We define the business logic with life-cycle objects on the server side. The access authorization of an application is checked by a security manager. The business logic is usually available via master- or subconnections from the client.

In order to complete the technology independence, the complete business logic of an application is also available via REST.

For the use of the REST services, the authentication with username and password is necessary. BASIC is used as the authentication mechanism. The credentials are checked by the security manager of the application as usual. You do not need to change a source code line to integrate the REST services.

How it works

The REST implementation in JVx has been implemented with Restlet. To use the REST services, the deployment descriptor must be configured as follows:

```
<servlet>
    <servlet-name>RestletServlet</servlet-name>
    <servlet-class>org.restlet.ext.servlet.ServerServlet</servlet-class>

    <init-param>
      <param-name>org.restlet.application</param-name>
      <param-value>com.sibvisions.rad.server.http.rest.RESTAdapter</param-
value>
    </init-param>
  </servlet>

  <servlet-mapping>
    <servlet-name>RestletServlet</servlet-name>
    <url-pattern>/services/rest/*</url-pattern>
  </servlet-mapping>
```

With this configuration, the following services are available:

Available services

Storage access (CRUD, Meta Data)

- Select
- Insert
- Update
- Delete
- Meta Data

GET-Request (Select)

Query all data:

`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME`

Query exactly one record:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME/PRIMARY_KEY`

If the PK is composed of several columns, the query parameters must be used:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME?PKCOLUMN=VALUE&PKCOLUMN2=VALUE2`

The query parameters can also be used to perform filtering with columns that are not PK columns.

Read more about complex query parameters?

GET-Response

The response always contains a list of HashMaps in JSON format. The column name is used as the key.

Example:

```
[ { "ID" : 0,
    "POST_ID" : 127,
    "POST_PLZ" : "1127",
    "STIEGE" : 8,
    "STRA_ID" : 68,
    "STRA_NAME" : "Strasse (69)",
    "HAUSNUMMER" : 37,
    "TUERNUMMER" : 79
  },
  { "ID" : 1,
    "POST_ID" : 50,
    "POST_PLZ" : "1050",
    "STIEGE" : 7,
    "STRA_ID" : 55,
    "STRA_NAME" : "Strasse (56)",
    "HAUSNUMMER" : 37,
    "TUERNUMMER" : 60
  },
]
```

POST-Request (Insert)

Insert a new record:
`http://server:port/webapp/services/rest/APPLICATION_NAME/LIFECYCLE_CLASS/data/STORAGE_NAME`

The request requires a HashMap in JSON format. The column name is used as the key.

Example:

```
{ "POST_ID" : "0",
  "STRA_ID" : "0",
  "HAUSNUMMER" : "9999"
}
```

POST-Response

The response returns the complete record in JSON format:

```
{ "ID" : 1008,
  "POST_ID" : 0,
  "POST_PLZ" : "1000",
  "STIEGE" : null,
  "STRA_ID" : 0,
  "STRA_NAME" : "Strasse (1)",
  "HAUSNUMMER" : 9999,
  "TUERNUMMER" : null
}
```

PUT-Request (Update)

Update a requord with Primary Key:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`/`**`PRIMARY_KEY`**

If the PK is composed of several columns or if the records are not to be identified via the PK, the query parameters must be used:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`?`**`COLUMN=VALUE`**`&`**`COLUMN2=VALUE2`**

The request requires a HashMap in JSON format. The column name is used as the key.

Example:

```
{ "ID" : "123",
  "HAUSNUMMER" : "0",
  "STIEGE" : "0",
  "TUERNUMMER" : "0"
}
```

It should be noted that PK columns are not updated.

PUT-Response

The response returns the complete record in JSON format:

```
{ "ID" : 0,
  "POST_ID" : 127,
  "POST_PLZ" : "1127",
  "STIEGE" : "0",
  "STRA_ID" : 68,
  "STRA_NAME" : "Strasse (69)",
  "HAUSNUMMER" : "0",
  "TUERNUMMER" : "0"
}
```

DELETE-Request (Delete)

Delete exactly one record:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`/`**`PRIMARY_KEY`**

If the PK is composed of several columns, the query parameters must be used:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**`?`**`PKCOLUMN=VALUE`**`&`**`PKCOLUMN2=VALUE2`**

DELETE-Response

The response returns the number of deleted records in JSON format (as number):

```
42
```

OPTIONS-Request (Meta Data)

Request Meta Data:
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/data/`**`STORAGE_NAME`**

