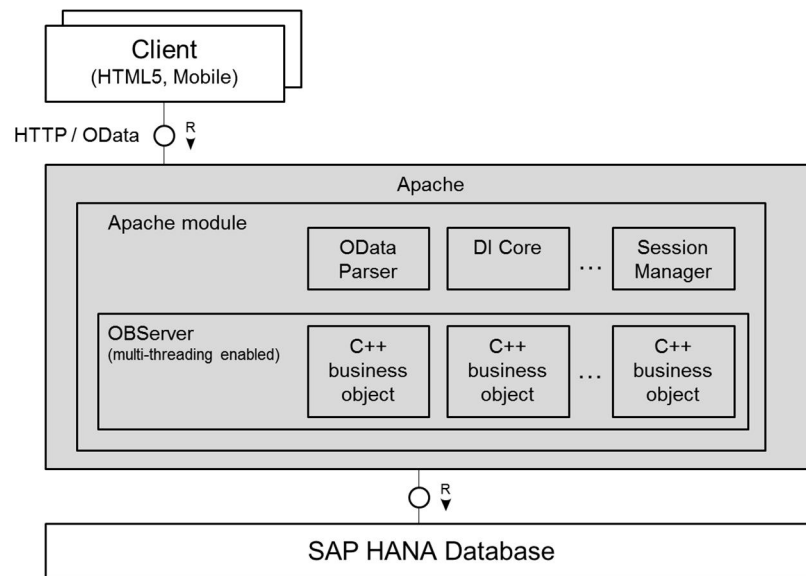
For hardware requirements, such as memory capacity or number of CPU cores, refer to SAP HANA hardware specifications.

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.

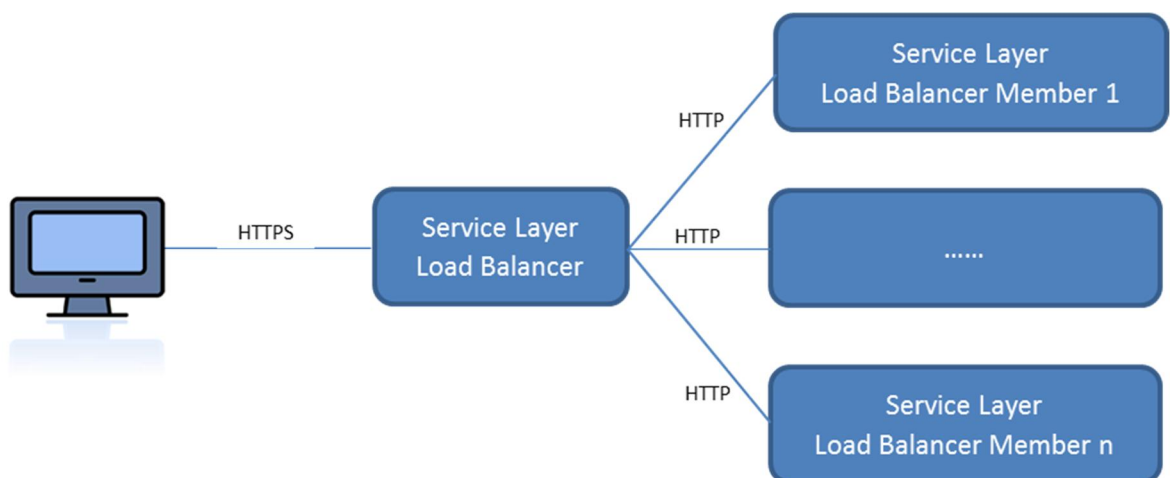


In order to achieve even higher availability and scalability, we recommend deploying multiple Service Layer instances with a load balancer in the front. The benefits include the following:

- 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 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.
- The load balancer and all load balancer members are installed on the same machine.

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.

The installation order of the load balancer and load balancer members does not affect the functioning of Service Layer. However, we recommend that you create load balancer members first because you must specify the information of load balancer members when installing the load balancer. Below is a simplified procedural description for installing the Service Layer (load balancer and load balancer members).

Prerequisites

When copying installation files to each server, ensure the following points are met:

- The following files are available:
 - RPM packages:
 - B1ServerToolsCommon
 - B1ServerToolsJava64
 - B1ServerToolsSupport
 - B1ServiceLayerApacheWebServer
 - B1ServiceLayerComponent
 - `install.bin`
- The original folder structure is kept. For example, the RPM packages must all reside in an RPM folder, separate from the binary file `install.bin`.

Procedure

1. Log on to the Linux server as `root`.
2. In a command line terminal, navigate to the directory `.../Packages.Linux/ServerComponents` where the `install.bin` script is located.
3. Start the installer from the command line by entering the following command:

```
./install.bin
```

The installation process begins.

