



AGETOR[®]

Runtime Environment

Table of contents

1	Environment.....	1
1.1.1	Tested platforms.....	1
1.2	Runtime Environment.....	1
1.3	Development Environment.....	1
1.4	Installation.....	2
1.4.1	Directory structure.....	2
1.4.2	Windows usage.....	3
1.4.3	Unix usage.....	3
1.4.4	Classpath.....	4
1.5	Configuration.....	4
1.5.1	Properties.....	5
2	The Agetor broker.....	9
2.1	Purpose.....	9
2.1.1	Scalability and load balancing.....	9
2.1.2	Security.....	9
2.1.3	Encryption.....	10
2.1.4	Identifying services.....	10
2.1.5	Permissions.....	10
2.2	Configuration.....	11
2.3	SETTINGS.....	12
2.3.1	Defining listener ports.....	13
2.3.2	Element: FIXEDCLIENTS.....	14
2.3.3	Element: SERVICE.....	17
2.3.4	Element: ENVIRONMENT.....	17
2.3.5	Element: LOGGING.....	18
2.3.6	Example configuration.....	18
2.4	Monitoring.....	19
2.4.1	Broker Command (bcmnd).....	19
2.4.2	Request.....	20
3	Service runner monitoring.....	21
3.1.1	Service Runner Command (scmd).....	21

1 Environment

The AGETOR Runtime Environment (ARE) is the smallest set of components needed to deploy and run AGETOR Internet solutions.

The AGETOR Development Kit (ADK) consists of various development tools, and is needed to develop and test Internet solutions. The ADK includes the AGETOR Runtime Environment (ARE).

1.1.1 Tested platforms

The AGETOR Runtime Environment as well as the AGETOR Development Kit will run on any platform supporting Java 1.3.

We have tested the AGETOR Runtime Environment and the AGETOR Development Kit on the following platforms (other OS or JVM can be used):

OS	JVM
IBM AIX 4.3	IBM JDK 1.3.1 for AIX 4.3
Windows NT 4.0	Sun JDK 1.3.1
Windows 2000	Sun JDK 1.3.1
Linux	Sun JDK 1.3.1
	IBM JDK 1.3.1

1.2 Runtime Environment

To deploy AGETOR you need the AGETOR Runtime Environment consisting of the following components:

- **Broker** The Broker is both a message router and a message queue. All communication passes through the Broker which relays messages.
- **ServiceRunner** The ServiceRunner is responsible for starting the necessary services based on configuration. This component may be omitted if services are started using other means.

1.3 Development Environment

The AGETOR Development Kit consists of the following development tools:

- `idl2java` Generates Java classes from an IDL specification supporting Java clients and services.
- `idl2vb` Generates Visual Basic code from an IDL specification supporting Visual Basic client and services as well as any other COM enabled system.
- `idl2ocx` Generates a Windows OCX from an IDL specification supporting any OCX enabled system
- `idl2abf` Generates ABF sources from an IDL specification supporting CA Ingres ABF services.
- `idl2c` Generates C sources from an IDL specification supporting clients and

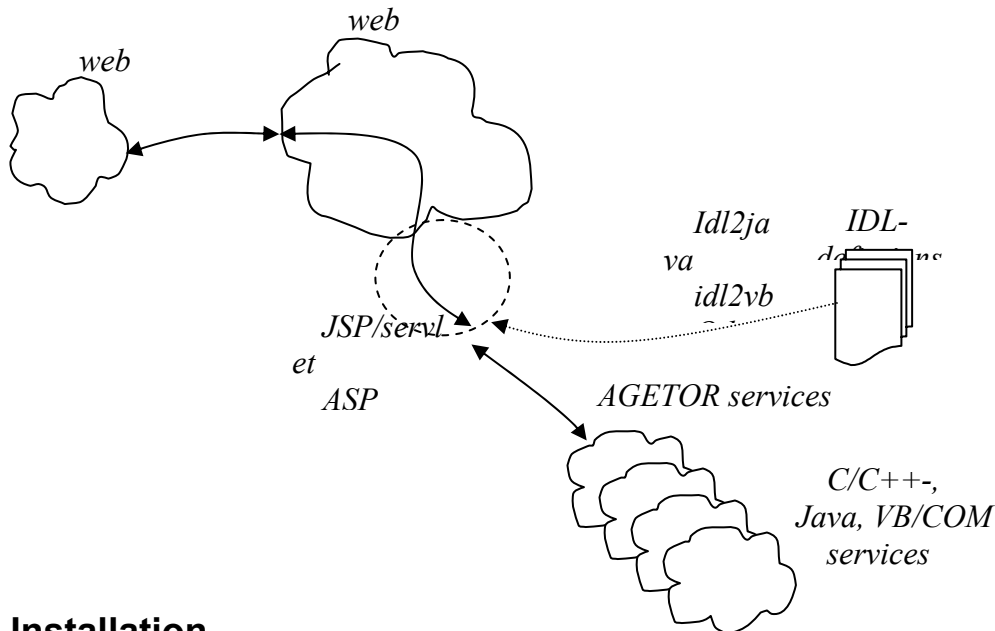
- `iml2java` services written in C. Generates Java Servlets from an IHTML document that any Web-server supporting servlets will execute.

