New in: 0.0 | Modified in: | Obsolete in: | Excised in:

Categorization

Entity Type
Symbol

Paclet Name
DBAPI

Context
DBAPI`

URI
DBAPI/ref/ODBapi

Keywords

Syntax Templates

Details

Lead
Athanassios I. Hatzis

Developers
Athanassios I. Hatzis

Authors
Athanassios I. Hatzis

Feature Name
XXXX

QA
XXXX

DA
XXXX

Docs
XXXX

Features Page Notes
XXXX

Comments
XXXX

ODBapi

`ODBapi[com → "add<Command>", options]`
Uses the POST method of the HTTP protocol for both reading and writing the database

`ODBapi[com → "del<Command>", options ]`
Uses either the POST or the DELETE method of the HTTP protocol to destructively alter the database

`ODBapi[com → "upd<Command>", options]`
Uses either the POST HTTP method to update record values with SQL UPDATE command or the PUT and PATCH methods of the HTTP protocol to update OrientDB structured Document records

`ODBapi[com → "get<Command>", options ]`
Uses the GET method of the HTTP protocol to retrieve values from the database. Operations are idempotent, i.e. they do not alter the database

`ODBapi[com → "impDatabase/expDatabase", options ]`
Export a gzip file that contains the database JSON export using the GET method. Import a database from an uploaded JSON text file.

`ODBapi[com → "login/logout", options ]`

Uses the GET method of the HTTP protocol to connect to a remote server using basic authentication and the same method for disconnecting

Details and Options

URL Construction

The following options are used in the construction of the HTTP request.
The OrientDB RESTful API uses the same syntax for all HTTP methods.

Syntax: http://<server>:<port>/<command>/[<database>/<arguments>]

Default options for the OrientDB server → "localhost" and the port → "2480"

Default option for the **<database>**, db → ""

Default option for the **<command>,** com → ""

Default options for **<arguments>**. These are controlled with the arg → "" option.
This option value is dependent on the following options :

dbtype → "" option is used in addDatabase command
and typical values are "plocal/graph", "plocal/document"

class → "" option is used in most of the commands

propnam → "", propval → "", and proptype → "" options are used in :
commands addProperty, updProperty, addInstance, addIndex, updValues, getRecords, ...

id → "", with this option we pass the record id, OrientDB **@rid**. It is used in :
updValues, updRecord, delRecords, and getRecords, commands.

con → "", with this option we pass the connection id used in :
delConnection command

Control of the URLFetch command

Each ODBapi command is executed through URLFetch and by default it returns
from OrientDB server the contents as a JSON string and a status code.

Default option for the HTTP method to use for request, method → "POST"

Default option for parameters to be sent for the http request, param → {}
URL parameters are not used in OrientDB API.

Default option for the contents of message body to be sent, body → ""
The body option is used indirectly to pass the sql script and sql commands.
The following options are used to constuct the message body:

sql → "" option is used in addOSQL and getOSQL

class → "" option is used in :
addDocument, addContent, addValues, addClass, addProperty, updProperty,
delClass, delRecords, delProperty, delVALUES, getClass

class → "" and superclass → "" options are used in addClass command

keys → "" and values → "" options are used in addVALUES command
These options are formatted according to SQL-92 syntax (see an example)

from → "" and to → "" options are used in creating edges with content, addCONTENT command.

construct → "" is used in addContent, addInstance and delRecords command

record → "" option takes a JSON string and is used in
addRecord, addContent, addEdge, and updRecord

ver → -1 option controls the version of the record to update, it is used in
updRecord

attribnam → "", attribval → "" options are used in updDatabase, updClass, updProperty

all → False, option is used in delRecords and updValues

uniq → False options in addInstance

indexnam → "", indextype → "" options in addIndex

mode->"COMMAND" in addOSQL

Default options for connecting to the server with usr → "admin" and pwd → "admin"

Enumerated Options

debug	True/False If True, prints message body of the http request
com	addOSQL addDatabase, addClass, addClassViaHTTP, addProperty, addPropertyViaHTTP, addRecord, addContent, addValues, addInstance, addIndex, addEdge, delDatabase, delClass, delProperty, delRecords, delValues, delConnection, updDatabase, updClass, updProperty, updValues, updRecord, getOSQL, getServer, getDatabases getClass, getRecords, impDatabase, expDatabase, logout, login
construct	DOCUMENT, VERTEX, EDGE
method	PUT, GET, DELETE, PUT, PATCH
mode	COMMAND, BATCH

The Input Assistant of the Wolfram Predictive Interface offers context-sensitive autocompletion for the enumerated option arguments of `ODBapi`. A drop-down list with option values is filtered automatically as you start typing the name of the option value e.g. com → log... and only two option values are displayed logout and login. If you want the drop-down list filtering to appear do not start quoting the option value. You can also select any of the ODBapi function templates to assist you in the completion of the command.

The auto-completion rules are automatically loaded on Front End from **OptionValues folder** at:

```
In[1]:= FileNameJoin[{$InstallationDirectory, "SystemFiles", "FrontEnd",
         "SystemResources", "FunctionalFrequency", "OptionValues"}]

Out[1]= C:\Mathematica\SystemFiles\FrontEnd\SystemResources\FunctionalFrequency\OptionValues
```

Therefore to enable auto-completion for the ODBapi function search for the file "ODBapi.m". If you installed the package under the $UserBaseDirectory then it is located at:

```
In[45]:= $UserBaseDirectory <> "\\Applications\\DBAPI\\Options\\ODBapi.m"

Out[45]= C:\Users\athanassios\AppData\Roaming\Mathematica\Applications\DBAPI\Options\ODBapi.m
```

Copy that file inside the **OptionValues folder** of your *Mathematica* installation.