Note

If you receive the error message “Permission denied”, you must set execution permission on the installer script to make it executable. To do so, run the following command:

```
chmod +x install.bin
```

4. In the welcome window of the setup wizard, choose the *Next* button.
5. In the *Specify Installation Folder* window, specify a folder in which you want to install Service Layer and choose the *Next* button.
6. In the *Select Features* window, select *Service Layer*.
7. In the *Specify Security Certificate* window, specify a security certificate and choose the *Next* button. You can also choose to use a self-signed certificate.
8. In the *Database Server Specification* window, specify the information of your SAP HANA database server.
9. In the *Service Layer* window, specify the following information for Service Layer and then choose the *Next* button:

- o *Install Service Layer Load Balancer*: Select the checkbox to install the load balancer.

When installing the load balancer, you need to specify the following information:

- o Port for the load balancer
- o Server name or IP address of all load balancer members, as well as their ports.
- o *Port*: Specify a port for the load balancer. Note that if the load balancer and load balancer members are installed on the same machine, each must use a different port.
- o *Service Layer Load Balancer Members*: Specify the server address and port number for each load balancer member.

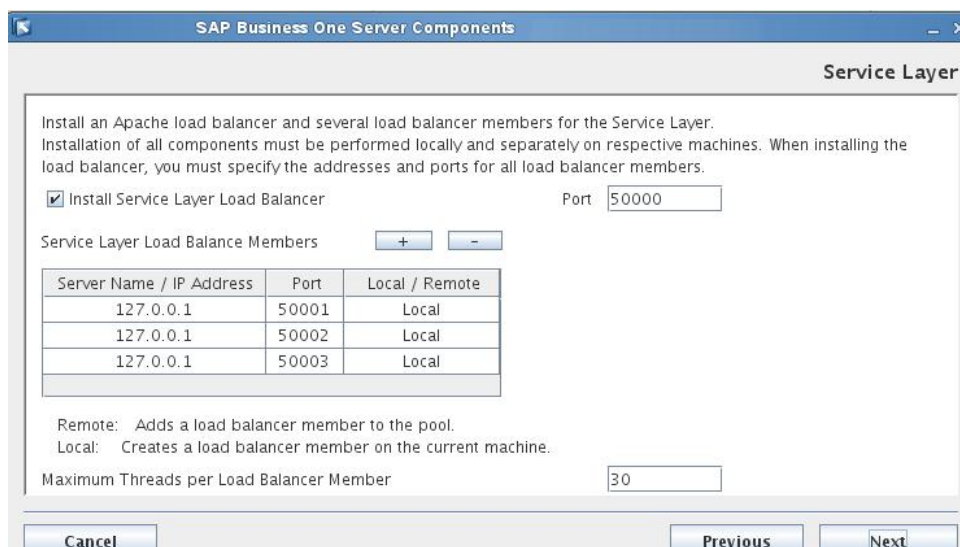
If you have selected the *Install Service Layer Load Balancer* checkbox, you can specify load balancer members either on local (current) or remote (different) machines. If on the local machine, the installer creates a local load balancer member; if on a remote machine, the load balancer member is added to the pool (cluster) of load balancer members, but you need to install the specific load balancer member on its own server.

If you have not selected the checkbox, you cannot edit the server address, which is automatically set to **127.0.0.1** (localhost). All specified load balancer members are created.

Note

IPv6 addresses are not allowed.

- o *Maximum Threads per Load Balancer Member*: Define the maximum number of threads to be run for each load balancer member.
- o *SLD Server* and *Port*: [Available as of 9.1 PL06] Specify a System Landscape Directory (SLD) server and the port for the SLD service. The Service Layer will connect to SAP Business One companies via the SLD. Note that you must specify the same SLD server for the load balancer and load balancer members if they are installed on different machines.



10. In the *Review Settings* window, review your settings and then choose the *Install* button to start the installation.
11. In the *Setup Progress* window, when the progress bar displays 100%, choose the *Next* button to finish the installation.
12. In the *Setup Process Completed* window, review the installation results and then choose the *Finish* button to exit the wizard.

Results

As of release 9.1 PL01, the Service Layer runs under the SAP Business One service user `B1service0` as all other SAP Business One services on Linux; user permissions for this user account should not be changed.

After installing Service Layer, you can check the status of each balancer member in the balancer manager. To do so, in a Web browser, navigate to <https://<Balancer Server Address>:<Port>/balancer-manager>.

