The tcpconns TCP/IP Communications Package for JAVA

includes certificate authority

(jdk 1.5+)

(C)2007-2009 by J.H.H.C. van der Meijs
tcpconns@jhhcvandermeijs.nl

tcpconns developer's manual

With tcpconns you can rapidly create multi-threaded TCP/IP server applications

This manual provides initial guidance on utilizing the tcpconns TCP/IP communications package, it is meant to be read from cover to cover

Icons used in this manual

 This icon indicates a note.

 This icon indicates advice that you SHOULD follow.

 This icon indicates things that you should NOT do.

 This icon indicates advice that is HELPFUL.

 This icon indicates (informational) resources that you might want to check out.

These icons come from the Tulliana-2.0 icon set, which can be accessed on www.kde-look.org (http://www.kde-look.org/content/show.php?content=38757); these icons are licensed under the LGPL license.

OpenOffice.org

This manual was written in openoffice.org, get it here.

Table of Contents

About

Why this package? This package prevents you from writing all essential layers necessary for robust TCP communication over and over. Even with high-level platform-independent classes that come with the standard java API libraries, do not underestimate the time and effort involved to produce an extensible reusable multi-threaded TCP/IP client/server solution that is also capable of delivering SSL/TLS. The tcpconns package has an intelligent and simple design: the tcpconns package allows you implement multi-threaded TCP/IP client/server applications in a straightforward and easy manner. The tcpconns package includes packages for certificate generation through a graphical user interface application, and also certificate conversion capability. For actual certificate generation tcpconns uses the bouncycastle.org package.

Tcpconns is Open Source Initiative approved Open Source Software. Open Source Initiative Approved is a trademark of the Open Source Initiative. http://www.opensource.org/

Why this manual? Proper documentation and annotated code is vital to any software library or application. Absence of adequate documentation might lead to misuse, data loss, and the mere fact that excellent code remains unused. **I am seriously lagging behind with the documentation, sorry for that, but I simply do not have the time. This manual should get you started without difficulty though, and if you follow the advice in this manual there should be no surprises.**

NOTE: Tcpconns currently is being developed inside the Netbeans 6.5.1 IDE on Ubuntu Linux 8.10. ***IT SHOULD RUN ON ANY PLATFORM THAT SUPPORTS JAVA*** *(JDK 1.5/JRE 5 and above).*

DO: Follow the instructions in this manual. This manual is meant to be read from cover to cover.

DO: Use this manual as a reference during the implementation of your protocol.

DO: Analyze the test packages that come with the complete tcpconns .zip package and do experiment on them. **Good programming practices dictate that you must thoroughly test new software before relying on it.**

DO: Install the sun.com unlimited strength policy files. To be able to use the SSL/TLS features in the default settings (namely TLSv1 with cipher suites TLS_DH_anon_WITH_AES_256_CBC_SHA and TLS_RSA_WITH _AES _256_CBC_SHA), and to be able to use the accompanying test projects, you need to have the **unlimited strength policy files** installed, otherwise e.g. the following exception may

occur: “cannot support TLS_RSA_WITH_AES_256_CBC_SHA with currently installed providers.”. The JCE unlimited strength jurisdiction policy files, together with installation instructions, can be obtained from www.sun.com (jdk5, jdk6).

NOTE: When you are recompiling the test packages with netbeans, you might need to reset the path to tcpconns1.jar (netbeans will automatically indicate reference problems; “solve them” in the way netbeans indicates by browsing to the tcpconns1.jar file).

NOTE: if you make use of the PEM reading functionality (or the certificate authority functions) then you need the bouncycastle provider jar from http://www.bouncycastle.org/ NOTE: bcprov-jdk15-xxx.jar is located in the /tcpconns/crypto-xxx/jars-folder; alternatively get the lastest cryptopackage from bouncycastle.org at http://www.bouncycastle.org/latest_releases.html. Either get: OR the crypto-xxx.tag.gz or crypto-xxx.zip and unpack it, OR get the bcprov-jdkxx-xxx.jar. To add the bcprov-jdkxx-xxx.jar to the tcpconns project (if you need to solve reference problems) or to your own project, then please do the following (also see the installation instructions elsewhere): right-click on the project’s name inside the netbeans ide and select “Properties” from the PopUp menu that has appeared; in the left tree view select “Libraries” and then go to the tab “Compile” on the right. There, remove any bcprov-jdkxx-xxx.jar files if they are broken (if applicable); press the “Add JAR” button, browse and select your freshly downloaded bcprov-jdkxx-xxx.jar; and then press OK down below. Now if there are building issues, they may not be apparent immediately. If red exclamation marks appear next to the source files, the most likely explanation is that some classes cannot be found (“cannot find symbol”). If this is the case make sure you haven’t selected the wrong .jar file by mistake (it happens). There should be no errors.

