



AGETOR®

AGETOR Connection
Configuration

Content

1	Prerequisites.....	3
2	Introduction.....	3
3	Basic concepts.....	3
3.1	Overview of default setup	3
4	Configuration basics.....	4
4.1	Properties.....	4
4.2	Socket configuration files	5
4.3	SSL Debug information	5
5	Socket configuration file format	5
5.1	sockets.....	5
5.1.1	sockets/connection-factory	5
6	Assigning socket types to Agetor components	7
6.1	Default set up in sockets.cfg	7
6.2	Overriding CID's in socket.cfg	8
6.3	Overriding CID's in component configurations.....	9
6.3.1	Broker.cfg	9
6.3.2	Service runner configuration	10
6.4	Overriding CID's on command line for a component	10
6.5	Providing a specific CID to the InternalORB class.....	10
7	Working with certificates.....	11
8	Factory builder implementations.....	11
8.1	dk.bording.inside.sockets.PlainSocketFactoryBuilder	11
8.2	dk.bording.inside.sockets.SSLSocketFactoryBuilder	11
9	Examples.....	12
9.1	SSL based Broker to service communication	12
9.2	SSL based Broker to broker firewall tunnelling.....	13
10	Socket factory log file.....	14
11	Troubleshooting	15
11.1	Use compatible socket types	15
11.2	Beware of invalid or expired certificates	15
11.3	Wrong certificate keys	15
12	References.....	15

1 Prerequisites

This document describes the communication options available for configuration of Agetor component communication. The document requires an understanding of the basic Agetor components such as the *Agetor Broker* and *services*. Some basic knowledge on secure sockets (SSL) and certificates is assumed.

2 Introduction

An Agetor based system relies on multiple potentially distributed components that communicate using network connections. Agetor components use the *socket* abstraction for network communication. This means that one component may act as a *client* (e.g. the broker) establishing a connection to a *server* (e.g. a database service) allowing the connection. Once established, the connection may be used for bi-directional data exchange.

Depending on the actual installation, different communication requirements may be present. For example when configuring the broker and a number of services on a local intranet, security concerns may be different than when requesting data from a remote Agetor service over the internet. In the latter case you may wish to establish a secure channel providing data encryption and authentication of client and server. On a more technical level, communication properties such as connection timeout, socket buffer sizes and Nagle'ing [1, 2] properties could be relevant to optimize communication.

In Agetor such connection properties are highly configurable.

3 Basic concepts

All Agetor component connections are created using the Agetor Socket Factory (ASF). Agetor components request a connection with a given connection identifier (CID). The ASF looks up the definition for the id in its configuration, creates an instance (client or server socket type) and return this to the component, which uses it.

This model allows changes in the connection types in an external manner with respect to the components.

3.1 Overview of default setup

The figure below illustrates how different AGETOR Components (AID) relates to connection id's (CID). A circle centre represents an AID, whereas (part of) the circumference represents a CID. Thus, the figure illustrates that the AID `bcmd` connects to the Broker using the CID's `bcmd` and `internalbroker`. If you wish SSL-encrypt this communication, you will need to make both `bcmd` and `internalbroker` SSL encrypted.

Similarly, if you want all services SSL-encrypted, you will need to assign `service` and `serviceclient` to an SSL CID. If you need to issue a ping-request using the AID `bcmd` you have to define `sport` as SSL-encrypted as well (due to the communication between broker and `servicerunner`).