To start working with Service Layer, ensure that the system database `SBOCOMMON` and your company database are installed or upgraded to the same version. In addition, the SAP HANA database user used for connection must have the following SQL object privileges:

- `SBOCOMMON`: **SELECT, INSERT, DELETE, UPDATE, EXECUTE** (all grantable)
- Company database: Full privileges

When installation is complete, the default Web browser on your server opens with links to various documentation files (for example, this user guide and the API reference). The documentation files are stored in the `<Installation Folder>/ServiceLayer/doc/` folder. In addition, you can access the API reference in a Web browser from anywhere via this URL: <https://<Load Balancer Server>:<Load Balancer Port>>. Note that only the following Web browsers are supported:

- Microsoft Internet Explorer 7 and higher
- Google Chrome
- Mozilla Firefox
- Apple Safari

As of 9.1 patch level 04, you can determine the version of the local Service Layer installation by running either of the following commands as `root`:

- `/etc/init.d/bls --version`

-
- `/etc/init.d/bls -v`

As of SAP Business One 9.1 patch level 06, the SLD server is used by Service Layer during logon. After installation, the SLD address is saved into the Service Layer configuration file (`b1s.conf`). During logon, Service Layer connects to SLD to validate the SAP Business One user and password, and get the DB credential from the SLD server.

For more information, see the *SAP Business One Administrator's Guide, version for SAP HANA* at <http://service.sap.com/smb/sbocustomer/documentation>.

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. Both HTTP and HTTPS connections are supported.	<code>http://<server>:<port>/b1s/<version></code> Example: <code>http://localhost:50000/b1s/v1</code> <code>https://<server>:<port>/b1s/<version></code> Example: <code>https://localhost:8443/b1s/v1</code>
Resource Path	Identifies the resource to be interacted with. It can be a collection of entities or a single entity.	<code>http://<server>:<port>/b1s/<version>/<resource_path></code> Example: <code>http://localhost:50000/b1s/v1/Items</code>
Query Options	Specifies multiple query options and operation parameters.	<code>http://<server>:<port>/b1s/<version>/<resource_path>?<query_options></code> Example: <code>http://localhost:50000/b1s/v1/Items?\$top=2&\$orderby=itemcode</code>
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: <ul style="list-style-type: none">• <code>POST https://localhost/b1s/v1/Login</code>• <code>POST /Login</code>
JSON Resource Representation	Represents and interacts with structured content, embedded in Service Layer requests and responses.	<code>{"key1": "value1", "arr1": [100, 200], "key2": "value2"}</code>



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>/bls/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=.node0; path=/bls
```

```
{
  "SessionId": "PTRzIjYK-weN6-1Lx1-ZG0J-3ARxfjcU0Shy",
}
```

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=.node0) 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=.node0
```

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=.node0
```

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" : -1001,
    "message" : {
```



```

        "lang" : "en-us",
        "value" : "Invalid session."
    }
}
}

```

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 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 2.1 -->
<Action IsBindable="true" Name="Close">
    <Parameter Name="Document" Type="SAPB1.Document"/>
</Action>

<!-- section 2.2 -->
<Action Name="OrdersService_Close">
    <Parameter Name="DocumentParams" Type="SAPB1.DocumentParams"/>
</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 2.1, 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`).
- In Section 2.2, you can see a global action named *OrdersService.Close* is defined. (You can use `POST /OrdersService.Close`).

Note

If you do not see the metadata sections, enable the `ExperimentalMetadata` option. You can add the option in the HTTP header:

```
GET /$metadata
BIS-ExperimentalMetadata: true
```

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.

3.3 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"
},
...
]
}

```

3.4 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).

BusinessPartners
Show/Hide | List Operations | Expand Operations

BusinessPartners('id')
Show/Hide | List Operations | Expand Operations

BusinessPartners is a business object that represents the Business Partners Master Data.

GET	BusinessPartners('id')
PUT	BusinessPartners('id')
PATCH	BusinessPartners('id')

Update a business partner by replacing exactly those property values that are specified in the request body. Missing properties will not be altered.

Example

```
PATCH https://localhost/b1s/v1/BusinessPartners('c001')
{
  "CardName": "Updated customer name"
}
```

DELETE	BusinessPartners('id')
--------	------------------------

3.4.1 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",

```

```

        ...
    }
    1,
    ...
}

```

3.4.2 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)
```


3.4.3 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.

3.4.4 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."
    }
  }
}

```

3.4.5 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.

3.5 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 (you have to set OData-Version: 4.0 in the request header to enable 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. You can set the Service Layer option `ExperimentalMetadata=true` to view all metadata.

Example

Closing an order - non-bound version

There is a hidden action named `OrdersService_Close` that you can also use to close an order as the bound action. To view the action, set `DebugLevel: 2` in HTTP header.