NOTE: If you wish to run the CertAuth program or the CertConvert program you must start their jars by double-clicking on them (provided you have correctly installed java on your machine and have the correct file associations in place; alternatively type the following (or something similar if you start a different .jar) on the command line in a terminal/dosbox: java -jar "CertAuth.jar"). The .jar is located in the /dist-folder of the respective application. If you want to copy the program to another location, then please do not forget to move the /lib-folder as well (!). Due to class-path definitions in netbeans you cannot have all .jars in a single folder; double-clicking on the .jar will then raise an error which is not entirely accurate. The error reads: "Could not find the main class. Program will exit", while the main class indeed is defined properly. The error is raised because the main class could not be loaded because the java virtual machine could not find the dependency jars (which are stored in the /lib folder).

NOTE: Javadoc documentation can be easily generated from within netbeans, just right-

click the project and select the appropriate item from the pop-up menu

NOTE: for the certificate authority's look and feel, at least in case of reference problems, please download the latest nimrodlf.jar-file from the nimrod website; right-click on the project's name; select Properties from the popup menu; select Libraries in the treeview on the left; and under the tab Compile press the Add JAR button and add the nimrodlf.jar-file to the project. NOTE: in the folder /tcpconns/nimrodlf a functional nimrodlf.jar file is available. NimrodLF is distributed under the conditions of the LGPL.

LINK: SocketTest (sourceforge.net) might be helpful to you when you are implementing your protocols. SocketTest "can be used to test any server or client that uses TCP or UDP protocol to communicate".

Legal

THIS SOFTWARE IS RELEASED UNDER THE FOLLOWING BINDING LEGAL AGREEMENT (BSD-LICENSE).

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of tcpconns nor the names of its copyright holders or its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT

LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF YOU DO NOT AGREE TO THE ABOVE TERMS AND CONDITIONS THEN DO *NOT* USE THIS SOFTWARE AND *DO* DESTROY ALL COPIES OF THIS SOFTWARE THAT YOU HAVE.

If you make changes to the code, it is appreciated that you send your modifications to tcpconns@jhhcvandermeijs.nl so that there will be only one canonical version of the tcpconns library.

TCPCONNS' CURRENT CODE IS NOT KNOWN TO VIOLATE ANY COPYRIGHTS OR INFRINGE ON ANY PATENTS. MOREOVER, IT ADHERES TO GENERAL PROGRAMMING RULES AND PRACTICES AND IS OF SUCH A SIMPLISTIC NATURE THAT A SIMILAR PACKAGE COULD BE CONSTRUCTED BY ANYONE.

IMPORTANT: IF YOU FEEL THAT A PIECE OF CODE IN TCPCONNS VIOLATES A COPYRIGHT OR INFRINGES ON A PATENT, PLEASE DO INFORM ONE OF THE AUTHORS. DO ALWAYS STATE EXACTLY WHAT CODE IS NOT ALLOWED UNDER EXACTLY WHAT RULING, COPYRIGHT, OR PATENT; AND PLEASE DO ELABORATE ON EXACTLY HOW THE CODE VIOLATES THE SPECIFIC COPYRIGHT OR INFRINGES ON THE SPECIFIC PATENT.

N.B. Tcpconns uses the jars from the www.bouncycastle.org project.
N.B. Tcpconns includes a method Utils.getJarName which contains code originally written by "vafarmboy" (assumed to be public domain).
N.B. Tcpconns includes a method Utils.getNetworkIp which contains code originally written by Marcello de Sales (assumed to be public domain).
N.B. FTPServer includes code from the danoFTP project (http://sourceforge.net/projects/danoftp), which comes under a BSD license.
N.B. Tcpconns includes a class Utils.Base64 which is Copyright (c) 2004, Mikael Grev, MiG InfoCom AB (BSD license).

Tcpconns is Open Source Initiative approved Open Source Software. Open Source Initiative Approved is a trademark of the Open Source Initiative. http://www.opensource.org/.