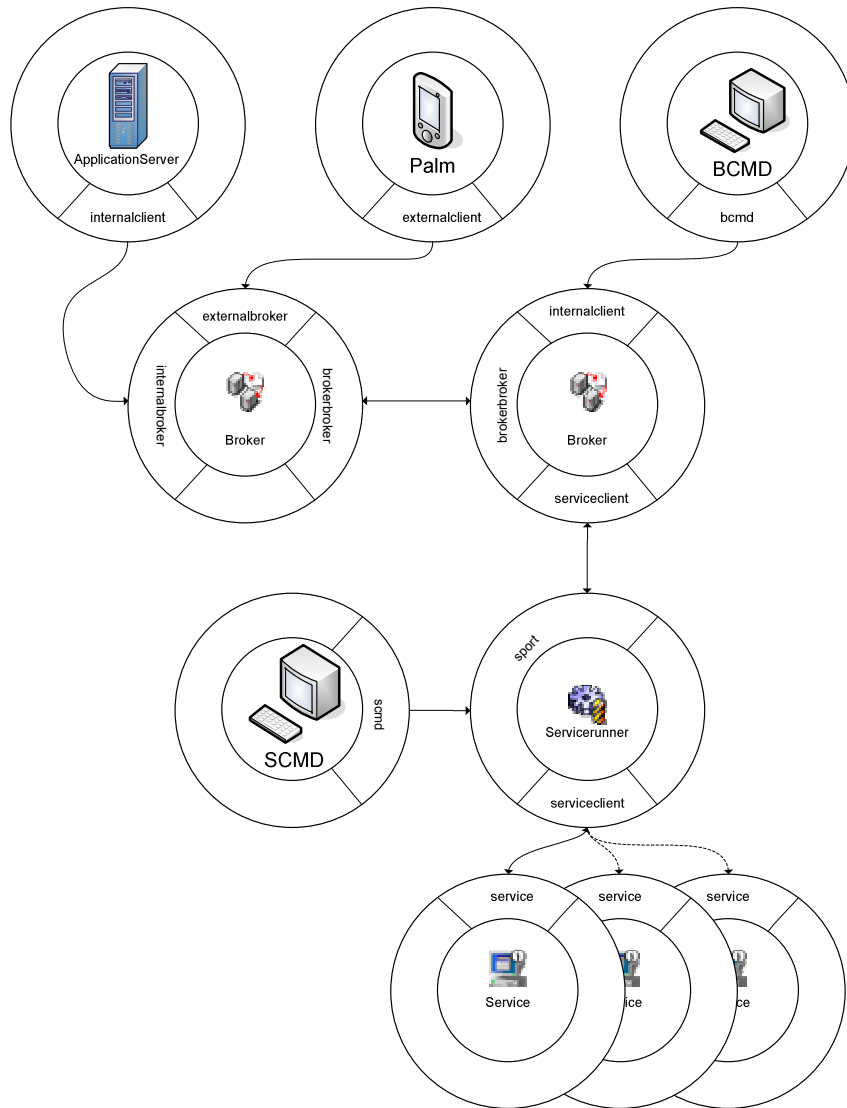


Figure 1: Default AID-CID relations

4 Configuration basics

The ASF configuration is controlled by a small number of properties and zero or more configuration files. Configuration files must be placed beneath the `$AGETOR_HOME/conf/sockets` directory.

4.1 Properties

Property	Description	Default
<code>Socketmanager.logfile</code>	File used for logging socket manager events	<code>sockets.log</code>
<code>Socketmanager.loglevel</code>	Log level used	Medium
<code>Sockets.usefactories</code>	Determines if the new configurable socket	True

	subsystem is in use or not. If false the installation will create connections as usual.	
<code>Socketmanager.default.factorybuilder</code>	Default factory builder used if no connection id is given for a client/server socket. I.e. the creator of default sockets.	<code>dk.bording.inside sockets.PlainSocket FactoryBuilder</code>

4.2 Socket configuration files

Socket configuration files define two things: (i) socket types - e.g. socket properties such as security, buffers and timeout and (ii) the default types used with Agetor components.

By default one socket type is defined with the CID *plain-socket*. This socket type amounts to the type of socket you would get from the `SocketFactory` and `ServerSocketFactory` classes in Java. The default types used by Agetor components like the broker, service-runner etc. are mapped by alias definitions. We return to this in section 6.1.

