Table of Contents

JVx is generally used to develop multi-tier applications with an emphasis on databases. To facilitate data exchange between client and enterprise tier, a transport layer is required. This transport layer not only has a very abstract definition in JVx, but an implementation based on http(s) already exists.

The transport or communication layer was designed protocol-independent. It is, therefore, possible to use the communication classes for different protocol implementations without adaption.

The following is an example of protocol-independent use:

```
HttpConnection con = new HttpConnection(URL);

MasterConnection macon = new MasterConnection(pConnection);
macon.setApplicationName("demo");
macon.setUserName("demo");
macon.setPassword("demo");
macon.open();
```

The http protocol defines that a web or application server has to be used. This makes development more difficult as an application server always has to be launched before the actual business logic can be tested. Of course, lightweight application servers are available such as Jetty or the Eclipse WTP with integrated Tomcat support. Nonetheless, relationships have to be considered and configurations have to be made. This is something we would rather live without, especially the search for communication errors. The fewer components have to be considered, the easier the search will be.

To support the developer as much as possible, JVx includes the VMConnection and the DirectServerConnection in addition to the HttpConnection.

The VMConnection can be seen as equivalent to the HttpConnection with the only difference that the communication happens without an application server and without http protocol. The server is automatically started in the current VM and, just as with the HttpConnection, is accessed via streams. The transferred objects are always serialized.

This type of communication makes the developer´s work much easier, but there is an even better way.

Both the VMConnection and the HttpConnection serialize and de-serialize the objects. However, if the server is running on the same VM as the client, serialization would not be necessary.

In this case, the DirectServerConnection is used, which means that server functions are called directly via the server class and objects are passed on to methods directly. There is no serialization, which positively affects performance.

Note

At no point does the developer have to take the communication protocol into account as it is encapsulated by JVx. However, we have to be aware of the difference in operation between DirectServerConnection and VMConnection or HttpConnection as the object serialization has to be considered during troubleshooting. In addition, during a VMConnection or a DirectServerConnection with implicitly started server, the server is stopped when the application is stopped.

During the development process, we recommend the use of the DirectServerConnection. If special

serializers have to be implemented, or serialization errors have to be debugged, there is no way around the VMConnection.

During productive operation, the HttpConnection or a special IConnection implementation is required.