Change Default Options

Default options can be changed with the SetOptions command e.g.

```
SetOptions[ODBapi, db → "DemoDB"];
```

Latest Changes v.1.0.3

updValues	updPropertyValues and updPropertyValue merged into updValues
delRecords	Enhanced with the all–>True Option
addIndex	Create Index on a property of a class
addEdge	Associate two records, i.e. link them bidirectionally
addInstance	Create a unique instance or simply an instance of Document or Vertex by setting a specific value on a field
addRecord	Enhanced to create either an empty Document or a Document with content
addValues	The construct option has been removed. Insert INTO Class works with both Document and Vertex
getRecords	Return records

Tutorials

XXXX

Related Demonstrations

XXXX

Related Links

ODBgetDataset . ODBgetFieldAttributes

See Also

XXXX

Examples

More Examples ▹

Load the two packages that are included in the **DBAPI`** context, the **DBAPI`Utils`** and the **DBAPI`OrientDB`**:

```
In[8]:= << DBAPI`OrientDB`
```

```
DBAPI Application Project

Promoted and Distributed by HEALIS- Healthy Information Systems/Services

Running on Mathematica Version → 10.3.1 for Microsoft Windows (64-bit) (December 9, 2015)


Data Utilities Package v0.9

Copyright December 2015, By Athanassios I. Hatzis

Distributed under GNU LGPL - GNU Lesser General Public License


OrientDB API Package v1.0.3

Copyright February 2016, By Athanassios I. Hatzis

Distributed under GNU LGPL - GNU Lesser General Public License
```

Add Commands

Add Database

Add Database command creates a, disk-based or memory, document or graph, database with a username and password on remote database Server

In[18]:=
```
ODBapi[com → "addDatabase", db → "DemoDB",
  dbtype → "plocal/graph", usr → "root", pwd → "123", debug → True] // Short
```

```
http://localhost:2480/database/DemoDB/plocal/graph
=== Body ===
```

Out[18]//Short=
```
{{"classes":[{"name":"ORole","superClass": … s":[{"name":"strictSql","value":"true"}]}]}, 200}
```

Add Class

Add a class by executing SQL CREATE CLASS to create a new class in the schema and optionally extend a superclass.

In[21]:=
```
ODBapi[com → "addClass", db → "DemoDB", class → "Person", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE CLASS Person
```

Out[21]=
```
{{"result":[{"@type":"d","@version":0,"value":12}]}, 200}
```

In[22]:=
```
ODBapi[com → "addClass", db → "DemoDB", class → "Employee", superclass → "Person", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE CLASS Employee extends Person
```

Out[22]=
```
{{"result":[{"@type":"d","@version":0,"value":13}]}, 200}
```

In[12]:=
```
ODBapi[com → "addClass", db → "DemoDB", class → "Flower", superclass → "V", debug → True];
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE CLASS Flower extends V
```

In[13]:=
```
ODBapi[com → "addClass", db → "DemoDB", class → "isOwnerOf", superclass → "E", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE CLASS isOwnerOf extends E
```

Out[13]=
```
{{"result":[{"@type":"d","@version":0,"value":17}]}, 200}
```

Add a class via HTTP

In[23]:=
```
ODBapi[com → "addClassViaHTTP", db → "DemoDB", class → "Company", debug → True]
```

```
http://localhost:2480/class/DemoDB/Company
=== Body ===
```

Out[23]=
```
{14, 201}
```

Add Property

Add a property via SQL CREATE PROPERTY command

In[24]:=
```
ODBapi[com → "addProperty", db → "DemoDB", class → "Person",
 propnam → "personName", proptype → "STRING", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE PROPERTY Person.personName STRING
```

Out[24]=
```
{{"result":[{"@type":"d","@version":0,"value":1}]}, 200}
```

In[25]:=
```
ODBapi[com → "addProperty", db → "DemoDB",
 class → "Car", propnam → "vehno", proptype → "STRING", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE PROPERTY Car.vehno STRING
```

Out[25]=
```
{{"result":[{"@type":"d","@version":0,"value":1}]}, 200}
```

Add a property via HTTP

In[25]:=
```
ODBapi[com → "addPropertyViaHTTP", db → "DemoDB", class → "Person",
 propnam → "personGender", proptype → "STRING", debug → True]
```

```
http://localhost:2480/property/DemoDB/Person/personGender/STRING
=== Body ===
```

Out[25]=
```
{2, 201}
```

Add Records

In Wolfram language we can represent the record with a hierarchical List of Rule

In[21]:=
```
john = {
   "telephones" → Thread[{"home", "business", "mobile"} → {"2104566345", "2108856844", "6974059256"}],
   "firstName" → "John", "lastName" → "Brown", "DOB" → "1971-10-01", "age" → 44};
```

Then we transform the list of rules above to a JSON string using the DBexpressionToJSON function of the **Utilities Package.**

In[23]:=
```
johnJSON = john // DBexpressionToJSON[#, compact → True] &
```

Out[23]=
```
{"telephones":{"home":"2104566345","business":"2108856844","mobile":"6974059256"},"firstName":"John",
  "lastName":"Brown","DOB":"1971-10-01","age":44}
```

Add a JSON stuctured Document by inserting the schemaless record into Person Class via the message body.