History of Changes

- 20091004 Several updates. Some package reordering. Removed CertAuth => all certificate authority functionality is now available in the new KeyStore Manager GUI. Regenerated sample certs. Upgrading is recommended. Various improvements to the KeyStore Manager GUI. Now it supports generating/verifying OpenPGP signatures with X.509 keysets. Added functionality that allows you to import a PGP keypair, i.e. that allows you convert a master/identity keypair into an X.509 keypair. Signatures generated with this conversion are completely exchangeable with signatures generated by GnuPG/PGP with the full original PGP keypair. N.B. only support for RSA keypairs. Minimal updates to documentation. Some package reordering.
- 20090920 Created a magnificent class named KeyStoreHelper, that will allow you to read and write pkcs#12 (.pfx/.p12), .jks, .bks, .uber, .jceks, and .pem keystores. Rewrote the CertConvert utility. Created a second magnificent class named X509PropertiesDialog, which can be used easily to display the properties of a X509 certificate (with or without private key, with or without the entire X509 certificate chain); the X509PropertiesDialog allows the user to export it (with or without private key: in .pem format) or, alternatively, to store it to disk (without private key: in .der/.cer format, with private key: in .pfx/.p12 format); X509PropertiesDialog warns if the X509 Certificate is outside of its validity period. Added X509CertificateSelectionDialog and KeySelectionDialog components. Created a new KeyStoreManagerGUI component that is much more powerful than the rewritten CertConvert utility, so I've decided to remove the CertConvert utility. The new KeyStoreManagerGUI is located right in tcpconns.jar, so when you run the tcpconns.jar, it will fire up the KeyStoreManagerGUI. And it is a GUI I love!, and yes.....so will you!!! Have fun! ...More is to follow, stay tuned!
- 20090607 (internal build). Some reordering in the ftp project. Added new methods for SSL/TLS in tcpconns (there are now methods that accept an SSLSocketFactory as parameter). Tested all possible FTPServer TLS settings against the FileZilla client. Tested FTPServer on Ubuntu Server 9.04/EXT2/SunJDK6 in a local network configuration against FileZilla.
- 20090601 (2nd beta of the Next Release). Added an XMLFileStore class that can be used to create and read settings in an .xml file. Placed the configuration settings for FTPServer inside an .xml file; FTPServer now has more advanced user rights management. Added a GUI application for FTPServer. Renamed the original ftpserver project to ftp, now the GUI is named FTPServer. Reorganized and rewrote large parts of FTPServer. The new version with GUI and advanced user management works on the systems listed below. Documentation will follow. In conclusion: I'm on my way,... not there yet, but I *am* getting there!
- 20090510 (Beta of the Next Release). Small code rewrites, some reordering, added a method getJarName to the Utils class. Cross platform tested FTPServer locally against the latest FileZilla releases between/on Ubuntu Desktop 9.04/EXT4/SunJDK6, PCBSD7.1 (FreeBSD7.2PRE)/UFS2/SunJDK6, and Windows Vista SP1 Home Premium/NTFS/SunJDK6. Formal analysis of RFCs, a FTPServer GUI, and a corresponding FTPClient will all follow at a later date. For now you are encouraged to stress test FTPServer on all platforms that support version 1.5/5 or 1.6/6 of the java runtime environment (JRE) or java development kit (JDK); don't forget to install the unlimited strength policy files if you wish to use the implicit TLS feature of FTPServer.
- 20090508 (7th Alpha of the Next Release). Rewrote and reorganized large parts of ftpserver.

Added implicit TLS.