4.3 SSL Debug information

To obtain debug information on the actual SSL communication turn on Java's built-in SSL debug mechanism. To do this in an AGETOR® installation add the parameter `-Djavax.net.debug=ssl` in the file `$AGETOR_HOME/bin/agetor.bat`. Setting this property will output all SSL related debug information when it is accessed by any AGETOR Component in this components log-file allowing you to verify that your data exchange is SSL encrypted.

For a full description of available Java security debug options – of which SSL is only a part - visit this URL: <http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html#Debug>

5 Socket configuration file format

This section describes the XML structure of socket configuration files. Usage is exemplified in later sections.

5.1 sockets

This is the mandatory root tag in socket configuration files. It is common practice to use the `xmlns` attribute to indicate the namespace. The attributes on the root tag are only there as an indication of namespace and are not used in practice.

5.1.1 sockets/connection-factory

Defines a factory for logical connection type that later may be referred for connection instance creation.

Attribute	Description	Default
<code>cid</code>	The CID of the definition, e.g. “ <code>ssl-socket</code> ”. This name should be descriptive reflecting the type of socket.	
<code>Type</code>	Either the type is “ <code>client</code> ” or “ <code>server</code> ”. Client type definitions define the properties for the type when created as a client to a server. The server has slightly other	

	possible options.	
Factory-builder	Class name of factory creating <code>SocketFactory</code> instances	

5.1.1.1 sockets/connection-factory/configuration

This tag groups factory specific configuration data. Any XML is accepted and will be handed to the factory builder so that the factory instance may be configured in accordance.

5.1.1.2 sockets/connection-factory/client

`client` type definitions define the properties for the type when created as a client to a server.

5.1.1.3 sockets/connection-factory/server

`server` type definitions define the properties for the type when created as a client to a server.

5.1.1.4 sockets/connection-factory/(client|server)/socket-options

This tag groups socket options for connections of the given type. The group may appear both for `<client>` and `<server>` definitions. For *client* type sockets the following options may be used:

Name	Description	Default
Tcp_nodelay	A Boolean value indicating whether the Nagle algorithm should be used on the underlying TCP/IP connection. Some systems use an implementation with transmission delays that have a negative impact on response time. If you suspect this to be the case, try setting this value to true. To obtain max benefit this should be done both at the client and server side of the connection if the peer is remote.	False
So_timeout		
So_linger	-1 means no linger.	
So_sndbuf	Size of send buffer	
So_rcvbuf	Size of receive buffer	
So_keepalive	Boolean value.	

For *server* type sockets the following options may be used:

Name	Description	Default
So_reuseaddr	When a TCP connection is closed the connection may remain in a timeout state for a period of time after the connection is closed (typically known as the <code>TIME_WAIT</code> state or <code>2MSL</code> wait state). For applications using a well known socket address or port it may not be possible to bind a socket to the required address if there is a connection in the timeout state involving the socket address or port. Setting the <code>SO_REUSEADDR</code> flag to true allows connections to bind to the address even though a previous socket is in the wait state.	
So_timeout	Sets the time before the socket is released. Normally you will not need to specify a value for this option!	0 (infinite)
Use_client_auth	Only relevant to SSL-based server sockets. This option controls if client authentication is required or not.	

So_rcvbuf	Size of receive buffer	
-----------	------------------------	--

5.1.1.4.1 *sockets/connection-factory/(client/server)/socket-options/option*

Attribute	Description	Default
Name	The name of the option, e.g. “tcp_nodelay”	
Value	The value given for the option. E.g. “true”	

5.1.1.5 *sockets/connection-factory/server/server-socket-options*

This tag groups socket options for that are only relevant for server sockets.

5.1.1.5.1 *sockets/connection-factory/server/server-socket-options/option*

Attribute	Description	Default
Name	The name of the option, e.g. “so_rcvbuf”	
Value	The value given for the option. E.g. “10”	

6 Assigning socket types to Agetor components

This section describes how the communication properties of different Agetor components (broker, service-runner, services) may be specified.