In metadata, you can see the definition in section 2.2:

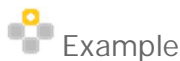
```
<!-- section 2.2 -->
<Action Name="OrdersService_Close">
  <Parameter Name="DocumentParams" Type="SAPB1.DocumentParams"/>
</Action>
```

And the first parameter `DocumentParams` is defined in section 1.3.

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

To close the order No.22, you can send the HTTP request:

```
POST /OrdersService_Close
{
  "DocumentParams": { "DocEntry": 22 }
}
```



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": "_SYS000000000081",
      "TaxCode": "",
      "LineTotal": 1000,
      "TaxTotal": 0,
    }
  ]
}
```

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


3.6 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	<p>Queries collections of entities.</p> <p>Currently supported functions for \$filter are:</p> <ul style="list-style-type: none"> startswith endswith contains substringof <p>Currently supported logical and relational operators include:</p> <ul style="list-style-type: none"> 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 <p> Note</p> <p>The operator not is supported as of 9.1 patch level 01.</p> <p>Parentheses are also supported.</p>	<pre>/Orders?\$filter=DocTotal gt 3000</pre> <pre>/Orders?\$filter=DocEntry lt 8 and (DocEntry lt 8 or DocEntry gt 116) and CardCode eq 'c1'</pre> <pre>/Orders?\$filter=DocEntry lt 8 and ((DocEntry lt 8 or DocEntry gt 116) and startswith(CardCode, 'c1'))</pre> <pre>/Items?\$filter=not (startswith(ItemName, 'item') and ForeignName eq null)</pre>
\$select	Returns the properties that are explicitly requested.	<pre>/Orders?\$select=DocEntry, DocTotal</pre>
\$orderby	Specifies the order in which entities are returned.	<pre>/Orders?\$orderby=DocTotal asc, DocEntry desc</pre>
\$top	Returns the first n (non-negative integer) records.	<pre>/Orders?\$top=3</pre>
\$skip	Specifies the result excluding the first n entities.	<pre>/Orders?\$top=3&\$skip=2</pre> <p>Where \$top and \$skip are used together, the \$skip is applied before the \$top, regardless of the order of appearance in the request.</p>
\$count	Returns the count of an entity collection.	<pre>/Orders/\$count</pre>

Option	Description	Example
		/Items/\$count?&filter=ItemCode eq 'test'
\$inlinecount	<p>Allows clients to request the number of matching resources inline with the resources in the response.</p> <p>i Note</p> <p>\$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.</p>	For more information, see the section How to use the \$inlinecount query option below.

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

3.6.1 Get All Entities

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

```
GET /Items
```

or

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

3.6.2 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
```

3.6.3 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.

3.6.4 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.

3.6.5 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.

3.6.6 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:

- o Set the configuration option `PageSize` in `conf/b1s.conf`.
- o 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`.

3.6.7 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.

3.6.7.1 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 like:

```

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

```

```

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

```

The equivalent SQL on HANA is:

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

3.6.7.2 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 /bls/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 HANA is:

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

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

On success, the response is like:

```

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

```

```

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

```

The equivalent SQL on HANA is:

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

3.6.7.4 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 HANA is:

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

3.6.7.5 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 HANA is:

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

3.6.7.6 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 /bls/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 HANA is:

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

3.6.7.7 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 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.

3.7.1 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://localhost:8443/b1s/v1/$batch`

3.7.2 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.

3.7.3 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`.

3.7.4 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:

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

3.7.5 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:
 - o Insert entity (with Content-ID = 1)
 - o Update request (with Content-ID = 2)

Sample Codes

```
POST https://localhost:443/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 /bls/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 /bls/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 /bls/v1/$1
Content-Type: application/json

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

```

3.7.6 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-5eelaaffa288
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-5eelaaffa288
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:http://10.58.81.158:9090/bls/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
```

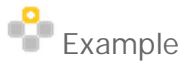
```
HTTP/1.1 204 No Content
```

```
--changesetresponse_8bfb3c36-dbf7-46a0-bdfe-670bbac86eb2--
```

```
--batchresponse_d878cedc-a0ad-4025-823e-5e1aaffa288--
```

- 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--

```

3.8 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"
    }
  }
}
```

3.9 Associations

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

3.9.1 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`.

3.9.2 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
```

3.9.3 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.

3.10 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](#).