- 20090503 (6th Alpha of the Next Release). Added usermanagement to ftpserver.
- 20090502 (5th Alpha of the Next Release). Various improvements to ftpserver.
- 20090421 (4th Alpha of the Next Release). In ftpserver: do GetCanonicalPath when setting userdir! (a testdrive on PCBSD7.1 revealed this small bug).
- 20090419 (3rd Alpha of the Next Release). Various small additions and small code improvements. Added an ftpserver project that demonstrates how you could use tcpconns; I borrowed some code from danoftp (http://sourceforge.net/projects/danoftp), which comes under a BSD license; my intention is to develop this ftpserver project further.
- 20090405 (2nd Alpha of the Next Release) Redesigned the certificate conversion application. Some bugfixes. SRNGSeeder was augmented with a method that automatically tries to create a random seed file in the user directory of the current user; the method tries to obtain random bytes from the hotbits website. Needless to say an active internet connection is necessary for this to happen; if an active internet connection isn't possible, it falls back on the default secure random implementation to seed the file. Alternatively, one can create or download separately one's own random seed file; then place the file in the user directory with the filename "tcpconnsSRNGSeeder.rnd"; that file will then subsequently be used. The CertAuth program now utilizes this functionality, but you can still select any random seed file.
- 20090323 (1st Alpha of the Next Release) For reasons of clarity/legibility/comprehensibility, I've decided to place the tcpconns classes in different subpackages.
- 20090322 Upgraded the bouncycastle jar to version 1.42
- 20080607 Application logic for CertAuth and CertConvert were updated. License notifications now also in place. Updated the documentation.
- 20080602 Added a CertConvert program that allows you to easily convert a .pem into a .pfx and vice versa inside a GUI interface. Also allows distillation of .cer (actually .der)-encoded certificates from both .pem and .pfx (which can be imported into e.g. a webbrowser and be used as a trusted certificate authority certificate to authenticate server certificates).
- 20080602 Removed the Extended Key Usage from RootCACertGenerator and CertGenerator. This so that the generated X509 certs can be used by e.g. the apache webserver (XAMPP) and mozilla firefox amongst others. If you are running XAMPP on linux then you should store the generated cert as server.crt into the /opt/lampp/etc/ssl.crt-folder (root access) and the generated rsa private key (unencrypted) as server.key into the /opt/lampp/etc/ssl.key-folder (root access). Please make sure that the server.key-file is only readable to as root user (if the server.key file is accessible by anyone else than root, then security is likely to be compromised if server.key is read by the wrong party).
- 20080601 CertAuth: now certs can be generated with the private key stored unprotected (unencrypted raw save to disk)
- 20080531 In CertAuth a java.lang.NullPointerException was raised when SRNGSeeder.addRandom() was being called. This only occurred when executing the CertAuth program inside M$ Window$. The problem was located in the SRNGSeeder.java file, and is now fixed. This 'bug' however did not pose a serious threat to the secureness of the random numbers generated since each time you create a new SecureRandom instance through SRNGSeeder.newSecureRandom(), the random file is rehashed. Any certificates that were generated on a windows machine are still secure.
- 20080120 added two classes that will enable the user to convert between .pfx (PKCS#12) and .pem (OpenSSL) encoded certificates and vice versa. Use is quite straightforward.

- 20080119 tcpconns is now included with a very very basic certificate authority that is able to generate 4096 bit RSA certificates which can be used for most if not all purposes; thanks to http://www.bouncycastle.org/. The certificate authority application makes use of the nimrod look and feel (LGPL license).
- 20080119 tcpconns now includes functionality to use PEM encoded (OpenSSL) certificates; thanks to http://www.bouncycastle.org/ (MIT X Consortium License)

Bug reports

Bugs always occur, no matter how hard you try to avoid them. If you think that a bug is present in the tcpconns library, please do notify the developer team at tcpconns@jhhcvandermeijs.nl. While bugs are often reproducible, sometimes the circumstances in which they occurred are very hard to duplicate.

- **DO**: check that your code is conform the instructions in this manual.
- **DO**: make sure that the bug you discovered is not a direct consequence of code modifications that you might have made to the tcpconns code itself.
- **DO**: try to reproduce the exact circumstances in which the bug occurred. If at all possible, try to write a simple program that demonstrates the bug.
- **DO**: please include information on your system hardware, host operating system, your jdk version, and the ide in which you develop your solution (if any).

Feature requests

If you require some functionality and are unable, incapable, or unwilling to implement it yourself, you may send your feature requests to tcpconns@jhhcvandermeijs.nl. On the other hand, if you have the time, energy, and expertise, it is very much appreciated that you help implement feature requests.

Contributions

Your involvement is very much appreciated! You can contribute in many ways. You can provide art, (re)write portions in this manual, (re)write javadoc comments, (re)write code, provide legal assistance, test the package on different operating systems, and/or make financial donations. You don't want to contribute? **That's fine too**, there are no obligations!

REQUEST: send your (annotated) modifications, enhancements, and/or improvements, whether they pertain to code or documentation, to tcpconns@jhhcvandermeijs.nl so that everyone can make full use of these modifications.

REQUEST: if you have successfully implemented a higher level protocol on top of tcpconns then please consider donating it to the tcpconns project so that many may benefit from your code.