The AGETOR Development Kit includes the AGETOR Runtime Environment and has the same software requirements.

Easy interaction with AGETOR services is supported through remote method invocation. The definitions of such methods and their parameter data types are specified using the standard Interface Definition Language (IDL). Using AGETOR tools the necessary Java communication code is produced from these definitions effectively making the communication transparent from the developer who only needs to know and use the methods and data types defined in IDL.

The input to servlets typically emerges from URL's or submission forms on web pages. The standard servlet interface provides some simple methods for accessing these parameters. The AGETOR servlet tools extend these capabilities providing Java methods to ease the mapping from simple name/value pairs to the structured data types used by the methods generated from IDL.

The figure below summarizes the described components in a visual way.



1.4 Installation

Use the AGETOR Install Tool to setup AGETOR.

1.4.1 Directory structure

The installation will have the following directory structure.

<pre> AGETOR_HOME - app - idl </pre>	<p>The root of the structure. Defined as an environment variable.</p> <p>IDL files defining remote server daemon methods and their parameters. These are converted to Java classes that may be imported and used in the Java sections of ihtml documents.</p>
--	---

- ihtml	Contains the IHTML files. These may contain embedded Java code and are converted into Java servlet source code using AGETOR tools.
- java	Old default location for classes developed for or generated by AGETOR®. Default in new installations is the value of the property <code>INSTALL_CLASSES_DIR</code>
servlet	- Old default for classes produced by <code>iml2java</code> . If used, the class files in this directory must be on the classpath. Default in new installations is the value of the property <code>INSTALL_CLASSES_DIR</code>
- lib	Old default for libraries. Default in new installations is the value of the property <code>INSTALL_LIB:DIR</code>
- resource	Contains the servlet templates generated from IDL files.
- vb	Contains Visual Basic code generated by AGETOR. OCX and COM are put into this directory when generated from IDL.
- xsl	XSL stylesheets declaring the IHTML control and layout markup. Customized presentation XSL are also put here.
- bin	General scripts to work with IDL and IHTML documents.
- def	Default scripts
- conf	Configuration of various runtime and development tools.
- def	Default configuration
- data	
- temp	Contains the temporary files created by the AGETOR Runtime. Use this directory for your temporary files.
- doc	Documentation.
- httpd	
- conf	Apache Webserver configuration files.
- logs	Output logs from services are put here.

1.4.2 Windows usage

The three most important files found in the `AGETOR_HOME\bin\` directory is `agetor.bat`, `j.bat`, and `prompt`.

<code>prompt.bat</code>	Sets the basic environment variables, <code>AGETOR_HOME</code> and <code>AGETOR_PROJECT_NAME</code> .
<code>agetor.bat</code>	Sets additional environment variables for the project.
<code>j.bat</code>	Runs the Java VM.

1.4.3 Unix usage

The three most important files found in the `AGETOR_HOME/bin` directory is `proj`, `agetor` and `j`.

<code>proj</code>	Sets the basic environment variables, <code>AGETOR_HOME</code> and <code>AGETOR_PROJECT_NAME</code> . This script has the same function on Unix as the <code>prompt</code> script on Windows
<code>agetor</code>	Sets additional environment variables for the project.
<code>J</code>	Runs the Java VM.

1.4.4 Classpath

Since ADK 2.0 the classpath is created dynamically based on three environment variables `INSTALL_LIB_DIR`, `INSTALL_CLASSES_DIR`, and `CUSTOM_CP`, which are defined in the `agetor` script. The two first variables both point to a single directory while the `CUSTOM_CP` can contain a several entries separated by a path separator (on Windows: semicolon, on Unix: colon). The classpath is constructed in this way:

1. The `CUSTOM_CP` is analyzed first. If it contains more than one entry, they are examined in the order they appear
2. If the custom entry is a directory the directory itself is added first. Then the directory is searched for jar files and then for zip files, which are added in the order they are found
3. If the custom entry is a specific jar or zip file, this file is simply added
4. If the custom entry is the pattern `*.jar` or `*.zip`, all files matching the pattern are added
5. `INSTALL_LIB_DIR` is searched for jar files and then for zip files, which are added in the order they are found
6. `INSTALL_CLASSES_DIR` is finally appended to the classpath

1.5 Configuration

Since ADK 2.0 there is no longer any need for a meta configuration file; configuration files are detected automatically using these rules:

- Files of configuration type, `<conftype>`, must be placed in the directory `AGETOR_HOME/conf/<conftype>` or a subdirectory hereof
- Configuration files other than property files must have the suffix `“.cfg”` or `“.xml”`
- Property files must have the suffix `“.properties”`
- Files in a directories called `“def”` or or directories containing the string `“backup”` – and any subdirectories hereof – are excluded
- Files are listed in alphabetical order using the full pathname (the exact algorithm is system-dependant)

Examples (all names are relative to `AGETOR_HOME`):

File	Included in
<code>conf/services/myservice.cfg</code>	services
<code>conf/services/myservice.cfg~1</code>	-
<code>conf/services/def/myservice.cfg</code>	-
<code>conf/services/backups/myservice.cfg</code>	-
<code>conf/services/myproduct/myservice.cfg</code>	services

The meta configuration file may still be used and the files specified in the meta configuration file are listed before the automatically included files.