6.1 Default set up in sockets.cfg

Defined socket types may be assigned to different Agetor components by the use of aliases. Such aliases define a mapping between an Agetor component id (AID) and a connection id (CID). The AID's that may be referred to are listed below:

AID	Description	Default
Service	The CID assigned to this component id will determine the <i>default</i> socket type used by all services if not overridden.	plain-socket
serviceclient	The CID assigned to this AID is default for service clients. E.g. the broker will use this CID for communication with services unless a CID is explicitly given in the broker configuration file.	plain-socket
Internalclient	This CID is used as default when an <code>InternalORB</code> class is used to create client connections (e.g. for the <code>bcmd</code> command or a data component).	plain-socket
internalbroker	The CID is also used by default for internal type server ports in the broker.	plain-socket
externalbroker	The CID is used by default for external type server ports in the broker.	plain-socket
brokerbroker	This CID is used when a broker establishes a fixed client connection to another broker. The CID must be compatible with the remote broker server socket type. E.g. if the remote broker uses SSL encryption, the <code>brokerbroker</code> CID must also refer to an SSL socket definition.	plain-socket
bcmd	CID used as default by <code>bcmd</code> command. The CID is handed to the <code>InternalORB</code> that is used for	plain-socket

	communication. The CID should be compatible with that of the broker port to which connections is being made.	
sport	Default CID used for the service runner	plain-socket
scmd	CID used as default by scmd command. The CID is handed to the <code>InternalORB</code> that is used for communication. The CID should be compatible with that of the service runner port to which connections is being made.	plain-socket

The above default definitions are used unless explicitly overridden. Overriding the defaults may be done in one of four ways:

- (i) Overriding in `socket.cfg`
- (ii) Overriding in component specific configuration file
- (iii) Overriding on command line for a component supporting this
- (iv) Overriding when instantiating an `InternalORB`

These methods are described in the following section.

6.2 Overriding CID's in `socket.cfg`

To change the CID of an Agetor component you should provide a connection definition with a unique CID. The CID is simply a string identifying the connection. The example below shows a socket definition that turns off the default use of the Nagle algorithm to reduce packet delay.

```
<connection-factory cid="nodelay-socket"
factory-builder="dk.bording.inside.sockets.PlainSocketFactoryBuilder" >
  <client>
    <socket-options>
      <option name="tcp_nodelay" value="true"/>
    </socket-options>
  </client>
  <server>
    <socket-options>
      <option name="tcp_nodelay" value="true"/>
    </socket-options>
  </server>
</connection-factory>
```

The `cid` attribute identifies the connection type. Note that a definition is made for both client and server. Using the definition as the default for one or more Agetor components is achieved by assigning the defined CID to the relevant AID. If for example we wish to use this no-delay socket with all services, we should add the following *aliases* to our socket configuration:

```
<aliases>
  <!-- Services -->
  <alias name="service" alias="nodelay-socket"/>
  <!-- Broker to services (should match service) -->
  <alias name="serviceclient" alias="nodelay-socket"/>
</aliases>
```

This definition states that services should use our no-delay socket by default and clients to services (broker) should do so as well.

All of the AID's listed in section 6.1 may be configured as above. The default for all AID's is the build-in CID *plain-socket*, which amounts to the type of socket you would get from the `SocketFactory` and `ServerSocketFactory` classes in Java.

6.3 Overriding CID's in component configurations

6.3.1 `Broker.cfg`

The broker uses connections for communication with services, other brokers and for listening on ports for incoming requests.