For your code contributions enclose the following declaration:

> ***"I declare, that to the best of my knowledge, my current contribution(s) to tcpconns: do(es) not violate any copyright(s), do(es) not infringe on any (pending) patent(s), and: do(es) not violate any non-disclosure agreement(s) that I am a party to. I give permission that my current contribution(s): may be rewritten and/or reordered, and: may be released as public domain software. I understand that my contribution(s) might not ever be published as part of the tcpconns library."***

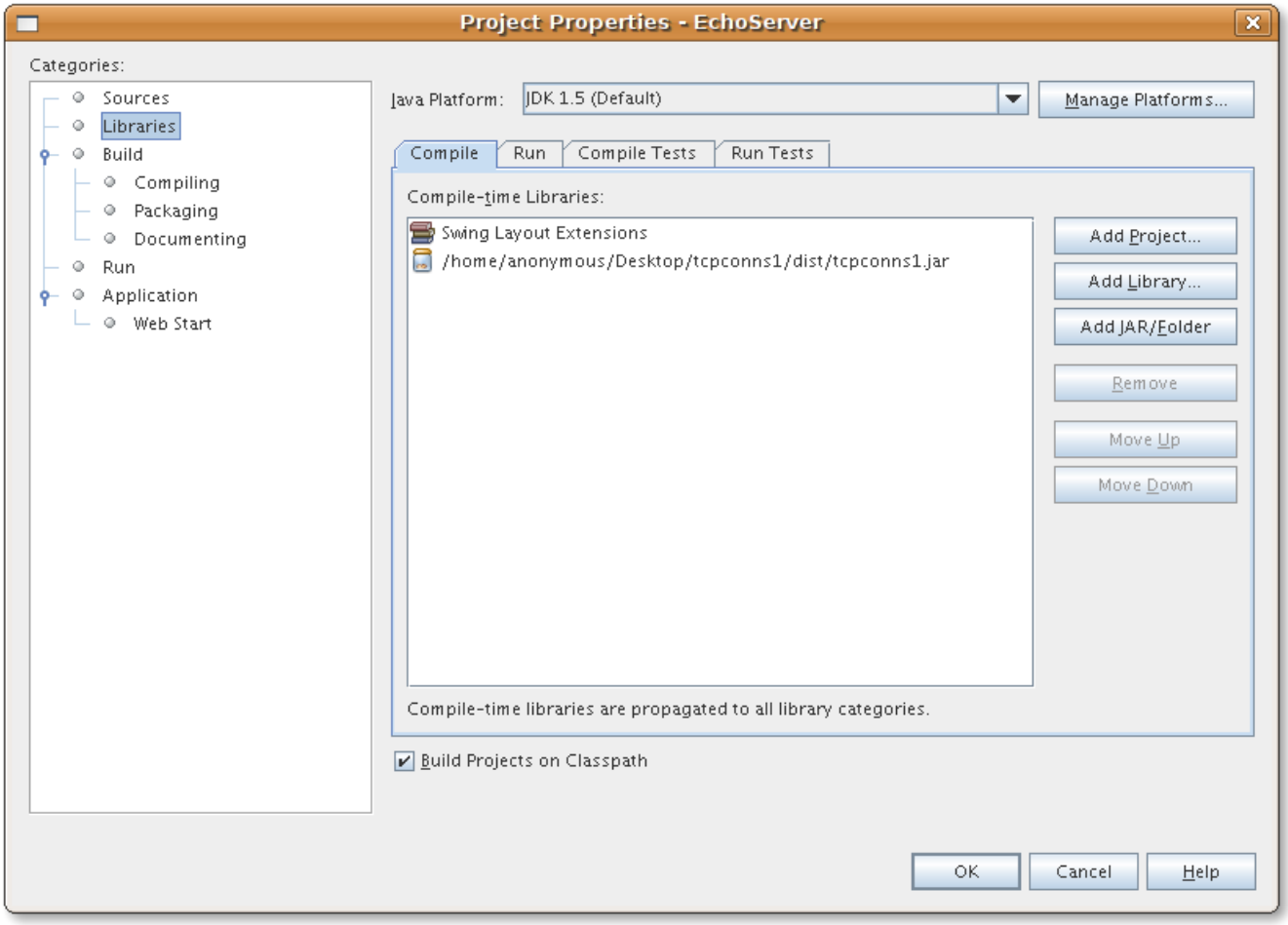
Installation

Requirements

- sun jdk 1.5 (or higher) **N.B. (!): please make sure that you also install the unlimited strength policy files if you wish to use the default settings of SSL/TLS in your project**
- netbeans (**highly** recommended; you need the jdk 1.6 or higher to run the latest netbeans.org)