The most common configuration types are:

CACHES	Defines the cache settings
COMPONENTS	Mapping logical component names to actual classes
DATAComponents	Declaration of datacomponents and their logical name
ENTITYCONTAINERS	Defines the entity container settings
IDL	Defining classes corresponding to IDL structures
PROPERTIES	Properties for different modules
SERVICES	The services that the ServiceRunner controls
SERVLETS	Mapping of logical names for servlets to classes
STORAGES	Declaration of storages

For each of the configuration types, a semicolon separated list configuration files can be supplied. The meta configuration file could look something like this:

```

components=components.cfg
datacomponents=datacomponents.cfg
idl=idl.cfg
properties=agotor.properties;project.properties
services=services.cfg
servlets=servlets.cfg
storages=storages.cfg
entitycontainers=entitycontainers.cfg
caches=caches.cfg

```

The sequence may be significant. Configuration files are listed from left to right and a latter configuration files may overwrite identical configuration in a previous files. This is up to the class, that has requested the list of configuration files of a given type. Most of the configuration classes in ADK uses overriding in this way, specifically, the property class, InsideProperties, will allow properties to be overridden.

1.5.1 Properties

The properties for all AGETOR tools and applications are described below.

AGETOR_I ^{PORT}	The AGETOR_I ^{PORT} is the absolute internal port on which the Broker is listening.
AGETOR_E ^{PORT}	The AGETOR_E ^{PORT} is the absolute external port on which the Broker is listening. Requests from outside the webserver are routed through this port.
AGETOR_S ^{PORT}	The AGETOR_S ^{PORT} is the absolute port for monitoring the ServiceRunner. The ServiceRunner Command Client is described elsewhere
agotor.port.range	The port base address by this project. All the relative port numbers will be added to this base in order to obtain the absolute port number.
AGETOR_B ^{ROKER}	The AGETOR_B ^{ROKER} is the IP address or

	machine name of the host running the Broker. Default value is localhost which is 127.0.0.1.
AGETOR_ENV	The AGETOR_ENV is the default environment for requests. Clients may access other environments explicitly but will implicitly default to this environment. Please refer to section 2 for a description of how the Broker handles environments.
AGETOR_LOGIN_OK	The URL to redirect the browser to on successful login.
AGETOR_LOGIN_ERR	The URL to redirect the browser to on failed login.
AGETOR_JAVACOMPILER	The java compiler to use for compiling java sources to class files.
agetor.servicerunner.excludedpackages	Packages excluded from reloading process in the servicerunner. Classes located in jar or zip archives must be excluded from reloading. The value consists of a semicolon (;) separated list of package (prefixes) of classes in jar or zip files. Example is com.sun;sun;org;COM
agetor.servicerunner.timeout	How long time the ServiceRunner will wait for a ServiceRunner client service callback.
agetor.datacomponent.timeout	Timeout value in seconds for datacomponents. When a datacomponent has not been accessed in this interval it is removed. Default value is 1800 seconds = 30 minutes.
agetor.tools.idl2java.typedsequences	Indicates whether the untyped java.util.Vector is used for sequences in IDL or a specific typed sequence class is generated for each IDL sequence. Default is false corresponding to untyped Vector.
agetor.tools.idl2java.structfactory	Controls whether the idl2java tool generates code for IDL that uses the idl.cfg configuration file for resolving classes. Default value is true.
agetor.orb.timeout	Timeout value in seconds for internal ORBs. When a internal ORB has not received an answer in