In[24]:=
```
ODBapi[com → "addRecord", db → "DemoDB", class → "Person", record → johnJSON, debug → True]
```

```
http://localhost:2480/document/DemoDB
=== Body ===

 {"@class":"Person","telephones":{"home":"2104566345","business":"2108856844","mobile":"6974059256"},"
   firstName":"John","lastName":"Brown","DOB":"1971-10-01","age":44}
```

Out[24]=
```
{{"@type":"d","@rid":"#12:0","@version":1,"@class":"Person","telephones":{"home":"2104566345","
    business":"2108856844","mobile":"6974059256"},"firstName":"John","lastName":"Brown","DOB":"1971-
    10-01","age":44}, 201}
```

Add another instance of Person class, this time with a structured document without content, i.e. a Document with only metadata info, @type, @rid, @version, @class. Omit the record parameter in that case.

In[30]:=
```
ODBapi[com → "addRecord", db → "DemoDB", class → "Person", debug → True]
```

```
http://localhost:2480/document/DemoDB
=== Body ===
{"@class":"Person"}
```

Out[30]=
```
{{"@type":"d","@rid":"#12:1","@version":1,"@class":"Person"}, 201}
```

You can also create an empty record, i.e. instance of Document or Vertex with the **addInstance** command.

Add the above JSON stuctured Document into Person via SQL INSERT INTO command. To memorize this command, think of a content that you add on an empty record.

In[31]:=
```
ODBapi[com → "addContent", db → "DemoDB", class → "Person",
  record → johnJSON, construct → "DOCUMENT", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
INSERT INTO Person content
   {"telephones":{"home":"2104566345","business":"2108856844","mobile":"6974059256"},"firstName":"
   John","lastName":"Brown","DOB":"1971-10-01","age":44}
```

Out[31]=
```
{{"result":[{"@type":"d","@rid":"#12:2","@version":1,"@class":"Person","telephones":{"home":"
    2104566345","business":"2108856844","mobile":"6974059256"},"firstName":"John","lastName":"Brown"
    ,"DOB":"1971-10-01","age":44}]}, 200}
```

In[32]:=
```
ODBapi[com → "addContent", db → "DemoDB", class → "Flower",
  record → "{'genus':'Mentha','kingdom':'Plantae'}", construct → "VERTEX", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE VERTEX Flower content {'genus':'Mentha','kingdom':'Plantae'}
```

Out[32]=
```
{{"result":[{"@type":"d","@rid":"#15:0","@version":1,"@class":"Flower","genus":"Mentha","kingdom":"
    Plantae"}]}, 200}
```

Add the above JSON stuctured Document into Person via SQL INSERT VALUES command

In[35]:=
```
kstr = john // Association // Keys // DBListSetToSQL92[#, values → False] &
```

Out[35]=
```
(telephones, firstName, lastName, DOB, age)
```

We apply a series of transformation to get the Keys and Values strings in SQL-92 format. Association and Keys are Wolfram functions, DBListSetToSQL92 is a function of the **Utilities package**.

In[36]:=
```
vstr = john // Association // Values // DBListSetToSQL92
```

Out[36]=
```
({"home":"2104566345","business":"2108856844","mobile":"6974059256"},
  "John", "Brown", "1971-10-01", 44)
```

In[37]:=
```
ODBapi[com → "addValues", db → "DemoDB", class → "Person", keys → kstr, values → vstr, debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
INSERT INTO Person (telephones, firstName, lastName, DOB, age)
   VALUES ({"home":"2104566345","business":"2108856844","mobile":"6974059256"},
   "John", "Brown", "1971-10-01", 44)
```

Out[37]=
```
{{"result":[{"@type":"d","@rid":"#12:3","@version":1,"@class":"Person","telephones":{"business":"
   2108856844","mobile":"6974059256","home":"2104566345"},"firstName":"John","lastName":"Brown","
   DOB":"1971-10-01","age":44}]}, 200}
```

Another example adding multiple vertices, (Flower class extends V)

In[38]:=
```
ODBapi[com → "addValues", db → "DemoDB", class → "Flower", keys → "(genus,kingdom)",
 values → "('Citrus','Plantae'),('Fragaria','Plantae')", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
INSERT INTO Flower (genus,kingdom) VALUES ('Citrus','Plantae'),('Fragaria','Plantae')
```

Out[38]=
```
{{"result":[{"@type":"d","@rid":"#15:1","@version":1,"@class":"Flower","genus":"Citrus","kingdom":"
   Plantae"},
   {"@type":"d","@rid":"#15:2","@version":1,"@class":"Flower","genus":"Fragaria","kingdom":"Plantae
   "}]},
  200}
```

Add Index

Create an Index on a property of a class and specify that the values of that property cannot be duplicated, i.e. unique index type.

In[27]:=
```
ODBapi[com → "addIndex", db → "DemoDB", class → "Car",
 indexnam → "vehnoNDX", indextype → "UNIQUE", propnam → "vehno", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE INDEX vehnoNDX ON Car (vehno) UNIQUE
```

Out[27]=
```
{{"result":[{"@type":"d","@version":0,"value":1,"@fieldTypes":"value=l"}]}, 200}
```

Add Instance

This command creates a new instance of a class. If the class extends Vertex, then it is based on the SQL CREATE VERTEX command. If it is simply a Document class, then SQL INSERT INTO is called instead. The class and construct options are mandatory.