6.3.1.1 Listener ports

To override the CID for a listener port (internal or external protocol) the syntax below may be used. The fragment below defines an internal and an external port where the external port uses the default CID for external ports since no CID is given explicitly. The internal port (`Internal1`) on the other hand specifies a CID with the value `"ssl-socket"`. The same could have been achieved by changing the `internalbroker` mapping to `"ssl-socket"` in the socket configuration file if the property should hold for all internal ports of the broker.

```
<TIMEOUT CLIENTS="1800" RESPONSE="30" SESSIONS="1800"/>
<SECURITY CRYPTING="false" PERMISSIONCONTROL="false"/>
<PORTS>
<INTERNAL NAME="Internal1" PORT="{AGETOR_IPORT}" CID="ssl-socket" >
  <TIMEOUT CLIENTS="1800" SESSIONS="1800"/>
</INTERNAL>

<INTERNAL NAME="Internal2" PORT="20033" >
  <TIMEOUT CLIENTS="1800" SESSIONS="1800"/>
</INTERNAL>

<EXTERNAL NAME="External1" PORT="{AGETOR_EPORT}" >
  <TIMEOUT CLIENTS="1800" SESSIONS="1800"/>
</EXTERNAL>
</PORTS>
```

6.3.1.2 Services

Connection types for individual services must be explicitly defined if they are different from the default CID for services (`serviceclient`). This may be relevant when communicating with e.g. a remote service over the Internet. The excerpt from the broker configuration below illustrates how the different connection types of the services are specified to the broker. Below `"nodelay-socket"` is used for the `docportal` service, an `"ssl-socket"` for the `jdbcserviceMT` and the default connection type with the `jdbcservice2`. Obviously all types must be defined in a configuration file for the socket factory.

```
<SERVICE CID="nodelay-socket" NAME="docportal" PORT="7011"/>
<SERVICE NAME="jdbcservice2" HOST="localhost" PORT="33033"/>
<SERVICE CID="ssl-socket" NAME="jdbcserviceMT" RPORT="50" CONNECTIONREFRESH="180"/>
```

6.3.1.3 Fixed clients connections (firewall tunnelling)

The fixed client tag in the broker configuration also allows a connection id to be given. This must be done in the broker tag as stated below.

```
<FIXEDCLIENTS>
```

```

<BROKER NAME={string}
  HOST={string}
  PORT={int}
  CID={string}
  IDENT={string}
  RECONNECT={int:120}
  INACTIVITYDELAY={int:0}
  CONNECTIONREFRESH={int:0}
/>

```

6.3.2 Service runner configuration

In the service runner connection types may be specified for programs of type orb. This is accomplished with the CID attribute of the PROGRAM tag.

```

<PROGRAM CLASS="com.agetor.docportal.ap.gen.JDBCService"
  TYPE="orb"
  RPORT="50"
  CID="ssl-socket"
  DESCRIPTION="jdbcserver"
  LOGFILE="jdbcserver.log"
>

```

6.4 Overriding CID's on command line for a component

Some of the Agetor programs allow the specification of CID on start up. Usually you should not need to specify CID's different from the default, however when communicating with remote components they may be configured with a different security level (SSL/TLS) requiring you to conform.

Program	Description	Default
Bcmd	The CID decides the socket type used for communication with the broker. Should match the CID used on the broker for the given port. Use the <code>-i<cid></code> option for specifying the CID: <code>bcmd -p21002 -i"ssl-socket"</code>	
Scmd	The CID decides the socket type used for communication with the service runner. Should match the CID used on the service runner.	!!NOT THERE YET
<service>	Any service based on the Agetor framework accepts a CID determining the listener socket type. Use the <code>-i<cid></code> option for specifying the CID.	

6.5 Providing a specific CID to the InternalORB class

When using an instance of the `InternalORB` class for remote method invocation [3], you may specify the CID as well as the host and port addresses of the target service. When developing programs that use the `InternalORB` it is good practice to let such parameters be configurable – e.g. provide them as command line options.