<code>agetor.socket.tentatives</code>	this interval a RemoteException is thrown. Default value is 10 seconds. Stettings value are in seconds eg. 10 is 10 seconds.
<code>agetor.socket.delay</code>	How many times a client will try to establish a socket connection to a server or to the broker: default = 3 The delay between each socket connection tentative: default 1 sec
<code>agetor.orb.request.oldformat</code>	Use the old protocol when a client issues a new ORB request. This is used for backward compatibility purpose. NB: the new ORB feature will be disabled if the old protocol is used. Default: true. NB. the value can be overridden from the InternalORB class.
<code>agetor.orb.protocol.encoding</code>	The string encoding used in the ORB request. NB this is used only in the new protocol. The default is the file.encoding property used by java. NB. the value can be overridden from the InternalORB class.
<code>agetor.orb.protocol.compression</code>	Enable the message compression. The only value possible is <i>gzip</i> . If omitted the no compression is used. NB. the value can be overridden from the InternalORB class.
<code>agetor.orb.server.loggroup</code>	Default log group used in the ServerORB. The loggroup can be overridden in the class instantiation.
<code>agetor.orb.server.requests.max</code>	How many concurrent requests a ServerORB based service allows. The value can be overridden by the command line: <i>--maxReq</i> Default: 1
<code>agetor.orb.server.requests.startpool</code>	Minimum threads count in the request-thread-pool. The value can be overridden by the command line: <i>--minReq</i> Default: 1
<code>agetor.orb.server.connections.max</code>	How many connections a ServerORB based service allows. The value can be overridden by the command line: <i>--maxConn</i> Default: 50
<code>agetor.orb.server.connections.startpool</code>	Minimum threads count in the connection-thread-pool. The value can be overridden by the command line: <i>--minConn</i> Default: 2
<code>agetor.orb.event.timeout</code>	How long time a ServerORB will wait the

termination of the events:
ServerORBEvents.beforeShutdown,
ServerORBEvents.afterShutdown,
ServerORBEvents.aboutToKill and
ServerORBEvents.onKill
Default: 30 sec

2 The Agetor broker

This section describes configuring, starting and monitoring the Broker.

2.1 Purpose

Basically the Broker is a message router and message queue. All remote procedure calls from clients to services are sent as messages through the Broker making it the central point routing all communication. This requires a high performing and scalable architecture ensuring short response times and reliable operation in high load situations.

2.1.1 Scalability and load balancing

The Broker itself is a multithreaded application forwarding requests to services and responses to clients with minimum overhead. However a service will have a minimum processing time for each method invocation, giving a maximum number of method invocation per time unit. Many concurrent users may create a contention situation when the number of method invocations for any one service exceeds this threshold.

Therefore the Broker allows you to replicate services, that is, start multiple services of the same type servicing the same method, potentially on different machines. The Broker will balance the request load between these equivalent services increasing the contention level by the number of replicas. This results in a highly scalable solution, which is limited primarily by the number of total backend system processing power available. Different load balancing methods are available. See the `LOADBALANCEMETHOD` property description in the configuration section.

2.1.2 Security

Besides the primary functions the Broker enforces security restrictions based on access rights on individual method invocations. A separate service for handling user permissions is responsible for telling the Broker the permission of individual users. This allows you to differentiate users accessing the same service in for example reading and writing permissions. The actual service will never receive a method invocation from a client with insufficient permissions, thus ensuring secure operation in an Internet based system.

2.1.2.1 Architecture


The Brokers primary function is to enable communication between clients and services. Services are programs permanently active (resident), accessing backend resources such as databases. They will respond to parameterized requests, as for instance products in a specific product category, price at ordering time etc.

Clients have permissions and services will require specific permissions for access. The Broker enforces these permissions and will encrypt communication.

Clients identify themselves by issuing a session-request with a user name, and the Broker registers a logical connection to the client if the user name is accepted. Acceptance requires the client to encrypt with its secret key. The client is then given a unique session number with the permission of the user and will send requests using this session number. The Broker will then route requests from this session number to services that will accept the session permissions.

2.1.3 Encryption

The Broker and client communicate using encrypted messages to prevent information from being accessible to other than the client and Broker. By using sequential message numbering message replication is eliminated. When a client issues a session request it sends the user name twice in plain text and encrypted form. If the client encrypted the user name with the correct key, the Broker will be able to match the plain user name with the decrypted user name. This authenticates the client and only then will the client gain access to services behind the Broker.

 With ADK 2.0.3 state of the art encryption protocols are supported by Agetor components. You may now configure broker, clients and services to use certificates and strong encryption as SSL and TLS are now supported.

2.1.4 Identifying services

A service typically handles several kinds of requests, each identified by a question number. Clients issue requests identifying these by the corresponding question number and sending the parameters for the request. The Broker finds the services handling this question number and forwards the requests to the service with the least load. When the service responds the Broker forwards this response to the client.

2.1.5 Permissions

Not all authorized clients should be able to access the same services. Customers, sales assistants, anonymous clients and system administrators all have different access rights. These access rights are enforced by permissions. Each client has a number of permissions and every service request requires one of several permissions. If the client does not have one of the required permissions this service will be inaccessible.

2.1.5.1 Client permissions

Client permissions are assigned to a specific user name with the key (password) and the environment.

Env	User	Key	permissions
myenv	anonymous	anonymous	1
myenv	cust1	cust1	1:2
myenv	sale	Sale	1:2:3
myenv	adm	Adm	1:2:100

2.1.5.2 Service permissions

Service requests identified by question number and environment are associated with one or several permissions.