3 Steps

- install this software package in a folder of your choosing
- add the tcpconns package to your project:



  - in netbeans, right-click on your project in the project list of the project tab; select properties from the pop-up menu; or select the project and then choose > File > “yourproject” properties from the main menu
  - select libraries in the tree on the left

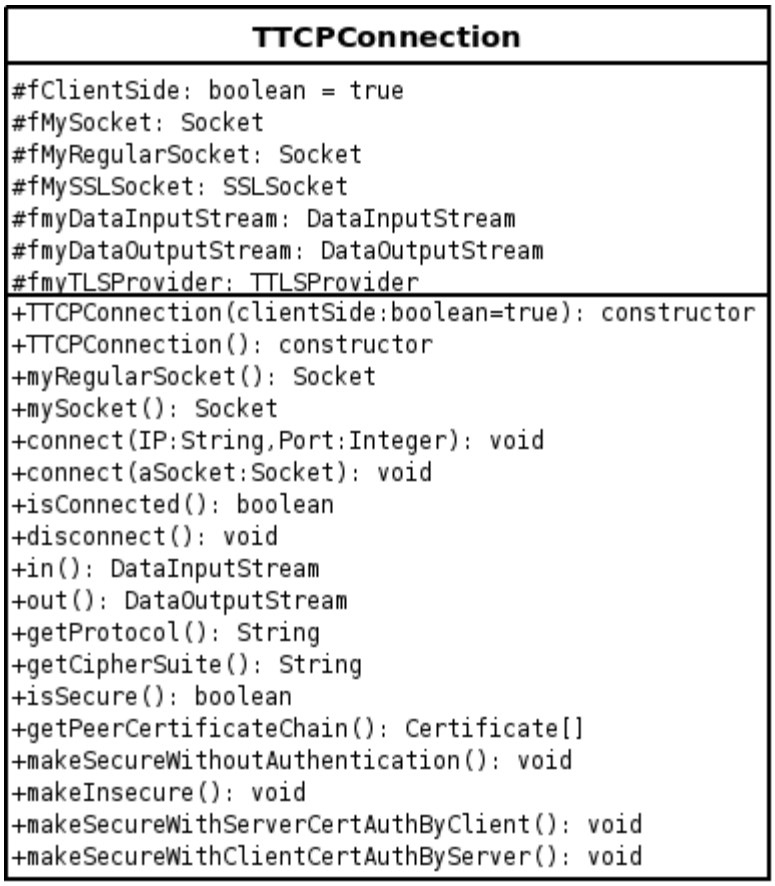
    - go to the tab “Compile” (default)
    - press the “Add JAR/Folder” button
    - then browse to the tcpconns1.jar and select it
    - the tcpconns1.jar should now appear in the list
    - (follow the same procedure for bcprov-jdkxx-xx.jar and/or nimrodlf.jar if applicable)
    - press the “OK” button
    - (re)build your project
    - the tcpconns functionality should be importable (available) now
- start using tcpconns

Essential classes

For instructions on how to use these essential classes, please see "Non-SSL/TLS implementation" below.

TTCPConnection

Use this class to establish a client connection to a server.



TServerSideProtocol

This class serves as the parent class to the implementation(s) of your server side protocol(s).

TTCPServer

Implements a multi-threaded tcp server.

Non-SSL/TLS implementation

Checklist implementation (client-side)

- create a TTCPConnection object:

```
TTCPConnection aTTCPConnection = new TTCPConnection();
```

- connect to the server:

```
aTTCPConnection.connect("127.0.0.1", 4000);
```

- verify connection and communicate with the server:

```
try {
      if (aTTCPConnection.isConnected()) {
            aTTCPConnection.out().writeInt(2);
            aTTCPConnection.out().flush();
            int a = aTTCPConnection.in().readInt();
      }
} catch (Exception e) {
      ...
} finally {
      ...
}
```

- disconnect when done:

```
aTTCPConnection.disconnect();
```

Checklist implementation (server-side)

- extend TServerSideProtocol:

```
class YourServerSideProtocol extends TServerSideProtocol {}
```