On the contrary, propnam and propval arguments are optional

In[18]:=
```
ODBapi[com → "addInstance", db → "DemoDB", class → "Car",
 propnam → "vehno", propval → "'YFR5886'", construct → "DOCUMENT", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
INSERT INTO Car SET vehno='YFR5886'
```

Out[18]=
```
{{"result":[{"@type":"d","@rid":"#16:2","@version":1,"@class":"Car","vehno":"YFR5886"}]}, 200}
```

In[23]:=
```
ODBapi[com → "addInstance", db → "DemoDB", class → "Car", construct → "DOCUMENT", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
INSERT INTO Car CONTENT {"@class":"Car"}
```

Out[23]=
```
{{"result":[{"@type":"d","@rid":"#16:3","@version":1,"@class":"Car"}]}, 200}
```

In[21]:=
```
ODBapi[com → "addInstance", db → "DemoDB", class → "Flower",
 propnam → "genus", propval → "'ROSARIA'", construct → "VERTEX", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE VERTEX Flower SET genus='ROSARIA'
```

Out[21]=
```
{{"result":[{"@type":"d","@rid":"#15:4","@version":1,"@class":"Flower","genus":"ROSARIA"}]}, 200}
```

In[22]:=
```
ODBapi[com → "addInstance", db → "DemoDB", class → "Flower", construct → "VERTEX", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE VERTEX Flower
```

Out[22]=
```
{{"result":[{"@type":"d","@rid":"#15:5","@version":1,"@class":"Flower"}]}, 200}
```

Add a single instance, i.e. a record with property values that cannot be duplicated. Omit the construct argument as it can be applied in both a Vertex and Document Class. As a prerequisite step for this a UNIQUE index should be created first, see add Index above. Once the index has been set, we can use the following command and set uniq argument to true. This forces OrientDB to use the SQL UPSERT command to update the value and change the version number if it exists or create a new record it if it does not exist.

In[36]:=
```
ODBapi[com → "addInstance", db → "DemoDB", class → "Car",
 propnam → "vehno", propval → "'UIT5564'", uniq → True, debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
UPDATE Car SET vehno='UIT5564' upsert return after @rid where vehno='UIT5564'
```

Out[36]=
```
{{"result":[{"@type":"d","@rid":"#16:4","@version":6,"@class":"Car","vehno":"UIT5564"}]}, 200}
```

Run the command above multiple times and notice how the @version field is changed. An attempt to create the record with the same value using a different command results in an error.

In[38]:=
```
ODBapi[com → "addContent", db → "DemoDB", class → "Car",
 record → "{vehno:'UIT5564'}", construct → "DOCUMENT", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
INSERT INTO Car content {vehno:'UIT5564'}
```

Out[38]=
```
{{

  "errors": [

    {

      "code": 500,

      "reason": 500,

      "content": "com.orientechnologies.orient.core.storage.ORecordDuplicatedException:
    Cannot index record Car{vehno:UIT5564}: found duplicated key 'UIT5564'
    in index 'vehnoNDX' previously assigned to the record #16:4 RID=#16:4"

    }

  ]

}, 500}
```

Add Edge

Update Car and Person to extend Vertex Class

In[14]:=
```
ODBapi[com → "updClass", db → "DemoDB", class → "Car", attribnam → "SUPERCLASS", attribval → "V"];
```

In[15]:=
```
ODBapi[com → "updClass", db → "DemoDB", class → "Person", attribnam → "SUPERCLASS", attribval → "V"];
```

AddEdge command creates a bidirectional link between two records. In that case we say that the two instances are associated. Three parameters are mandatory in all forms of addEdge command: class, from and to.

The simplest form is to associate the records with an OrientDB lightweight edge. This is an edge with an empty content, i.e. no fields or properties defined or set on the edge.

```
In[16]:= ODBapi[com → "addEdge", db → "DemoDB", class → "isOwnerOf", from → "12:0", to → "16:0", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE EDGE isOwnerOf FROM 12:0 TO 16:0
```

```
Out[16]= {{"result":[{"@type":"d","@version":0,"@class":"isOwnerOf","in":"#16:0","out":"#12:0","@fieldTypes":"
           in=x,out=x"}]}, 200}
```

Prior to the execution of this command make sure that database has been configured to use LightWeightEdges. See the updDatabase command on this.

The other form of addEdge command is using the propnam and propval arguments to set a property value on the edge.

```
In[21]:= ODBapi[com → "updDatabase", db → "DemoDB",
          attribnam → "custom useLightweightEdges=", attribval → "false"];
```

```
In[22]:= ODBapi[com → "addEdge", db → "DemoDB", class → "isOwnerOf", from → "12:3",
          to → "16:0", propnam → "country", propval → "'Germany'", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE EDGE isOwnerOf FROM 12:3 TO 16:0
```

```
Out[22]= {{"result":[{"@type":"d","@rid":"#17:0","@version":1,"@class":"isOwnerOf","out":"#12:3","in":"#16:0",
           "@fieldTypes":"out=x,in=x"}]}, 200}
```

There is also a third form of addEdge command that is using the record argument to create content on the edge.

```
In[24]:= ODBapi[com → "addEdge", db → "DemoDB", class → "isOwnerOf", from → "12:0",
          to → "16:2", record → "{'country':'Greece','since':'2014/1/1'}", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
CREATE EDGE isOwnerOf FROM 12:0 TO 16:2 CONTENT {'country':'Greece','since':'2014/1/1'}
```

```
Out[24]= {{"result":[{"@type":"d","@rid":"#17:1","@version":1,"@class":"isOwnerOf","country":"Greece","since":
           "2014/1/1","out":"#12:0","in":"#16:2","@fieldTypes":"out=x,in=x"}]}, 200}
```