Env	question	description	permissions
myenv	1	request product info	1
myenv	2	submit order	2

myenv	3	statistics	3
Myenv	9000	user info	100
Myenv	9001	service permissions	100

2.2 Configuration

The command `broker-start` starts the Broker with the configuration file `${AGETOR_HOME}/conf/broker.cfg`. This configuration file specifies the services available to clients, and is simply an XML file. The XML configuration file may reference properties by using `${property-name}`, which will be replaced by the value of the property (environment variable defined during setup). This configuration file consists of the following elements: a `settings` element, a logging element, a `fixedclients` element, a `services` element and one or more environment elements. The file contains a number of sections each defining different settings:

```
<BROKER>
  <SETTINGS>
    <RESOURCES INITIALTHREADS="..." />
    <TIMEOUT CLIENTS="..." SESSIONS="..." RESPONSE="..." />
    <SECURITY CRYPTING="true/false" PERMISSIONCONTROL="true/false" />
    <PORTS>
      <INTERNAL CID="..." NAME="..." PORT="..." >
        <TIMEOUT CLIENTS=="..." PERMISSIONCONTROL=="..." RESPONSE=="..." SESSIONS=="..." />
      </INTERNAL>
      <EXTERNAL CID="..." NAME="..." PORT="..." ">
        <TIMEOUT CLIENTS=="..." PERMISSIONCONTROL=="..." RESPONSE=="..." SESSIONS=="..." />
      </EXTERNAL>
    </PORTS>
  </SETTINGS>

  <FIXEDCLIENTS>
    <BROKER NAME="..." HOST="..." PORT="..." IDENT="..." CID=="..." CONNECTIONREFRESH=="..."
    INACTIVITYDELAY=="..." />
  </FIXEDCLIENTS>

  <SERVICES>
    <SERVICE NAME="..." RPORT="..." CID=="..." CONNECTIONREFRESH=="..."
    INACTIVITYDELAY=="..." />
  </SERVICES>

  <ENVIRONMENT NAME="...">
    <QUESTION NAME="..." QNO="..." />
  </ENVIRONMENT>

  <ENVIRONMENT NAME="...">
    ...
  </ENVIRONMENT>

  <LOGGING>
    <OVERALL FORMAT="[%TIME]%NL%TEXT" />
    <ERROR LEVEL="MAXIMUM" FORMAT="*** %CLASS.%METHOD:%TEXT" />
    <MESSAGE LEVEL="MINIMUM" />
    <BROKER LEVEL="MINIMUM" />
    <SERVICE LEVEL="MINIMUM" />
    <SESSION LEVEL="MINIMUM" />
    <CONFIG LEVEL="MINIMUM" />
    <EVENT LEVEL="MINIMUM" />
    <ENVIRONMENT LEVEL="MINIMUM" />
    <DEBUG LEVEL="MINIMUM" FORMAT="*** %CLASS.%METHOD:%TEXT" />
    <CONTROL LEVEL="MINIMUM" />
    <CLIENT LEVEL="MINIMUM" />

```

2.3 SETTINGS

This element contains general broker configuration properties.

2.3.1 Resources

This section defines general resource settings of the broker.

`RESOURCES.INITIALTHREADS` Attribute specifies the initial number of client threads allocated. The broker pools the client threads and allocates more if required.

`RESOURCES.LOADBALANCEMETHOD` Attribute specifies the algorithm used for load balancing when more services are available for a request. At present you may choose between `LOADRESPONSESIMPLE` and `LOADNAIVE`.

`LOADRESPONSESIMPLE` will make a qualified guess as to what service instance that is able to reply faster. The algorithm computes the response time as

$$(\text{current_load} + 1) * \text{avrg_response_time}$$

which is suitable in most cases and thus the default.

`LOADNAIVE` does not consider average response time of the services but solely the current load (number of requests being processes by the service at the moment). The service with least load is chosen. If requests for a service may vary significantly in processing time (e.g. different methods are called in the service) this algorithm may provide a better utilization of the services.

`SECURITY.CRYPTING` Attribute switch is either on or off and decides whether or not messages transferred by the broker are encrypted. (**Deprecated! You should use secure connection types on internal ports instead.**)

This is the global default. It may be overridden on port level from ADK 2.0.3.

`SECURITY.PERMISSIONCONTROL` Attribute switch specifies whether permission control on method invocations is activated. Activating this requires a service implementing the security interface.

`TIMEOUT.CLIENTS` The number of seconds a client may hold an open connection to the broker with no activity (i.e. requests sent) before the connection is closed by the broker.

This is the global default. It may be overridden on port level from ADK 2.0.3.

`TIMEOUT.SESIONS` The number of seconds a session may exists in the broker

with no activity (i.e. requests sent) before the session is removed by the broker.

This is the global default. It may be overridden on port level from ADK 2.0.3.

TIMEOUT.RESPONSE

Number of seconds the broker will wait for a service to respond to a client before timing out and sending a NO_SERVICE_CONTACT error to the client.