3.10.1 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:

- o For type (EntityType or ComplexType in the metadata) Document (including all marketing document entities): DocEntry, DocNum, and DocumentLines
- o 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

```

3.11 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_".

3.11.1 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 SAP HANA 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='OCD', 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": "OCD",
      "FieldID": 0,
      ...
    }
  ]
}
```

From the response above, you can see that in addition to table OCD, 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='OCD', 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)`

3.11.2 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.",
    },
    ...
  ]
}
```

3.12 User-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".

3.12.1 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:

- o The table name uses capital letters. This name will be used when you get the metadata of this new table via GET /UserTablesMD('MYTBL').
- o 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"
}

```

3.12.2 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')
```

3.13 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 [http://localhost/b1s/v1/\\$metadata](http://localhost/b1s/v1/$metadata).

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.

3.13.1 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)
```

3.13.2 CRUD Operations

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



Example

How to create entities for a UDO

The following request creates an order with 2 item lines for the UDO "MyOrder":

```
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
    }
  ]
}
```

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.

3.14 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

3.14.1 Setting up an Attachment Folder

An attachment folder is generally a shared folder on the Windows platform for the SAP Business One client. Service Layer runs on Linux and thus 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, \\10.58.32.131\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  
"//10.58.32.131/temp/SL/attachments2/" /mnt/attachments2
```



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:

```
//10.58.32.131/temp/SL/attachments2/ /mnt/attachments2 cifs  
credentials=/etc/mycifspass 0 0
```
3. Reboot the Linux server; the Windows shared folder is automatically mounted.

3.14.2 Uploading an Attachment

Considering that the source file to attach may be on Linux or on Windows, Service Layer has to support both of these two cases.

3.14.2.1 Attach source file from Linux

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 (\\10.58.32.131\\temp\\SL\\attachments); the source file is saved there as well.

3.14.2.2 Attach source file from Windows

One way to add an attachment on Windows 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 /bls/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

Bl Service Layer (SL) is a new generation of extension API for consuming Bl 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://10.58.136.174:50000/bls/v1/$metadata#Attachments2/@Element",
  "AbsoluteEntry" : "3",
  "Attachments2_Lines" : [
    {
      "SourcePath" : "/tmp/sap_bl_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"
  }
]
}

```

3.14.3 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
```

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'
```

3.14.4 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'



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'

Note that from the business logic perspective, it is not allowed to delete an attachment or attachment line.

3.15 Item 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>
```

3.15.1 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, `\\10.58.32.131\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
"/10.58.32.131/temp/SL/itemimages" /mnt/itemimages
```

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

3.15.2 Getting an Item Image

From the SAP Business One client, you can specify an item image for an item. To get the item image via Service Layer, send a request such as:

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

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://10.58.136.174:50000/b1s/v1/$metadata#ItemImages/@Element",
  "odata.mediaReadLink": "ItemImages('i001')/$value",
}
```



```

    "odata.mediaContentType": "image/jpeg",
    "ItemCode": "'i001'",
    "Picture": "sap_hana_1.jpg"
  }

```

3.15.3 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_hana_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
```

Note

For test purposes only, you can use the Chrome plug-in `POSTMAN` to update an item image.

3.15.4 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 `Picture` of the entity `Items` instead.