An example instantiation is shown below. The variable `cid` holds the CID, which could be “`ssl-socket`” for instance.

```
orb = new InternalORB(host, port, cid);
```

7 Working with certificates

Agetor comes with a default trust store containing a single certificate. This trust store was created using the `keytool` command [5], which is the certificate manipulation tool provided by Sun Microsystems with their Java implementation. The trust store holds a 1024-bit RSA key that can be used for secure communication between Agetor components. You may want to generate your own certificates in real scenarios. To this end you may use the `keytool` [5] from sun, which also allows you to inspect, update and delete the content of trust stores.

```
C:\projects\agetor2\conf\sockets\certs>keytool -genkey -alias agetor -keyalg RSA -keypass agetor
-keystore trust.dat -storepass agetor -dname "CN=agetor@bording.dk, OU=Interactive, O=Bording
Data A/S, L=Copenhagen, ST=Denmark, C=dk"
```

Illustration 1 Generating the default Agetor certificate

Please refer to firewall tunneling example in section 9.2 to see how an SSL-socket is defined using this certificate.

8 Factory builder implementations

This section describes how new socket factory builders may be plugged into Agetor through simple configuration. Unless you are planning on providing new fundamental socket factory types you may skip this section.

The factories used for creating client and server socket instances are provided by factory builder classes. Such factory builders may be plugged into Agetor by extending the abstract class `ConfigurableSocketFactoryBuilder`.

The extending class must provide implementation of the following abstract methods:

```
public abstract SocketFactory getSocketFactory() throws Exception ;
public abstract ServerSocketFactory getServerSocketFactory() throws Exception ;
abstract public void configure(Element e, boolean validate) throws Throwable;
```

The `configure()` method receives a DOM element representing the parsed XML configuration input. The method is responsible for extracting information and validating the given information.

The `getSocketFacotory()` and `getServerSocketFactory()` methods must provide standard Java `javax.net.SocketFactory` instances and `javax.net.ServerSocketFactory` instances. To do so, extend these abstract base classes and implement abstract methods.

Two implementations are provided with Agetor. A plain socket factory builder and a factory builder for SSL/TLS socket types. The use and configuration of these are described below.

8.1 `dk.bording.inside.sockets.PlainSocketFactoryBuilder`

This factory builder provides ordinary Java sockets as would be returned from the `SocketFactory` and `ServerSocketFactory` classes. However, it applies whatever options configured to sockets before these are returned to the Agetor component requesting the socket.

8.2 `dk.bording.inside.sockets.SSLSocketFactoryBuilder`

This factory builder is able to create socket factory instances that provide secure SSL and TLS communication. The implementations are based on Sun's SSL socket factory implementations and

require a trusted keystore file to be specified. The filename may be absolute. If not, the prefixed path defaults to `AGETOR_HOME/conf/sockets/certs/`. The keystore is protected by a password, has a type and contains certificates of a certain format. At present time only “JKS” may be given for type and “SUNX509” for provider type. The attributes mentioned are defined using the `<keystore>` tag as in the example below. The protocol tag allows one of “SSL” and “TLS” to be specified.

```
<keystore file="trust.dat"
  password="secret"
  keypassword="mostsecret"
  type="JKS"
  provider="SUNX509" />
<protocol name="SSL"/>
```

9 Examples

This section provides a few configuration examples and is by no means exhaustive.

9.1 SSL based Broker to service communication

To have the broker communicate with a remote service using SSL, first you must ensure that an appropriate SSL socket is defined. Agetor comes with an SSL configuration in the `$AGETOR_HOME/conf/socket/def` directory named `socket_SSL.xml`. The content is listed below. The used CID is `ssl-socket`.

```
<sockets>
  <connection-factory cid="ssl-socket" factory-
builder="dk.bording.inside.sockets.SSLSocketBuilderFactory" >
    <configuration>
      <keystore file="trust.dat"
        password="agetor"
        keypassword="agetor"
        type="JKS"
        provider="SUNX509" />
      <protocol name="SSL"/>
    </configuration>
    <client>
      <socket-options>
        <option name="tcp_nodelay" value="true"/>
      </socket-options>
    </client>
    <server>
      <server-socket-options>
        <option name="use_client_auth" value="false"/>
      </server-socket-options>
      <socket-options>
        <option name="tcp_nodelay" value="true"/>
      </socket-options>
    </server>
  </connection-factory></sockets>
```