2.3.2 Defining listener ports

From ADK 2.0.3 the broker supports definition of multiple listener ports with different communication properties. Ports may be of type either internal or external (deprecated). External ports are a historic relic and no longer recommended since the encryption properties and protocol of external ports are now handled in superior ways using internal ports. The broker will continue to support old configuration file types though.


There may be multiple internal ports on which different types of clients connect to the broker. A web server may relay request on one port and another broker may initialize connection on another (e.g. SSL-based port).

The template below illustrates the definition of two internal and an external port. The two port types accept the same attributes and only differ in tag (INTERNAL vs. EXTERNAL). Note that global definitions for timeout and security are defined. However the timeout properties for individual ports may override these.

```

<TIMEOUT CLIENTS="..." SESSIONS="..." RESPONSE="..." />
<SECURITY CRYPTING="..." PERMISSIONCONTROL="..." />
<PORTS>
  <INTERNAL CID="..." NAME="..." PORT="..." >
    <TIMEOUT CLIENTS=="..." SESSIONS=="..." />
  </INTERNAL>
  <INTERNAL CID="..." NAME="..." PORT="..." >
    <TIMEOUT CLIENTS=="..." SESSIONS=="..." />
  </INTERNAL>
  <EXTERNAL CID="..." NAME="..." PORT="..." ">
    <TIMEOUT CLIENTS=="..." SESSIONS=="..." />
  </EXTERNAL>
</PORTS>

```

 Only the ports explicitly defined exist. I.e., if no ports are defined in the <PORT> section, the broker cannot be contacted since no defaults are assumed! However, if no port section is defined, the broker defaults to old semantics and allocates an internal and external port.

The tags and attributes related to port definitions are described in the table below.

PORTS	This tag holds all listener port definitions. Two basic types exist: internal and external. This way of defining port was introduced in ADK 2.0.3 and allows multiple internal and external ports that may use different socket types.
PORTS.INTERNAL	Attribute specifies the port on which the Broker allows external clients to connect. If omitted then AGETOR_IPOINT is used.
PORTS.EXTERNAL	Attribute specifies the port on which the Broker allows external

Name	The name of the service that this inner broker is know as at the outer broker.
Host	The name or IP address of the machine where the outer broker is running.
Port	The internal port on the host where the outer broker may be reached.
Reconnect	Seconds to wait before establishing new connection to the outer broker in case of lost connection.
Inactivitydelay	Maximum seconds without line activity before the connection is reestablished.
Connectionrefresh	Maximum period without reestablished connection. If period is exceeded, the connection is reestablished.

A matching service entry defining the presence of the inner connecting broker must be defined on the insecure side (X):

```
<SERVICE HOST="string"
NAME="string"
TYPE="broker"
IDENT="string"
REACHABLE="false"
CONNECTING="true"
/>
```

It is imperative to specify matching IDENT and NAME attributes on the two brokers.

Example inner (secure side of firewall) brokers configuration:

```
<FIXEDCLIENTS>
  <BROKER NAME="inner_service"
    HOST="outer_host"
    PORT="20002"
    IDENT="unique_id"
    RECONNECT="15"
    INACTIVITYDELAY="33"
    CONNECTIONREFRESH="60"/>
</FIXEDCLIENTS>
```

Example corresponding outer (insecure) brokers configuration:

```
<SERVICES>
  <SERVICE NAME="inner_service"
    TYPE="broker"
    IDENT="unique_id"
    REACHABLE="false"
    CONNECTING="true"
    PORT="23002" />
</SERVICES>
```

2.3.3.1 Reconnection options

The java broker support fixed clients for inside out connections on systems with firewall disallowing ingoing connections. In some situations the firewall may lay down connections without

the involved brokers get notified. To circumvent this problem the inner broker can reestablish fixed connections whenever no traffic occurred for a time period as well as at a user defined refresh interval.

Three options on the fixed client connection control the reconnect behavior: RECONNECT, INACTIVITYDELAY and CONNECTIONREFRESH.

2.3.3.1.1 RECONNECT

This option defines how many seconds the connecting broker should wait after the connection has been lost before it attempts a reconnect. This ensures that the connecting broker doesn't try to reconnect continuously if the peer broker is down for a longer period. The option does not apply when reconnect is performed due to inactivity or as periodic refresh (see below).

2.3.3.1.2 INACTIVITYDELAY

This option is in effect if the specified value is greater than zero. Otherwise it is ignored. The option specifies for how long the connecting broker will keep a connection that doesn't have any traffic. When the threshold is exceeded the connection is broken and a new established. This will ensure that "dead" connections (to firewall only) are cleaned up and new made. If it is known that the firewall may drop the connection after N seconds of inactivity this option could be set to N/2 seconds ensuring that a dead line never occurs.