Add OSQL

AddOSQLScript means that the database may change. OrientDB batch of SQL commands can be non-idempotent and are executed via POST in a single call. The 'mode' argument in batch mode is mandatory, default value is "command".

```
In[43]:= osqlScript = "
         CREATE CLASS Car EXTENDS V;
         CREATE VERTEX Car SET brand='FIAT',model='Punto',year='2000-01-01'";
         ODBapi[com → "addOSQL", db → "DemoDB", sql → osqlScript, mode → "BATCH", debug → True]
```

```
http://localhost:2480/batch/DemoDB
=== Body ===
{
    "transaction": false,
    "operations": [
        {
            "type": "script",
            "language": "sql",
            "script": "\nCREATE CLASS Car EXTENDS
    V;\nCREATE VERTEX Car SET brand='FIAT',model='Punto',year='2000-01-01'"
        }
    ]
}
```

```
Out[44]= {{"result":[{"@type":"d","@rid":"#16:0","@version":1,"@class":"Car","brand":"FIAT","model":"Punto","
           year":"2000-01-01"}]}, 200}
```

AddOSQLCommand means that the database may change.
OrientDB SQL command executed under the hood via POST can be non-idempotent

In[45]:=
```
ODBapi[com → "addOSQL", db → "DemoDB",
  sql → "UPDATE Car SET model='bravo'", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
UPDATE Car SET model='bravo'
```

Out[45]=
```
{{"result":[{"@type":"d","@version":0,"value":1}]}, 200}
```

Update Commands

Update Database

Update Database command is executed via SQL - ALTER DATABASE operation. This command updates database settings.

Change DATETIMEFORMAT attribute in the database to use ISO 8601 dates

In[56]:=
```
ODBapi[com → "updDatabase", db → "DemoDB", attribnam → "DATETIMEFORMAT",
  attribval → "yyyy/MM/dd'T'HH:mm:ss.SSS'Z'", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
ALTER DATABASE DATETIMEFORMAT yyyy/MM/dd'T'HH:mm:ss.SSS'Z'
```

Out[56]=
```
{, 204}
```

In[57]:=
```
ODBapi[com → "updDatabase", db → "DemoDB",
  attribnam → "DATEFORMAT", attribval → "yyyy/MM/dd", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
ALTER DATABASE DATEFORMAT yyyy/MM/dd
```

Out[57]=
```
{, 204}
```

Enable Lightweight Edges, i.e. bidirectional links

In[47]:=
```
ODBapi[com → "updDatabase", db → "DemoDB",
  attribnam → "custom useLightweightEdges=", attribval → "true", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
ALTER DATABASE custom useLightweightEdges= true
```

Out[47]=
```
{, 204}
```

Update Class

Update Class command is executed via SQL - ALTER CLASS operation. This command alters a class in the schema.

In[48]:=
```
ODBapi[com → "updClass", db → "DemoDB", class → "Person",
  attribnam → "SHORTNAME", attribval → "P", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
ALTER CLASS Person SHORTNAME P
```

Out[48]=
```
{, 204}
```

Update Property

Update Property command is executed via SQL - ALTER PROPERTY operation. This command alter's a class's property in the schema.

```
In[49]:= ODBapi[com → "updProperty", db → "DemoDB", class → "Person",
          propnam → "personName", attribnam → "MANDATORY", attribval → "false", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
ALTER PROPERTY Person.personName MANDATORY false
```

```
Out[49]= {, 204}
```

Update Value(s)

Update property values for ALL records in a class that have that property (field) or insert key-value pairs in the Document record if that field does not exist. The 'all' argument is mandatory and must be set to true. Also the propnam, proval and class arguments are mandatory too.

```
In[51]:= ODBapi[com → "updValues", db → "DemoDB", class → "Car",
          propnam → "cc", propval → "1200", all -> True, debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
UPDATE Car set cc=1200
```

```
Out[51]= {{"result":[{"@type":"d","@version":0,"value":1}]}, 200}
```

Similar command, but this time update the field value of multiple records based on a condition

```
In[55]:= ODBapi[com → "updValues", db → "DemoDB", class → "Person",
          propnam → "age", propval → "57 where age<60", all → True, debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
UPDATE Person set age=57 where age<60
```

```
Out[55]= {{"result":[{"@type":"d","@version":0,"value":3}]}, 200}
```

Update the property value of a single record. The default value of 'all' argument is False and can be omitted. Note that class argument is not required here.

```
In[9]:= ODBapi[com → "updValues", db → "DemoDB", id → "16:0",
         propnam → "year", propval → "'2001/01/01'", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
UPDATE 16:0 set year='2001/01/01'
```

```
Out[9]= {{"result":[{"@type":"d","@version":0,"value":1}]}, 200}
```

Update Record

Update a record via the PUT-Document HTTP API command. In this update mode the entire document is replaced. The method option is optional, as the default value is "PUT"

First we modify the Rule expression of the **addRecord example** and convert it to a JSON string using built-in **ToAssociations** function and **DBexpressionToJSON** functions of the **Utilities Package.**

```
In[25]:= johnJSON = ReplacePart[john // ToAssociations, {
              {"telephones", "business"} → "2108880809",
              {"telephones", "mobile"} → "6972020202"}] // DBexpressionToJSON;
```