3.16 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>

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://pvgd50839753a:8080;http://pvgd50839753a:8443"
```

You can refer to [Configuration Options for Service Layer](#) for more details about the CORS configurations.

4 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/bls.conf`.

4.1 Configuration Options for Service Layer

You can specify the configuration options to control the behavior of the service in the file `<Installation Directory>/ServiceLayer/conf/bls.conf`. The file is in the JSON format and the options are case-sensitive. Once you save the changes, all configuration options take effect immediately.

Note

For 9.1 PL00-PL03, the file path is `<Installation Directory>/ServiceLayer/bls/modules/bls.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
<i>Server</i>	String	The SAP HANA instance. Default value is 127.0.0.1:30015.
<i>DbUserName</i> <i>DbPassword</i>	String	The user name and password for the SAP HANA database instance. Default value is: <code>SYSTEM</code> and <code>manager</code> respectively. As of 9.1 patch level 06, these two options are optional and NOT recommended to set, because Service Layer can get them from the SLD server. They are used only when the SLD server does not work.
<i>License Server</i>	String	Default value is 127.0.0.1:40000. 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.
<i>SessionTimeout</i>	Integer	Measured by minutes. Defines the timeout period for each session. Default value is 30 (minutes).

Other Options

Option	Type	Description and Default Values
<i>ExperimentalMetadata</i>	Boolean	<p>Default value is <code>False</code>.</p> <p>By default, only a subset of entities and actions is exposed.</p> <p>If this option is <code>True</code>, the metadata returns the full set of entities and actions. However, we do not recommend that you set this option to <code>True</code>, for it has not been tested yet. We cannot guarantee that the feature currently works exactly as intended and will not be changed in the future.</p>
<i>WCFCompatible</i>	Boolean	<p>Default value is <code>False</code>.</p> <p>If the value is set to <code>True</code>, the Microsoft WCF component can consume Service Layer. The application works around some limitations of WCF and the application behavior is as follows:</p> <ul style="list-style-type: none"> • <code>EnumType</code> is replaced by <code>Edm.String</code>, since <code>EnumType</code> is not supported by WCF in metadata. • The property name cannot be the same as the type name. For example, <code>BatchNumber.BatchNumber</code> is automatically renamed to <code>BatchNumber.BatchNumberProperty</code>. • Use type <code>Edm.DateTime</code> instead of <code>Edm.Time</code>, as Microsoft .net does not have a <code>Time</code> type and uses <code>TimeSpan</code> instead, which is not compatible with SAP Business One. • As of SAP Business One 9.1, version for SAP HANA patch level 10, the <code>Navigation</code> and <code>Association</code> parts of the metadata are exposed to WCF.
<i>MetadataWithoutSession</i>	Boolean	<p>Default value is <code>False</code>.</p> <p>To get metadata, you must first log in and then request metadata in the same session, which means the HTTP request must contain the <code>B1SESSION</code> cookie item returned by the login request.</p> <p>If this option is set to <code>True</code>, you can get metadata anywhere after you log in to the service. Internally, Service Layer takes one of the existing sessions for the metadata request.</p>
<i>PageSize</i>	Integer	<p>Defines the page size when paging is applied for a query.</p> <p>Default value is 20.</p>
<i>Schema</i>	String	<p>Available as of 9.1 patch level 03.</p> <p>Default value is empty.</p> <p>Specifies a default user-defined schema. Unless you specify another user-defined schema in a request, the default schema is used for data output.</p> <p>Note that you must create a corresponding schema file to have this option take effect. For more information, see User-Defined Schemas.</p>
<i>CorsEnable</i>	Boolean	<p>Available as of 9.1 patch level 08.</p> <p>Default value is false. It functions as a switch to enable CORS (Cross Origin Resource Sharing).</p>

Option	Type	Description and Default Values
		If this item is set to true, Service Layer will check the value of <i>CorsAllowedOrigins</i> .
<i>CorsAllowedOrigins</i>	String	<p>Available as of 9.1 patch level 08.</p> <p>Default value is empty ("").</p> <p>This item takes effect only if <i>CorsEnable</i> is true. It is a semi-colon separated string list where each string is a representation of a trusted origin. For example:</p> <pre>"CorsAllowedOrigins" : "http://pvgd50839753a:8080;http://pvgd50839753a:8443"</pre> <p>CorsAllowedOrigins can also be configured as "*" to support requests from all origins. However, in production environments, it is not recommended due to security issues.</p>

4.2 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.

5 Limitations

This section lists the limitations of SAP Business One Service Layer.

5.1 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>.

5.2 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

The following features are optional, thus not required for Service Layer to support:

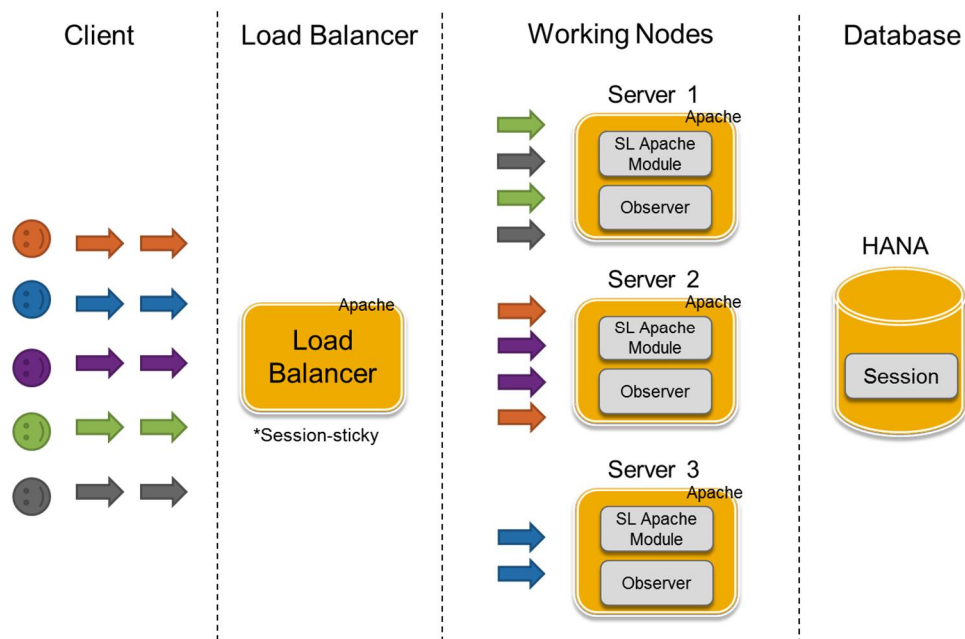
- Service layer does not support JSONP.
- Service layer does not support CORS.

6 High Availability and Performance

6.1 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.

6.2 Load Test Benchmarking

Performance testing was conducted for the purpose of determining concurrent transaction capacity with the condition of reasonable response time for 90% of all transactions. Testing was done on a standard installed environment with a set of configuration options, including CPU core numbers used by SAP HANA, CPU core numbers used by Service Layer, number of concurrent requests, maximum working thread number in each Service Layer instance, and so on. Concurrent testing began with user login and comprised creating an order with 20 lines, copying the order to a delivery, and finally copying the delivery to an invoice. One thing to note is that, although the three documents were posted separately, all were counted as one transaction when calculating the transactions per second (tps). The correctness of posted transactions was checked after the test in order not to affect the throughput test result. Moreover, the SAP HANA log was saved to a fast-IO disk, such as a solid state disk (SSD), to maximize the performance of SAP HANA.

No	SAP HANA CPU Cores	Service Layer CPU Cores	Number of Concurrent Requests	Concurrent Throughput (tps)	Average Response Time in Seconds (Order/Delivery/Invoice)
1	32	10	60	2.6	3.9/18.4/8.0
2	32	10	40	2.7	3.2/10.1/6.3
3	32	10	20	0.9	4.2/9.4/6.9
4	32	4	40	1.5	4.2/10.6/7.1
5	32	2	40	0.9	7.8/18.9/13.9
6	80	15	60	6.7	1.3/3.6/3.1

According to the above test data, the highest concurrent throughput reaches 6.7 with a good response time on each kind of document. The final test result also shows that concurrent throughput is mainly affected by the following factors:

- Increasing the number of concurrent requests will gradually increase tps, but will reach the limit at 40 concurrent requests, where the benefit begins to diminish and response time become worse.
- Increasing Service Layer CPU cores will improve tps, for example, if you increase CPU cores from 4 to 10, the tps roughly doubles.
- Adding more CPU cores to Service Layer will eventually overwhelm the SAP HANA server; therefore, a larger multi-processor SAP HANA server is recommended. For example, in the last row, after increasing SAP HANA CPU cores from 32 to 80, the concurrent throughput increases significantly from 2.7 to 6.7.

7 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:

- o DI API derives from Microsoft COM technology and fits best in the Windows native environment;
- o DI Server targets SOAP-based data integration scenarios and prefers Web-services architecture;
- o 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:

- o 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.
- o 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. How do I get metadata anonymously for third-party development?

In some cases, you may require anonymous login to Service Layer. For example, you use Microsoft WCF libraries to consume Service Layer; however, you cannot import Service Layer into Microsoft Visual Studio .NET as an external web service because retrieval of Service Layer metadata requires login. To work around the login, do the following:

1. Set the configuration option `MetadataWithoutSession` to `true`. For more information, see [Configuration Options for Service Layer](#).
2. Log in to the service node from where you want to get metadata. For example, if you install a load balancer member on a machine separate from the load balancer and you want to get metadata from that member, you will need to specify the address of the member instead of the load balancer:

```
POST http://<IP Address>:<Port>/b1s/v1/Login
```
3. Get metadata from the logged-in member anonymously:

```
GET http://<IP Address>:<Port>b1s/v1/$metadata
```

5. 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:

- o `PATCH /Orders(1)`
- o `POST /Orders(1)`
`X-HTTP-Method-Override: PATCH`

Note that the `POST` method can also override the `PUT`, `MERGE`, and `DELETE` methods.

8 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.

8.1 CRUD APIs

We take the object Order as a typical example to illustrate how to invoke CRUD APIs.

8.1.1 Creating Entities

DI API

```
SAPbobsCOM.Company oCompany;
...
SAPbobsCOM.Documents order =
(Document)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"
    }
  ]
}
```

8.1.2 Retrieving Entities

DI API

```
SAPbobsCOM.Documents order =
(Document)oCompany.GetBusinessObject(BoObjectTypes.oOrders);
bool bRet = order.GetByKey(2);
if (bRet)
{
  Console.WriteLine(order.GetAsXML());
}
```

Service Layer

GET /Orders(2)

8.1.3 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"  
}
```

8.1.4 Deleting Entities

DI API

```
SAPbobsCOM.Documents order =  
(Documents)oCompany.GetBusinessObject(BoObjectTypes.oOrders);  
order.GetByKey(2);  
order.Remove();
```

Service Layer

```
DELETE /Orders(2)
```

8.2 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.