Having the broker and a specific service use these definitions require us to modify the broker configuration and start the service with the CID as a parameter. The broker configuration could look as below:

```
<SERVICE CID="ssl-socket" NAME="mysecureservice" HOST="www.remotemachine.com" PORT="20050"/>
```

On the machine where the service is executed (could be the same as the broker, but this would probably not justify the secure connection), the CID definition must be present and the service started with the CID:

```
<PROGRAM CLASS="com.agetor.docportal.ap.gen.MySecureService"
  TYPE="orb"
  PORT="20050"
  CID="ssl-socket"
  DESCRIPTION="mysecureserver"
  LOGFILE="mysecureserver.log"
>
```

9.2 SSL based Broker to broker firewall tunnelling

To set up firewall tunnelling [4] that is encrypted, you need a secure socket definition as described in section 9.1. Besides, the proxy-broker must expose a secure internal port for the broker on the secure network to connect to (*internal broker*). If clients of the proxy-broker don't need to communicate encrypted, a dedicated port could be defined as show below. In line 2 an ordinary internal port without encryption is defined. In line 6 an SSL-port is defined by the name `intssl`.

```
1.      <PORTS>
2.          <INTERNAL CID="plain-socket" NAME="int1" PORT="20002">
3.              <TIMEOUT CLIENTS="20" SESSIONS="1800"/>
4.          </INTERNAL>
5.          <INTERNAL CID="ssl-socket" NAME="intssl" PORT="21002">
6.              <TIMEOUT CLIENTS="20" SESSIONS="1800"/>
7.          </INTERNAL>
8.      </PORTS>
```

Illustration 2. proxy-broker's port definitions. An SSL port is defined for the internal broker (on secure network) to connect to.

The internal broker must connect to the encrypted port using the same socket type (i.e. the definition must be present on the internal broker. A fixed client section must refer to the proxy-broker's port (see [4] for details).

```
<FIXEDCLIENTS>
  <BROKER NAME="internalbroker"
    HOST="localhost"
    PORT="21002"
    IDENT="hi proxy"
    CID="ssl-socket"/>
</FIXEDCLIENTS>
```

Illustration 3. Fixed client definition ensuring SSL tunnel to proxy-broker.

Note that the broker may be controlled using `bcmd` on either of the internal ports configured. However the `bcmd` command should be instructed to use the correct socket type in the non-default case. In the listing below the broker is first contacted on its SSL-port (1-22). Then the broker is contacted on its non-encrypted port (24-). In line 31 the `iport` command displays the internal ports of the broker with their CID, port, client timeout, session timeout etc.

```
1. C:\projects\agetor2\conf\sockets\certs>bcmd -p21002 -i"ssl-socket"
2. |=====BROKER=====
3. |Version                = AGETOR Broker 2.0.1 (20030711134117)
4. |Startup time           = Tue Feb 10 17:31:09 CET 2004
5. |Total memory           = 2621440
6. |Free memory            = 1176768
7. |-----
```

```

8. [0]>show
9.      |=====Sessions=====
10.     |id   user      env      idle
11.     |-----
12.     |-----
13.     |0 sessions
14.     |=====Services=====
15.     |Name                               sent  rec  load idle      average
16.     |-----
17.     |ADMIN:internal                       3    4    1   0:0:0    561 ms
18.     |internalbroker:localhost( 0        0    0   0:0:20    -
19.     |servicerunner:127.0.0.1(2 0        0    0   0:0:20    -
20.     |-----
21.     |3 services
22. [1]>q
23.
24. C:\projects\agator2\conf\sockets\certs>bcmd -p20002
25.     |=====BROKER=====
26.     |Version                               = AGETOR Broker 2.0.1 (20030711134117)
27.     |Startup time                          = Tue Feb 10 17:31:09 CET 2004
28.     |Total memory                           = 2621440
29.     |Free memory                            = 1192528
30.     |-----
31. [0]>iports
32.     |Internal ports:
33.     |id   | cid   | port | cl.to | sess.to | perm.ctrl | resp.to
34.     |-----+-----+-----+-----+-----+-----+-----
35.     |int1@20002 |plain-socket|20.002|20    |1.800   |false     |30
36.     |intssl@21002|ssl-socket |21.002|20    |1.800   |false     |30
37.     |2 internal ports
38. [1]>