Then we pass the result as the value of record → option

```
In[26]:= ODBapi[com → "updRecord", db → "DemoDB", id → "12:0", record → johnJSON, debug → True]
```

```
http://localhost:2480/document/DemoDB/12:0
=== Body ===
{
    "telephones": {
        "home": "2104566345",
        "business": "2108880809",
        "mobile": "6972020202"
    },
    "firstName": "John",
    "lastName": "Brown",
    "DOB": "1971-10-01",
    "age": 44
}
```

```
Out[26]= {{"@type":"d","@rid":"#12:0","@version":2,"@class":"Person","telephones":{"home":"2104566345","
          business":"2108880809","mobile":"6972020202"},"firstName":"John","lastName":"Brown","DOB":"1971-
          10-01","age":44}, 200}
```

```
In[63]:= johnJSON = ReplacePart[john // ToAssociations, {
            {"firstName"} → "Marie",
            {"lastName"} → "Osborne"}] // DBexpressionToJSON;
```

```
In[64]:= ODBapi[com → "updRecord", db → "DemoDB", id → "12:3", record → johnJSON, method → "PUT", debug → True]
```

```
http://localhost:2480/document/DemoDB/12:3
=== Body ===
{
    "telephones": {
        "home": "2104566345",
        "business": "2108856844",
        "mobile": "6974059256"
    },
    "firstName": "Marie",
    "lastName": "Osborne",
    "DOB": "1971-10-01",
    "age": 44
}
```

```
Out[64]= {{"@type":"d","@rid":"#12:3","@version":3,"@class":"Person","age":44,"telephones":{"business":"
          2108856844","mobile":"6974059256","home":"2104566345"},"firstName":"Marie","lastName":"Osborne",
          "DOB":"1971-10-01"}, 200}
```

Update a record via the PATCH-Document HTTP API command. This will update the record Document with only the difference to apply. The method and ver arguments here are mandatory.

Build an expression with the telephones to change only

```
In[65]:= telcom = <|"telephones" →
            Thread[{"home", "business", "mobile", "pager"} →
              {"2104566345", "2109990909", "6973030303", "444"}]|> // DBexpressionToJSON;
```

In[67]:=
```
ODBapi[com → "updRecord", db → "DemoDB", id → "12:3",
 record → telcom, ver → 3, debug → True, method → "PATCH"]
```

```
http://localhost:2480/document/DemoDB/12:3
=== Body ===
{"@class":"Person", "@version":3,
    "telephones": {
        "home": "2104566345",
        "business": "2109990909",
        "mobile": "6973030303",
        "pager": "444"
    }
}
```

Out[67]=
```
{{"@type":"d","@rid":"#12:3","@version":4,"@class":"Person","age":44,"telephones":{"home":"2104566345
  ","business":"2109990909","mobile":"6973030303","pager":"444"},"firstName":"Marie","lastName":"
  Osborne","DOB":"1971-10-01"}, 200}
```

Get Commands

Get Server

Get Server Information

In[9]:=
```
ODBapi[com → "getServer", usr → "root", pwd → "123", debug → True] // ODBgetDataset
```

```
http://localhost:2480/server
=== Body ===
```

Out[9]=

connections	{ ‹| connectionId → 103, remoteAddress → /0:0:0:0:0:0:0:1:50188, db → DemoDB, ⋯13 |›, ⋯1 }
dbs	{ }
storages	{ ‹| name → GratefulDeadConcerts, type → OLocalPaginatedStorage, path → C:/orientdb219/databases/GratefulDe
properties	{ ‹| name → db.pool.min, value → 1 |›, ‹| name → db.pool.max, value → 50 |›, ‹| name → profiler.enabled, value -
globalProperties	{ ‹| key → environment.dumpCfgAtStartup, description → Dumps the configuration at application startup, value → …
3 levels | 5 rows	

The postfix operation of **ODBgetDataset** transforms JSON Output above into a Wolfram Dataset. Further processing can be applied on the resulting Dataset.

In[107]:=
```
%["connections", All, {"connectionId", "remoteAddress", "db", "user"}]
```

Out[107]=

connectionId	remoteAddress	db	user
65	/127.0.0.1:49386	–	–
39	/127.0.0.1:49357	–	–
40	/127.0.0.1:49357	–	–
41	/127.0.0.1:49357	–	–
29	/0:0:0:0:0:0:0:1:49343	DemoDB	admin
2 levels | 5 rows			

Get Database

Get Database Information

```
In[11]:= dbInfo = ODBapi[com → "getDatabases", db → "DemoDB", debug → True] // ODBgetDataset
```

```
http://localhost:2480/database/DemoDB
=== Body ===
```

Out[11]=

server	‹| version → 2.1.9–SNAPSHOT, build → 2.1.x@r${buildNumber}; 2016–01–07 10:51:24+0000, osName → Windows 10,
classes	{ ‹| name → Car, superClass → V, superClasses → …, ⋯7 |›, ⋯15 }
clusters	{ ‹| id → 0, name → internal, records → 3, conflictStrategy → …, ⋯4 |›, ⋯16 }
currentUser	admin
indexes	{ ‹| name → OUser.name, configuration → … |›, ⋯2 }
config	‹| values → { ‹| name → dateFormat, value → … |›, ‹| name → dateTimeFormat, value → … |›, ⋯8 }, ⋯1 |›
6 levels | 6 rows	

The postfix operation of **ODBgetDataset** transforms JSON Output above into a Wolfram Dataset Further processing can be applied on the resulting Dataset.

```
In[12]:= dbInfo["classes", All, {"name", "superClass", "records", "properties"}]
```