If the line may be lost regardless of traffic, a method to detect and reestablish the lost line is to have a continuous job running at the peer broker machine that executes a *ping* command periodically. Combine this with a short INACTIVITYDELAY (2-3 times the ping period) and the line will be reestablished if no data (pings) flow.

2.3.3.1.3 CONNECTIONREFRESH

With this option you may specify longest allowed connection time between brokers without reestablishment of the channel. If for example your system for some reason after N seconds connection always drop the line thru the firewall regardless of traffic, this option will ensure that the connection is reestablished when its age exceed CONNECTIONREFRESH seconds (where CONNECTIONREFRESH < N).

Attribute	Type	unit	Legal values
RECONNECT	Int	Seconds	>0
INACTIVITYDELAY	Int	Seconds	0, >0
CONNECTIONREFRESH	Int	Seconds	0, >0

If INACTIVITYDELAY or CONNECTIONREFRESH are 0, their functions do not apply. However they may both be specified together. E.g.

```
INACTIVITYDELAY="1200" CONNECTIONREFRESH="36000" RECONNECT="30"
```

Will ensure that broken lines are reestablished after 30 seconds, inactivity for more than 20 minutes ensures reconnect and that a reconnect is made at least every 10 hours regardless of traffic.

2.3.4 Element: SERVICE

The services element specifies the list of services this Broker will route requests to. These services may be `socket_servers` and/or `connecting_brokers`.

A `socket_server` defines a service that is connected through TCP. The required attributes are:

Attribute	Description	Default
Name	The name identifying this service	
Host	The name or IP address of the machine where the service is running	
Rport	The relative port on the host where the service is listening for connections. The absolute port is obtained by adding the property <code>agetor.port.range</code> . The value can be true or false	
Port	The absolute port on the host where the service is listening for connections.	
CONNECTIONREFRESH	Number of seconds between connection reestablishment. A value of 0 means never reconnect.	0
INACTIVITYDELAY	Number of seconds without traffic from service before connection reestablishment. A value of 0 disables this property.	0

A `connecting_broker` defines a broker that operates a fixed client to this broker. This broker connects to the `connecting_broker` allowing it to use services on this broker. The required attributes are:

- `name` The name identifying the remote Broker.
- `host` The name or IP address of the machine where the Broker is running.
- `ident` The identifier used for accepting the fixed client connection.

2.3.5 Element: ENVIRONMENT

Each Broker has several environments grouping related services. Clients have permissions to a specified environment and specified question numbers in this environment.

The required attribute for the environment is:

- `name` The name identifying the environment.

Questions correspond to methods. However several methods may be associated to the same question number and several services may handle the same question numbers. The Broker routes messages based on question numbers and user permission.

The required attributes for a question are:

name	The name identifying the service handling the question.
qno	The number identifying this question.

2.3.6 Element: LOGGING

This element controls logging information output from the Broker during normal operation. The sub elements controls logging for different groups of Broker functionality and each has a level attribute ranging from 0 to 10 where 0 disables logging and 10 enables detailed logging.

overall	This element limits all other sub elements so that logging may be completed disabled by setting this level attribute to 0. Setting this level attribute to 10 will enable logging in each group specified by the groups level attribute.
client	Specifies log output from client administration.
service	Specifies log output from service administration.
session	Specifies log output from session administration.
env	Specifies log output from environment administration.
config	Specifies log output during startup configuration. This will help you find configuration errors preventing startup.
control	Specifies log output from Broker Control Client access.
broker	Specifies general Broker event logging.
message	Specifies message level logging which is only useful for debug purpose.
error	Specifies error event logging level.

2.3.7 Example configuration

The following is an example configuration of the Broker, illustrating different features:

```
<BROKER NAME="My Broker">
  <SETTINGS>
    <RESOURCES INITIALTHREADS="20" LOADBALANCEMETHOD="LOADNAIVE"/>
    <TIMEOUT CLIENTS="1800" SESSIONS="1800" RESPONSE="30" />
    <SECURITY CRYPTING="false" PERMISSIONCONTROL="false" />
  </SETTINGS>
  <FIXEDCLIENTS>
    <BROKER NAME="connection" HOST="somehost" PORT="3010" IDENT="id2"/>
  </FIXEDCLIENTS>
  <SERVICES>
    <SERVICE NAME="myservice" HOST="mymachine" PORT="7020" CONNECTIONREFRESH="3600"/>
    <SERVICE NAME="myservice2" HOST="mymachine2" PORT="4120"/>
    <SERVICE NAME="myservice3" HOST="mymachine2" PORT="3020" INACTIVITYDELAY="600"/>
  </SERVICES>
</BROKER NAME="My Broker">
```

```

<SERVICE NAME="mylocalservice1" PORT="10"/>
<SERVICE NAME="mylocalservice2" PORT="11"/>
<SERVICE NAME="mylocalservice3" PORT="12"/>
</SERVICES>

<ENVIRONMENT NAME="myenv">
  <QUESTION NAME="permdm" QNO="9005 9006"/>
  <QUESTION NAME="myservice" QNO="200"/>
  <QUESTION NAME="myservice" QNO="210"/>
  <QUESTION NAME="myservice2" QNO="210"/>

  <QUESTION NAME="mylocalservice1" QNO="310"/>
  <QUESTION NAME="mylocalservice2" QNO="311"/>
  <QUESTION NAME="mylocalservice3" QNO="312"/>
</ENVIRONMENT>

<LOGGING>
  <OVERALL FORMAT="[%TIME]%NL%TEXT" />
  <ERROR LEVEL="MAXIMUM" FORMAT="** %CLASS.%METHOD:%TEXT" />
  <MESSAGE LEVEL="MINIMUM" />
  <BROKER LEVEL="MINIMUM" />
  <SERVICE LEVEL="MINIMUM" />
  <SESSION LEVEL="MINIMUM" />
  <CONFIG LEVEL="MINIMUM" />
  <EVENT LEVEL="MINIMUM" />
  <ENVIRONMENT LEVEL="MINIMUM" />
  <DEBUG LEVEL="MINIMUM" FORMAT="** %CLASS.%METHOD:%TEXT" />
  <CONTROL LEVEL="MINIMUM" />
  <CLIENT LEVEL="MINIMUM" />
</LOGGING>
</BROKER>