```

The listing below shows how the internal broker is contacted on its internal port and the status of fixed clients is inquired using the `fc` (fixed clients) command.

```

C:\projects\agator2\logs>bcmd -p30002
|=====BROKER=====
|Version                               = AGETOR Broker 2.0.1 (20030711134117)
|Startup time                          = Tue Feb 10 17:42:56 CET 2004
|Total memory                           = 2715648
|Free memory                            = 965048
|-----
[2]>fc
|-----
|Fixed clients:
|id   | host  | port | online | err | recon.del | inactiv.del | refresh.del
|-----+-----+-----+-----+-----+-----+-----+-----
internalbroker|localhost|21.002|true   |0   |120       |0           |30
1 fixed clients
[3]>

```

10 Socket factory log file

The log file used (determined by `socketmanager.logfile` property with `sockets.log` as default) will contain information about the load and initialization of the socket factory subsystem. Furthermore you may find error information related to the creation of socket factories used for client/server communication. However once a connection is established, any errors pertaining to that connection is no longer the responsibility of the socket factory and is logged in the log file of the relevant component using the socket. E.g. communication errors between the broker and its services is logged in the broker log.

11 Troubleshooting

11.1 Use compatible socket types

When working with socket connections of different types it may be crucial to use compatible socket types for clients and server. E.g. using a plain socket type against an SSL-server may simple cause a long connection period followed by a timeout when the socket implementation gives up understanding the SSL negotiation from the other end.

Using variations of the plain socket type with different buffer sizes, timeouts and Nagle properties does not cause incompatibility since the same basic protocol (TCP/IP) is still used.

11.2 Beware of invalid or expired certificates

Another cause of problems is often seen when using invalid or expired certificates for encrypted communication. The excerpt below shows the kind of exception this may cause. If you experience problems with the communication you should consult the log files of the involved components. In particular if communication has worked for a period but suddenly ceases, the cause may be an expired certificate. Checking the socket log may shed light on problems resulting from invalid or unknown certificates.

```
C:\projects\agator2>bcmd -p21002 -i"ssl-socket"
|Unable to get broker information: dk.bording.inside.orb.RemoteException:
  nested exception(dk.bording.inside.comm.CommException): Unknown error
  nested exception(javax.net.ssl.SSLHandshakeException):
java.security.cert.CertificateExpiredException: NotAfter: Mon Jan 12 14:15:17 CET 2004
```

11.3 Wrong certificate keys

Providing a wrong key for the trust store in use will result in an instantiation error when the SSL-socket is first used. Consult the socket log for details.

12 References

- [1] Nagle algorithm. <http://compnetworking.about.com/library/glossary/bldef-nagle.htm>
- [2] “Web Servers Should Turn Off Nagle to Avoid Unnecessary 200 ms Delays”. <http://citeseer.nj.nec.com/363758.html>
- [3] Agetor ORB Guide. http://www.agator.com/doc2/guides/ORB_guide.pdf
- [4] Agetor Runtime Environment Guide. http://www.agator.com/doc2/guides/ARE_guide.pdf
- [5] keytool from Sun. <http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html>