# Table of Contents

***ServletServer*** ..... 1
***REST Services*** ..... 2
***Life-cycle Objects*** ..... 4
***Object-specific Calls*** ..... 4
***Environment-dependent Access*** ..... 4
***Checklist*** ..... 4

By default, care is already taken to ensure that object access is only possible with authentication. If this is not sufficient, additional security measures can be activated in JVx:

# ServletServer

The `ServletServer` is required for communication between a desktop client and the backend. You can completely disable the `ServletServer` if you do not use a desktop client or if you don't use `HttpConnection` directly.

To restrict the available functions, you can use the following in the config.xml of the Server Zone:

```
<security>
  <serializedaccess>
    <rules>
      <allow>Session!.*</allow>

      <deny>Session!.createSubSession(*)</deny>
    </rules>
  </serializedaccess>
</security>
```

With the above rules, you define that logging in to the server is possible, but creating a sub-connection is not. It is not possible to perform further calls successfully. The deny rules are applied before the allow rules!

### Rule Definition

A rule is defined by: **ObjectName.MethodName(*[param1[, param2]]*)**

- The square brackets define optional fields
- With wildcard notation, you can fine-tune access control

### More Control

If you need more control, e.g., because you require state-dependent access control, you can achieve this by implementing `ISerializedCommunicationAccessChecker`:

```
public class CustomAccessChecker implements
ISerializedCommunicationAccessChecker
{
    public void init(IServer pServer)
    {
    }

    public boolean isAccessAllowed(IRequest pRequest, ISession pSession,
String pObjectName, String pMethod,
                                   Object[] pParams,
```

```
ChangedHashtable<String, Object> pProperties, boolean pIsCallBack)
    {
        if (IConnection.OBJ_SESSION.equals(pObjectName))
        {
            return true;
        }

        if (pSession != null)
        {
            if ("myapplication".equals(pSession.getApplicationName())
            {
                if ("users".equals(pObjectName))
                {
                    return true;
                }
            }
        }

        return false;
    }
}
```

You then define the class in the `config.xml` of the Server Zone:

```
<security>
  <serializedaccess>
    <checker>j.v.x.CustomAccessChecker</checker>
  </serializedaccess>
</security>
```

You can define any number of `checkers`.

Another option is to load your checker using the standard Java Service Loader.

## REST Services

If you use REST Services to enable access to application data via standard HTTP, such as:

```
<servlet>
  <servlet-name>RestletServlet</servlet-name>
  <servlet-class>com.sibvisions.rad.server.http.rest.RESTServlet</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>
```

then you can disable individual services.

### Admin Services

Disable admin services in general:

```
<init-param>
    <param-name>adminservices.available</param-name>
    <param-value>false</param-value>
</init-param>
```

Admin services are enabled by default.

To disable individual services:

```
<init-param>
    <param-name>adminservice.NAME.enabled</param-name>
    <param-value>false</param-value>
</init-param>
```

Predefined Admin Services:

- checkDB
- changePassword
- testAuthentication
- memoryStatus

To disable all admin services:

```
<init-param>
    <param-name>adminservices.enabled</param-name>
    <param-value>false</param-value>
</init-param>
```

### Custom Admin Services

In JVx, it is possible to extend admin services. These are then called custom admin services. To disable them:

```
<init-param>
    <param-name>adminservice.NAME.enabled</param-name>
    <param-value>false</param-value>
```

```
</init-param>
```

## Life-cycle Objects

Business logic is defined in Life-cycle objects. Default security ensures that all public methods are callable. If you change a method to `protected` or `private`, it is no longer callable, e.g.:

```
protected IDBAccess getDataSource() throws Exception
{
    ...
}
```

However, if a method must be `public`, you can protect it from external calls using an annotation:

```
@NotAccessible
public IDBAccess getDataSource() throws Exception
{
    ...
}
```

Details on `@Accessible` and `@NotAccessible` can be found here.

## Object-specific Calls

If you want to control specific accesses in your application, you can achieve this by implementing IObjectAccessController. This interface is checked every time a Life-cycle Object is called. This allows fine-grained control at the instance level.

Access control for REST only works for REST calls, and access control for `ServletServer` only works when an `HttpConnection` is used. Using the IObjectAccessController works independently of this.

## Environment-dependent Access

If you work with Accessible and NotAccessible, you can also control access per environment, as described here.

## Checklist

- Disable `ServletServer` or restrict calls
- Completely disable REST Services or only enable individual services
- Check Life-cycle Objects (public, protected, private) and verify the use of `@NotAccessible`
- Environment-dependent use of `@Accessible` and `@NotAccessible`

From:
https://doc.sibvisions.com/ - **Documentation**

Permanent link:
**https://doc.sibvisions.com/jvx/security**

Last update: **2026/11/11 12:27**