8.3 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 /bls/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 /bls/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/>.

8.4 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": "http://localhost:805/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).

8.5 UDO APIs

8.5.1 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= BoUDObjType.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"
    }
  ]
}
```

8.5.2 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

8.6 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.

8.6.1 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)
```

8.6.2 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"
}
```

9 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.

9.1 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
BillOfExchangeTransactionLines	BillOfExchangeTransaction_Lines
BillOfExchangeTransBankPages	BillOfExchangeTrans_BankPages
BillOfExchangeTransDeposits	BillOfExchangeTrans_Deposits
BPAccountReceivablePayableCollection	BPAccountReceivablePayable
BPFiscalTaxIDCollection	BPFiscalTaxID
BPWithholdingTaxCollection	BPWithholdingTax
BudgetCostAccountingLines	BudgetCostAccounting_Lines
BudgetLines	Budget_Lines
BusinessPlaceIENumbers	IENumbers
BusinessPlaceTributaryInfos	TributaryInfos
CashFlowAssignments	PrimaryFormItems
DocumentAdditionalExpenses	DocumentsAdditionalExpenses
DocumentInstallments	Document_Installments
DocumentLineAdditionalExpenses	Document_LinesAdditionalExpenses
DocumentLines	Document_Lines
DocumentSpecialLines	Document_SpecialLines
EmployeeAbsenceInfoLines	EmployeeAbsenceInfo
EmployeeEducationInfoLines	EmployeeEducationInfo

Service Layer	DI API
EmployeePreviousEmploymentInfoLines	EmployeePrevEmploymentInfo
EmployeeReviewsInfoLines	EmployeeReviewsInfo
EmployeeRolesInfoLines	EmployeeRolesInfo
EmployeeSavingsPaymentInfoLines	EmployeeSavingsPaymentInfo
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
MaterialRevaluationLines	MaterialRevaluation_lines
PaymentAccounts	Payments_Accounts
PaymentChecks	Payments_Checks
PaymentCreditCards	Payments_CreditCards
PaymentInvoices	Payments_Invoices
PickListsLines	PickLists_Lines
ProductionOrderLines	ProductionOrders_Lines
ProductionOrdersSalesOrderLines	ProductionOrders_SalesOrderLines
ProductTreeLines	ProductTrees_Lines
ProgressiveTax_Lines	WithholdingTaxCodes_ProgressiveTax_Lines
SalesForecastLines	SalesForecast_Lines
SNBLinesCollection	SNBLines
SpecialPriceDataAreas	SpecialPricesDataAreas
SpecialPriceQuantityAreas	SpecialPricesQuantityAreas
StockTransferLines	StockTransfer_Lines

Service Layer	DI API
UserPermission	UserPermissionItem
WithholdingTaxDataCollection	WithholdingTaxData

9.2 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

9.3 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

© 2016 SAP SE or an SAP affiliate company. All rights reserved. No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. Please see

<http://www.sap.com/corporate-en/legal/copyright/index.epx#trademark> for additional trademark information and notices.

Please see <http://www.sap.com/corporate-en/about/legal/copyright/thirdparty-notices.html> for third party trademark notices.

Please see <https://help.sap.com/disclaimer-full> for important disclaimers and legal information.

JAVA™ DISCLAIMER

Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressly prohibited, as is any decompilation of these components.

Any Java™ Source Code delivered with this product is only to be used by SAP's Support Services and may not be modified or altered in any way.

SAP BUSINESS ONE – ADDITIONAL COPYRIGHTS & TRADEMARKS

DotNetZip, .NET, SQL Server, Visual C++, Visual #, and Windows Installer are trademarks or registered trademarks of Microsoft Corporation.

DynaPDF is a trademark or registered trademark of DynaForms Software for Documents - Jens Boschulte.

EDTFTPJ/PRO is a trademark or registered trademark of Enterprise Distributed Technologies.

InstallAnywhere and InstallShield are trademarks or registered trademarks of Flexera Software LLC.

SEE4C (SMTP/POP3/IMAP Email Component Library for C/C++) is the copyright of MarshallSoft Computing, Inc. MARSHALLSOFT is a trademark or registered trademark of MarshallSoft Computing, Inc. Victor Image Processing Library and VIC32 are trademarks or registered trademarks of Catenary Systems. The Victor Image Processing Library is copyright material. This includes the source code, object code, DLLs, examples, and documentation. This material is protected by United States copyright law as well as international copyright treaty provisions.