**Changelog:**

|  |  |  |
| --- | --- | --- |
| ***Date*** | ***Author*** | ***Description*** |
| 2010-02-23 | PH | Separation of the client-side communication component documentation as a separate working document |
| 2010-02-26 | KK | Description of the customer's façade |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Client-side communication component

The SBDR client component performs all operations involving placing data in the database, modifying it and retrieving it. A detailed list of services offered by the server can be found in the document: TD\_UslugiBD.doc. A list of operations available to users in: Prototyp\_usecases\_biezace.doc

The primary place to use a communication component is a client applet. However, a wider use of the component is not ruled out.

The full source code of the communication component along with the libraries used is available in the BDR space svn on the assembla.com in the DatabaseConnection project: [https://svn2.assembla.com/svn/bdr/Aplikacje/DatabaseConnection/   
in the form of the Eclipse 3.4 project](https://svn2.assembla.com/svn/bdr/Aplikacje/DatabaseConnection/)

## Component Tasks

The primary goal of creating a separate communication component is to separate client applications from the database server at all possible levels. That is, at the communication layer, network protocols, server implementation, and any other technologies used on the server side. Thanks to this, the client and the user interface are completely independent of the server, treating the communication component as a black box responsible for performing the task of connecting and sending data, or receiving it from the server.

## Construction of the communication component

The component consists of three basic packages that divide it into three main modules:

**motion.database.ws** – contains the connection code to the services provided on the server. This module and all its submodules were generated automatically by the jax-ws utility. This module changes every time the form of communication is changed, when services are added, and so on. Each subpackage corresponds to one server-side Web Service. Each class represents a single service.

**motion.database –** contains the basic façade object – DatabaseConnection, which is responsible for forwarding all operations to the server. A more detailed description of this class will be provided in the next section.

**test.unit –** a testing module containing unit tests and examples of using all methods from the DatabaseConnection class.

## DatabaseConnection Fasadow Class

The façade is implemented according to the singleton pattern, but its instance is obtained by the ***getInstance() method.***

Due to the fact that Web Services are stateless, the number of façade states is quite limited and currently includes only: UNINITIALIZED and INITIALIZED. The first state is reached immediately after the system starts. The second state is the normal state in which operations can be performed on the server. The transition to this state occurs as a result of setting the connection string to the server. Currently, this involves calling two methods: ***setWSCredentials()*** and ***setFTPSCredentials()***.

The façade class completely isolates clients from the database server. In particular, it offers methods that, when called, result in the execution of appropriate operations (or sequence of operations) on the server. Errors are reported using exceptions.

Isolation from the server also means that any structural types do not depend on the structures returned by the server, but only on the façade. They are all defined in the ***motion.database*** package. Attribute values (static and generic) are represented as HashMaps. For example, the Session data façade class represents a server-side session object. Together with it, the **SessionStaticAttributes class is defined**, which contains an enumeration of static session attributes. This makes it easier to recover data from the session. The exact method of implementation and functionality related to the façade types of the database is currently being tested and researched. The reader is referred to the sample code in ***the test.unit*** module.

### Authorization of the connection to the server

The façade uses two channels to communicate with the server: Web Services and FTPS. Each of them requires separate authorization: user and password. Each time an operation is performed on the server, it is preceded by authorization.

The façade customer can change the authorization parameters at any time.

### Invoking Server Services

Calling services on the server is done by calling the methods of the DatabaseConnection façade class once. These methods hide all operations performed in the process of uploading data to the server, including the process of uploading or downloading files. If a method exits without throwing an exception, the operation is considered successful.