Table of Contents

A standard JVx application requests data from the server side via Connection. This concept doesn't support sending messages from the server to the client side.

In JVx, we have a keep alive mechanism and this usually checks every 30 seconds if the connection to the server-side is still valid. This alive check can be used to send properties to the client using connection properties. It's also known as server side polling mechanism. But this mechanism isn't the best approach for sending any kind of information to the client side.

To solve this problem, we introduced the Callback Broker on server side. This broker is available for any session via SessionContext. It makes it possible to send any object to the client-side. The client-side is able to listen to so called call-back calls and, e.g., change the UI or trigger data updates.

A callback call is not the same as an async call. The difference is that an async call will be triggered from the client side and starts a new thread on server side for the execution. A callback call is triggered from the server side and sends information to the client side.

If you publish a message, it's not guaranteed that the message will be pushed immediately to the client side. This is technology-dependent, e.g., it will work immediately with Vaadin UI because websockets are supported. If the technology doesn't support push, the pull mechanism via alive check will be used. So it's guaranteed that the client side receives the message as soon as possible.

If you want to use the publish mechanism, simply register a listener on your connection:

```
ICallBackResultListener listener = new ICallBackResultListener()
{
    public void callBackResult(CallBackResultEvent pEvent) throws Throwable
    {
        if ("COUNT_ADD".equals(pEvent.getInstruction()))
        {
            counter += ((Integer)pEvent.getObject()).intValue();
        }
    }
}

AbstractConnection con = getConnection();
con.addCallBackResultListener(listener);
```

To send a message, simply call:

```
SessionContext.getCurrentInstance().getCallBackBroker().publish("COUNT_ADD",
Integer.valueOf(1));
```

on the server side.

But be careful if you run in a thread because the broker won't be available outside the server execution thread. So, use a cached instance of the broker, e.g.:

```
final ICallBackBroker broker =
SessionContext.getCurrentInstance().getCallBackBroker();

Thread th = new Thread(new Runnable()
{
```

```
    public void run()
    {
        //... server logic

        broker.publish("COUNT_ADD", Integer.valueOf(i++));
    }
});

th.start();
```

The ICallBackBroker interface defines an enum for the PublishMode:

```
public enum PublishMode
{
    /** only the current session. */
    CurrentSession,
    /** other sessions from the current master session. */
    OtherSessions,
    /** only the master of the current session. */
    CurrentMasterSession,
    /** all sessions which are like the current session, and the current
session. */
    AllCurrentSessions,
    /** all sessions which are like the current session, but not the current
session. */
    AllOtherSessions,
    /** all master sessions. */
    AllMasterSessions,
    /** all master sessions but not the master session of the current
session. */
    AllOtherMasterSessions
}
```

If you don't use a specific mode, the CurrentSession mode will be used as default setting, like in the above example.