- write constructor with super:

```
public YourServerSideProtocol(
           TTCPConnection aTTCPConnection,
           TTCPServer aTTCPServer
         ) {
      super(aTTCPConnection, aTTCPServer);
}
```

- override processRequest so that it contains your protocol interpreter code (allow IOExceptions to surface beyond processRequest, i.e. if exceptions are caught, rethrow them):

```
@Override
public void processRequest() throws IOException {
      int a = this.conn().in().readInt();
      this.conn().out().writeInt(a);
      this.conn().out().flush();
}
```

- extend TTCPServer:

```
class YourServer extends TTCPServer {}
```

- override newServerSideProtocol:

```
@Override
synchronized public TserverSideProtocol
           newServerSideProtocol(
                TTCPConnection aTTCPConnection,
                TTCPServer aTTCPServer
           ) throws IOException {
      return new YourServerSideProtocol(aTTCPConnection,
                                        aTTCPServer);
}
```

- create an instance of your server class:

```
YourServer aYourServerInstance = new YourServer();
```

- set timeout value and maximum number of clients prn:

```
aYourServerInstance.setTimeOut(10*1000);
aYourServerInstance.setMaxClients(256);
```

- activate your server class (adjust firewall if necessary):

```
aYourServerInstance.activate(“0.0.0.0”, 4000);
```

- deactivate or deactivateNow when done:

```
aYourServerInstance.deactivate();
aYourServerInstance.deactivateNow();
```

Communicate over the connection

Communication is quite straightforward.
On the client side simply use the methods of **TTCPConnection.in()** to read from the server and **TTCPConnection.out()** to write to the server. When writing to the server do not forget to call **TTCPConnection.out().flush()** to force the sending of any buffered data. The original Socket is available though **TTCPConnection.mySocket()**.
On the server side, the TTCPConnection is available through **this.conn()**. Then use as you would use TTCPConnection on the client side; however, when the client writes, the server must read and vice versa. Also: when done writing to a client do not forget to call **this.conn().out().flush()** to force the sending of any buffered data. Of course, the original Socket is available here too, if you should need it.

Adding SSL/TLS

For the purpose of simplicity SSL = TLS and TLS = SSL.
In SSL each party can or cannot present its credentials (in the form of X509 certificates), and each party can or cannot verify the credentials of the other party if they are presented.

X509 Presentation		***X509 Verification***		***Method to be used***	
Client	***Server***	***Server By Client***	***Client By Server***	***Client***	***Server***
NO	NO	NOT AVAILABLE	NOT AVAILABLE	*Make Secure Without Authentication*	*Make Secure Without Authentication*
NO	YES	CLIENT SHOULD CHECK SERVER CERTIFICATE VALIDITY AND REVOCATION STATUS	NOT AVAILABLE: ON SERVERSIDE CLIENT CREDENTIALS CANNOT BE CHECKED	*Make Secure With Server Cert Authentication by Client* [***OR*** Make Secure Without Authentication]	*Make Secure With Server Cert Authentication by Client*
YES	YES	CLIENT SHOULD CHECK SERVER CERTIFICATE VALIDITY AND REVOCATION STATUS	SERVER SHOULD CHECK CLIENT CERTIFICATE VALIDITY AND REVOCATION STATUS	***Make Secure With Client Cert Authentication by Server***	***Make Secure With Client Cert Authentication by Server*** (X509 presentation by client is enforced: no X509 == no connection)

Unauthenticated SSL/TLS

The addition of unauthenticated SSL/TLS is a breeze: on either side simply give the order makeSecureWithoutAuthentication:

- client: `aTTCPConnection.makeSecureWithoutAuthentication();`
- server (TServerSideProtocol descendant): `this.conn().makeSecureWithoutAuthentication();`

To stop the current SSL session without closing the connection, call makeInsecure:

- client: `aTTCPConnection.makeInsecure();`
- server: `this.conn().makeInsecure();`

Call `isSecure()` to verify whether you are in secure mode or not.

Authenticated SSL/TLS

Authenticated SSL/TLS involves the utilization of X509 certificates and keys.