Out[12]=

name	superClass	records	properties
Car	V	1	KeyAbsent
Company		0	KeyAbsent
E		0	KeyAbsent
Employee	Person	0	{ ‹| name → personGender, linkedClass → KeyAbsent, type → STRING, ⋯6 |›, ⋯1 }
Flower	V	3	KeyAbsent
OFunction		0	{ ‹| name → idempotent, linkedClass → KeyAbsent, type → BOOLEAN, ⋯6 |›, ⋯4 }
OIdentity		6	KeyAbsent
ORIDs		0	KeyAbsent
ORestricted		0	{ ‹| name → _allowUpdate, linkedClass → OIdentity, type → LINKSET, ⋯7 |›, ⋯3 }
ORole	OIdentity	3	{ ‹| name → mode, linkedClass → KeyAbsent, type → BYTE, ⋯6 |›, ⋯3 }
OSchedule		0	{ ‹| name → function, linkedClass → OFunction, type → LINK, ⋯7 |›, ⋯6 }
OTriggered		0	KeyAbsent
OUser	OIdentity	3	{ ‹| name → password, linkedClass → KeyAbsent, type → STRING, ⋯6 |›, ⋯3 }
Person		4	{ ‹| name → personGender, linkedClass → KeyAbsent, type → STRING, ⋯6 |›, ⋯1 }
V		4	KeyAbsent
_studio		1	KeyAbsent
4 levels | 16 rows			

Get Databases - A list of the databases on the server

```
In[13]:= ODBapi[com → "getDatabases", debug → True]
```

```
http://localhost:2480/listDatabases
=== Body ===
```

```
Out[13]= {{"@type":"d","@version":0,"databases":["GratefulDeadConcerts","DemoDB"],"@fieldTypes":"databases=e"}
         , 200}
```

Get Class

Get Class Information from the server

In[35]:=
```
classInfo = ODBapi[com → "getClass", db → "DemoDB", class → "Person", debug → True]
```

```
http://localhost:2480/class/DemoDB/Person
=== Body ===
```

Out[35]=
```
{{"name":"Person","superClass":"V","superClasses":["V"],"alias":"P","abstract":false,"strictmode":
    false,"clusters":[12],"defaultCluster":12,"clusterSelection":"round-robin","records":4,"
    properties":[{"name":"personGender","type":"STRING","mandatory":false,"readonly":false,"notNull"
    :false,"min":null,"max":null,"regexp":null,"collate":"default"},{"name":"personName","type":"
    STRING","mandatory":false,"readonly":false,"notNull":false,"min":null,"max":null,"regexp":null,"
    collate":"default"}]}, 200}
```

The postfix operation of **ODBgetDataset** transforms JSON Output above into a Wolfram Dataset Further processing can be applied on the resulting Dataset.

In[36]:=
```
classInfoDS = classInfo // ODBgetDataset
```

Out[36]=

name	Person
superClass	V
superClasses	{V}
alias	P
abstract	False
strictmode	False
clusters	{12}
defaultCluster	12
clusterSelection	round-robin
records	4
properties	{ ‹| name → personGender, type → STRING, mandatory → False, readonly → False, ⋯5 |›, ⋯1 }

3 levels | 11 rows

With the package function **ODBgetFieldAttributes** we can view specific attributes for all properties (fields) of the Car class

In[38]:=
```
ODBgetFieldAttributes[classInfoDS, {"name", "type", "mandatory", "notNull"}]
```

Out[38]=

name	type	mandatory	notNull
personGender	STRING	False	False
personName	STRING	False	False

2 levels | 2 rows

Get Record

Get a single Record - Returns a JSON structured document with data and metadata fields that represents an OrientDB record. The id parameter is mandatory in that case.

In[43]:=
```
ODBapi[com → "getRecords", db → "DemoDB", id → "16:0", debug → True]
```

```
http://localhost:2480/document/DemoDB/16:0
=== Body ===
```

Out[43]=
```
{{"@type":"d","@rid":"#16:0","@version":7,"@class":"Car","in_isOwnerOf":["#12:0","#12:3","#17:0"],"
    brand":"FIAT","model":"bravo","year":"2001/01/01","cc":1200,"@fieldTypes":"in_isOwnerOf=g"},
  200}
```

In[44]:=
```
% // ODBgetDataset
```

Out[44]=

@type	d
@rid	#16:0
@version	7
@class	Car
in_isOwnerOf	{#12:0, #12:3, #17:0}
brand	FIAT
model	bravo
year	2001/01/01
cc	1200
@fieldTypes	in_isOwnerOf=g
2 levels | 10 rows	

Get data from multiple records, the projection argument (prjkt), and the propnam, propval arguments that specify a condition can be omitted.