OPTIONS-Response

The response returns the meta data in JSON format:

```
{ "autoIncrementColumnNames" : [ "ID" ],
  "columnMetaData" : [ { "allowedValues" : null,
        "autoIncrement" : true,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Id",
        "linkReference" : null,
        "name" : "ID",
        "nullable" : false,
```

```
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Post Id",
        "linkReference" : null,
        "name" : "POST_ID",
        "nullable" : false,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 12,
        "defaultValue" : null,
        "label" : "Plz",
        "linkReference" : { "columnNames" : [ "POST_ID", "POST_PLZ"],
                            "referencedColumnNames" : [ "ID", "PLZ"],
                            "referencedStorage" :
".subStorages.postleitzahlen"
                          },
        "name" : "POST_PLZ",
        "nullable" : false,
        "precision" : 2147483647,
        "scale" : 0,
        "signed" : false,
        "sqltype" : 12,
        "writable" : false
      },
      { "allowedValues" : null,
        "autoIncrement" : false,
        "dataType" : 3,
        "defaultValue" : null,
        "label" : "Stra Id",
        "linkReference" : null,
        "name" : "STRA_ID",
        "nullable" : false,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
```

```
    },
    { "allowedValues" : null,
      "autoIncrement" : false,
      "dataType" : 12,
      "defaultValue" : null,
      "label" : "Name",
      "linkReference" : { "columnNames" : [ "STRA_ID", "STRA_NAME"],
                          "referencedColumnNames" : [ "ID", "NAME"],
                          "referencedStorage" : ".subStorages.strassen"
                        },
      "name" : "STRA_NAME",
      "nullable" : false,
      "precision" : 2147483647,
      "scale" : 0,
      "signed" : false,
      "sqltype" : 12,
      "writable" : false
    },
    { "allowedValues" : null,
      "autoIncrement" : false,
      "dataType" : 3,
      "defaultValue" : null,
      "label" : "Hausnummer",
      "linkReference" : null,
      "name" : "HAUSNUMMER",
      "nullable" : false,
      "precision" : 10,
      "scale" : 0,
      "signed" : true,
      "sqltype" : 4,
      "writable" : true
    },
    { "allowedValues" : null,
      "autoIncrement" : false,
      "dataType" : 3,
      "defaultValue" : null,
      "label" : "Stiege",
      "linkReference" : null,
      "name" : "STIEGE",
      "nullable" : true,
      "precision" : 10,
      "scale" : 0,
      "signed" : true,
      "sqltype" : 4,
      "writable" : true
    },
    { "allowedValues" : null,
      "autoIncrement" : false,
      "dataType" : 3,
      "defaultValue" : null,
      "label" : "Tuernummer",
```

```
        "linkReference" : null,
        "name" : "TUERNUMMER",
        "nullable" : true,
        "precision" : 10,
        "scale" : 0,
        "signed" : true,
        "sqltype" : 4,
        "writable" : true
      }
    ],
  "columnNames" : [ "ID",
      "POST_ID",
      "POST_PLZ",
      "STRA_ID",
      "STRA_NAME",
      "HAUSNUMMER",
      "STIEGE",
      "TUERNUMMER"
    ],
  "primaryKeyColumnNames" : [ "ID" ],
  "representationColumnNames" : [ "ID",
      "POST_ID",
      "POST_PLZ",
      "STRA_ID",
      "STRA_NAME",
      "HAUSNUMMER",
      "STIEGE",
      "TUERNUMMER"
    ]
}
```


Call actions

The server-side actions can be called directly from the life-cycle object as well as from available business objects. It's also possible to use parameters.

- Action without parameter
- Action with parameter

GET-Request

Call a server-side action (without parameter):
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/action/`**`ACTION_NAME`**

Call a method from a business object (without parameter):
`http://server:port/webapp/services/rest/`**`APPLICATION_NAME`**`/`**`LIFECYCLE_CLASS`**`/obje`

ct/**OBJECT_NAME**/**ACTION_NAME**

GET-Response

The response returns the return value of the action in JSON format.

POST/PUT-Request

Call a serer-side action (with parameter):
http://server:port/webapp/services/rest/**APPLICATION_NAME**/**LIFECYCLE_CLASS**/action/**ACTION_NAME**

Call a method from a business object (with parameter):
http://server:port/webapp/services/rest/**APPLICATION_NAME**/**LIFECYCLE_CLASS**/object/**OBJECT_NAME**/**ACTION_NAME**

The request requires an array of objects populated with the parameters for the action.

Example:

Action:

```
public String calculate(Number pFirst, Number pSecond)
{
  return "" + pFirst.intValue() + pSecond.intValue();
}
```

JSON Request:

```
[123,1]
```

POST/PUT-Response

The response returns the return value of the action in JSON format.

Examples

Using php:

```
$ch = curl_init();
curl_setopt($ch,
CURLOPT_URL,'https://<server>/DB/services/rest/League/Standings/data/All');

curl_setopt($ch, CURLOPT_RETURNTRANSFER,1);
curl_setopt($ch, CURLOPT_HTTPAUTH, CURLAUTH_BASIC);
```

```
curl_setopt($ch, CURLOPT_USERPWD, "user:password");
//curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 5);

$json = json_decode(curl_exec($ch), true);

curl_close($ch);
```

AngularJS 4 with VisionX and JVx REST services
AngularJS with JVx in action
Oracle JET with VisionX/JVx

Note

For action calls, the correct data types must be used! In general, it is best to dispense with primitive data types, as parameters, and instead of using arrays, the List Interface should be used. It is also recommended to use Number for all numerical values. This avoids problems due to JSON serialization.

The life-cycle name should be the fully qualified class name, with package. If only the simple class name is used, JVx will try to find a matching class. If several classes are considered, then no class is used. You can optionally define a search path in the config.xml of the application:

```
<application>
  ..
  <rest>
    <search>
      <path>/com/sibvisions/app/myapp</app>
      <path>/com/sibvisions/app/myapp/screens/sub/</app>
    </search>
  </rest>
</application>
```

Additional information about this feature is available in our Support System.