```

2.4 Monitoring

Monitoring the Broker can be done by the command program `bcmd` or by using the AGETOR Control Center web page at <http://<host>/agetor/admin/> web page.

2.4.1 Broker Command (bcmd)

The following commands are available:

Information	Shows general information about the Broker status.
services/show	Shows information about the configured services, their outstanding requests, average response time and idle time.
Clients	Shows information about the currently connected clients.
sessions	Shows the current sessions and their status.
Envs	Shows the configured environments and their services.
request env qno method params	Sends a request to the service identified by the given environment (<i>env</i>) and question number (<i>qno</i>). The parameters for the request are given by <i>params</i> .
ping service	Tests whether the service is running and connected to the Broker.

ping	Successively pings all services.
kill service	Shuts down this specific service program. This will only work if the service was running and connected to the Broker.
shutdown	Terminates the broker cutting off all communication.
Q	Terminates the client program but <i>not</i> the broker.
Help	Shows a list of commands.

The bcmd program connects to the Broker using the AGETOR_IPORT. Therefore it will be able to maintain the broker from any machine being able to access the broker on this port.

2.4.2 Request

The format of the request command is:

```
request(env, qno, method, ...)
```

env - The environment in which the question number is defined.

qno - The question number to invoke.

method - The name of the method to invoke.

... - Simple parameters required by the method.

All parameters are simple typed and defaults to string if it cannot be converted to an integer.

You may force the type of the parameters by prefixing the following:

%s	string
%i	short
%i	long
%i	char
%i	boolean
%f	float
%e	double
%d	Date

3 Service runner monitoring

Monitoring the ServiceRunner can be done both by the command program `scmd` and by the browser interface, which accesses the ServiceRunner as any other AGETOR service. The browser interface description is not in the scope of this guide but can be found at <http://<host>/agetor/admin/>.

3.1.1 Service Runner Command (`scmd`)

After the service runner is started it may be contacted through the client command program `scmd`. The communication is based on standard AGETOR IDL and a small number of commands may be issued from the interactive interface. These are:

<code>show</code>	Shows all service and their status which may be one of <i>running</i> , <i>stopping</i> or <i>stopped</i> .
<code>start <service></code>	Starts the named service. If the program type is Java-method, orb or client the classes are reloaded.
<code>stop <service></code>	Stops a service.
<code>kill <service> [<time>]</code>	Wait the service terminate for the indicated time, thereafter the service is killed. If time is not defined then the service is killed immediately.
<code>status <service></code>	Return the service status if the service implement the callback. It work only for orb and client type services.
<code>shutdown</code> <code>kill:<time></code>	<code>--</code> Signals all the service to stop and await the services termination unless a kill time is specified. In this case the ServiceRunner will wait only for the specified time. Thereafter the service runner is terminated.
<code>quit</code>	Terminates the client program but <i>not</i> the service runner or any programs within it.

By default, the `scmd` program connects to the ServiceRunner by contacting it directly.

The command program can be called from the command line and from other machines with one or more of the following arguments:

```
scmd [-h<host>] [-p<port>] [--<command> [--<command>]...]
```

Examples:

Contact the ServiceRunner on a remote server

```
scmd -hsomeserver -p20005
```

Shutdown the local ServiceRunner

```
scmd --shutdown
```

Shutdown the ServiceRunner on a remote server

```
scmd -hsomeserver -p20005 --shutdown
```

By giving `scmd` one or more commands on the command line the command(s) will be executed and afterwards `scmd` will exit. The interactive interface will not be started.

References

- [1] Agetor Connection Configuration Guide.