In[9]:=
```
ODBapi[com → "getRecords", db → "DemoDB", prjkt → "@rid, firstName, lastName, DOB",
 class → "Person", propnam → "lastName", propval → "Brown", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
select @rid, firstName, lastName, DOB from Person WHERE lastName="Brown"
```

Out[9]=
```
{{"result":[{"@type":"d","@rid":"#-2:0","@version":0,"rid":"#12:0","firstName":"John","lastName":"
    Brown","DOB":"1971-10-01","@fieldTypes":"rid=x"},
    {"@type":"d","@rid":"#-2:1","@version":0,"rid":"#12:2","firstName":"John","lastName":"Brown","
    DOB":"1971-10-01","@fieldTypes":"rid=x"}]}, 200}
```

In[10]:=
```
(ODBgetDataset@%)[All, {"rid", "firstName", "lastName", "DOB"}]
```

Out[10]=

rid	firstName	lastName	DOB
#12:0	John	Brown	1971–10–01
#12:2	John	Brown	1971–10–01
2 levels | 2 rows			

In[16]:=
```
(ODBapi[com → "getRecords", prjkt → "@rid", db → "DemoDB", class → "Person", debug → True] //
   ODBgetDataset)[All, {"rid"}]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
select @rid from Person
```

Out[16]=

rid
#12:0
#12:1
#12:2
#12:3
2 levels | 4 rows

Get OSQL

Executes an OrientDB SQL query against the database. The SQL command is read-only. It is executed via the GET method of HTTP and therefore it cannot change the database.

```
In[18]:= ODBapi[com → "getOSQL", db → "DemoDB",
          sql → "select from Flower where kingdom='Plantae'", debug -> True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
select from Flower where kingdom='Plantae'
```

```
Out[18]= {{"result":[{"@type":"d","@rid":"#15:0","@version":1,"@class":"Flower","kingdom":"Plantae","genus":"
           Mentha"},
          {"@type":"d","@rid":"#15:1","@version":1,"@class":"Flower","kingdom":"Plantae","genus":"Citrus"}
          ,
          {"@type":"d","@rid":"#15:2","@version":1,"@class":"Flower","kingdom":"Plantae","genus":"Fragaria
          "}]},
         200}
```

Del Commands

Del Database

Delete a Database with basic authentication to the server

```
In[77]:= ODBapi[com → "delDatabase", db → "TestDB", usr → "root", pwd → "123", debug → True]
```

```
http://localhost:2480/database/TestDB
=== Body ===
```

```
Out[77]= {, 204}
```

Del Class

Delete a Class, i.e. remove completely the class from the schema. If the class extends vertex (V) then all the vertices have to be deleted first. Command delClass is based on SQL DROP CLASS operation.

```
In[19]:= ODBapi[com -> "delClass", db → "DemoDB", class → "Company", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
DROP CLASS Company
```

```
Out[19]= {{"result":[{"@type":"d","@version":0,"value":true}]}, 200}
```

Del Property

Delete a property via SQL - DROP PROPERTY
Be aware that although the property is removed from the schema, it still remains as a key in records that have been created with that property.

```
In[20]:= ODBapi[com → "delProperty", db → "DemoDB", class → "Person", propnam → "personGender", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
DROP PROPERTY Person.personGender
```

```
Out[20]= {, 204}
```

Delete values - This command removes a field from all records by executing SQL UPDATE

```
In[21]:= ODBapi[com → "delValues", db → "DemoDB", class → "Person", propnam → "telephones", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
UPDATE Person remove telephones
```

```
Out[21]= {{"result":[{"@type":"d","@version":0,"value":4}]}, 200}
```

Del Records

Delete all records from a DOCUMENT CLASS (based on SQL TRUNCATE) or VERTEX CLASS (based on SQL DELETE VERTEX). The class, all and construct parameters are mandatory in that case.

```
In[22]:= ODBapi[com → "delRecords", db → "DemoDB",
          class → "Person", all → True, construct → "VERTEX", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
DELETE VERTEX Person
```

```
Out[22]= {{"result":[{"@type":"d","@version":0,"value":4}]}, 200}
```

Delete Records command can also be used to erase specific records with RIDs. It is based on SQL TRUNCATE RECORD by listing the record IDs.

```
In[23]:= ODBapi[com → "delRecords", db → "DemoDB", id → "[16:1, 16:3]", debug → True]
```

```
http://localhost:2480/command/DemoDB/sql
=== Body ===
TRUNCATE RECORD [16:1, 16:3]
```

```
Out[23]= {{"result":[{"@type":"d","@version":0,"value":2}]}, 200}
```

Del Connection

Delete Server Connection - It requires a connection id and root password to kill the connection

```
In[94]:= ODBapi[com → "delConnection", con → "49", usr → "root", pwd → "123", debug → True]
```

```
http://localhost:2480/connection/kill/49
=== Body ===
```

```
Out[94]= {, 204}
```

Import/Export

Import

Imports a database from an uploaded JSON text file

```
In[24]:= ODBapi[com → "impDatabase", db → "FilmDB", debug → True];
```

```
http://localhost:2480/import/FilmDB
=== Body ===
```

Export

Exports a gzip file that contains the database JSON export via GET-Export

```
In[26]:= ODBapi[com → "expDatabase", db → "DemoDB", debug → True];
```

```
http://localhost:2480/export/DemoDB
=== Body ===
```

Login/Logout

Login

Connect to a remote server using basic authentication via GET-Connect HTTP method

```
In[24]:= ODBapi[com → "login", db → "GratefulDeadConcerts", usr → "admin", pwd -> "admin", debug → True]
```

```
http://localhost:2480/connect/GratefulDeadConcerts
=== Body ===
```

```
Out[24]= {, 204}
```

Logout

Disconnect from server via GET-Disconnect HTTP method

```
In[44]:= ODBapi[com -> "logout", debug → True]
```

```
http://localhost:2480/disconnect
=== Body ===
```

```
Out[44]= $Failed
```

More Examples

- Scope
- Generalizations & Extensions
- Options
- Applications
- Properties & Relations
- Possible Issues
- Interactive Examples
- Neat Examples