Presentation of an X.509 Certificate

For public&&private key storage files (file needs to include both) do something like this:

```
this.conn().makeSecureWithClientCertAuthByServer(

                KeyStoreHelper.createKeyManagers(
                     ((EchoServer)(this.fTTCPServer)).fKeyStoreFileName,
                     ((EchoServer)(this.fTTCPServer)).fKeyStorePassphrase
                )

          ); //client MAY present X509; server MUST present X509
```

fKeyStoreFileName may point to a keystore of the following types: pkcs#12 (old or new type), openSSL's .pem format, jceks, jks, bks, and uber.

or:

```
this.conn().makeSecureWithServerCertAuthByClient(

                KeyStoreHelper.createKeyManagers(
                     ((EchoServer)(this.fTTCPServer)).fKeyStoreFileName,
                     ((EchoServer)(this.fTTCPServer)).fKeyStorePassphrase
                )

          ); //client MUST present X509; server MUST present X509
```

fKeyStoreFileName may point to a keystore of the following types: pkcs#12 (old or new type), openSSL's .pem format, jceks, jks, bks, and uber.

Verification of a peer X509 Certificate with a .pem CA X509 Certificate

After having established an SSL connection, do something similar like:

```
java.security.cert.Certificate[] peerCertChain =
                        this.conn().getPeerCertificateChain();

//check whether we have a peerCertChain
if (peerCertChain == null) {this.conn().disconnect(); return;}

//check whether the peerCertChain contains certs
if (peerCertChain.length == 0) {this.conn().disconnect(); return;}

//get certificate
if (!(peerCertChain[0] instanceof X509Certificate))
    {this.conn().disconnect(); return;}
X509Certificate anX509 = ((X509Certificate)(peerCertChain[0]));

try {

    //check/obtain trusted CA certificate
    if (this.fTrustedCA == null) { //X509Certificate fTrustedCA
        this.fTrustedCA =
              KeyStoreHelper.loadX509(this.edtCACert.getText());
              //.cer/.der-formatted OR .crt/.pem-formatted
    }
    if (      KeyStoreHelper.isValid(
                   ((X509Certificate)(peerCertChain[0])),
                   this.fTrustedCA)
        ) {

    //cert is valid: check for revocation status (!!!) here

     } else {
        this.conn().disconnect();
        return;
     }
  } catch (Exception e) {
     this.conn().disconnect();
     return;
  }
```

Other classes

TTempFile

Use this class to obtain a temporary file handle.

Design

Listener Thread

Once the server is started, a dedicated, isolated, and code-safe listener thread is constantly waiting for incoming client connections. When a client connects, the listener thread will create a separate TServerSideProtocol object for the incoming client: objects of TServerSideProtocol contain the actual code that serves the client. The listener thread subsequently hands every TServerSideProtocol object over to the server's thread pool, which will schedule it for execution. Each client thus corresponds to one single separate TServerSideProtocol object that runs in isolation from other TServerSideProtocol objects. Although they run separate, they must still recognize each other's existence (prn) and work in cooperation when accessing **shared resources** to prevent a total foul up.
Why is there only one listener thread running? The reason you'd want to have multiple listener threads (probably) is to be able to allow more than one client to connect simultaneously. The java API ServerSocket class is created with a "backlog" property, which allows it to sequester all incoming clients who aren't officially connected yet. These connection requests are thus kept and handled immediately when the listener thread comes back to get the next client in line. This allows multiple incoming connections to be handled in a proper way. Thus, theoretically, more than one listener thread is not necessary. In practice, stress testing has indicated that if more than 512 clients simultaneously try to connect they all will connect and all will be adequately handled within a few seconds without losing their connection status prematurely.
If there is only one listener thread running, there is a risk that if the thread dies unexpectedly that incoming clients will no longer be served! This is true. However, the code inside the listener thread is safe: it is very unlikely to fail you. So the reason for listener thread termination is to be sought outside of the listener thread itself. Possible reasons for unexpected listener thread termination are: that the server software is not working properly, or that the java virtual machine is not working properly, or that the host operating system is not working properly, or that the machine hardware is not working properly. If the listener thread is to fail in such cases it is probably not the worst thing that could happen; in fact, it is something you'd very much want to happen. If you wish, you can regularly check whether your server is still alive. You have to implement this yourself. It is advisable that you check your server's availability from a machine in a different network as your server (so that it checks server availability in general and not just local availability). If repeated polls indicate that the server has failed, you can let it take the action that you desire (such as sending an email). The reason that this functionality is not provided with the tcpconns package is simple: if the listener thread fails then clearly physical intervention is necessary. Depending on a local thread to send an email might give you a false sense of server availability, since it is very unlikely that such an action could be undertaken if the system is unable to keep the listener thread up and running. It is (thus) recommended that you test your server's availability from another machine.