Borland VisiBroker™ 8.0

GateKeeper Guide

Borland Software Corporation
20450 Stevens Creek Blvd., Suite 800
Cupertino, CA 95014 USA
www.borland.com

VB 80 GateKeeper Guide
April 2007

Borland®

Contents

1

Introduction to GateKeeper

This section provides an overview of GateKeeper and describes different ways to start GateKeeper.

What is GateKeeper?

GateKeeper is an OMG–CORBA compliant General Inter-ORB Protocol (GIOP) Proxy Server developed by Borland Software Corporation which enables CORBA clients and servers to communicate across networks while conforming to security restrictions imposed by Internet browsers, firewalls, and Java sandbox security. In effect, GateKeeper serves as a gateway or proxy for clients and servers when security restrictions prevent clients from communicating with the servers directly.

GateKeeper is often used when you do not want to expose the server directly to clients or when a client's access to the server is restricted. In the latter case, either the client is an unsigned applet or there is an intervening firewall.

GateKeeper as a Gateway or Proxy

When a distributed system based on the VisiBroker ORB is deployed over the Internet or Intranet, there are many security restrictions that can apply to the system, including:

- server-side firewalls preventing clients from accessing certain server hosts.
- client-side firewalls preventing outgoing connections.
- client-side firewalls prohibiting protocols other than HTTP.

GateKeeper, along with the VisiBroker ORB, provides mechanisms to work with these restrictions based on the OMG CORBA Firewall specification by acting as a gateway or proxy between the client and the server. When certain restrictions prevent the client from connecting directly to the server, the client can choose to connect to GateKeeper. The client can send messages to GateKeeper which will forward the messages to the server.

When certain restrictions prevent the server from connecting back to the client to do callbacks, the server can choose to connect to GateKeeper. The server can send callback messages to GateKeeper which will forward the messages to the client.

In short, GateKeeper provides the following features:

- Proxy to overcome firewalls

- Callback enabling
- Location transparency
- **Java:** HTTP tunneling

Additional capabilities of GateKeeper

In brief, the additional capabilities of GateKeeper are:

- **Java:** Acts as a simple Web Server to load java classes. Java sandbox security prevents unsigned Java applets from communicating with servers other than the ones running on the host machine from which the applets were downloaded. GateKeeper can be configured to overcome this problem.
- **Java:** BootStrapping. GateKeeper can run as a servlet inside any Web Server that supports servlets. This configuration enables IIOP over HTTP (HIOP) and is useful for Java clients.
- Load Balancing and Fault Tolerance. A master GateKeeper and one or more slave GateKeepers can be clustered together and viewed as a single GateKeeper by the clients. This configuration provides the flexibility to balance the load and allows some degree of fault tolerance.
- Customizable IP-based access control. GateKeeper can be configured to deny or grant accessibility based on criteria such as operations, signed by, and so forth.

Note

For more details on GateKeeper configurations, see “Advanced features.”

Primary Use of GateKeeper

GateKeeper is primarily used as a proxy to overcome firewall and transport restrictions. In addition, GateKeeper acts as a Web Server and also incorporates load balancing and access control. GateKeeper, however, should never be used like a full-fledged Web Server, a full-fledged load balancing system, nor a full-fledged access control system. GateKeeper should instead complement its full-fledged counterparts.

Installing GateKeeper

GateKeeper is shipped as a component of VisiBroker. GateKeeper requires the following components:

- VisiBroker Smart Agent
- VisiBroker ORB Libraries
- VisiBroker GateKeeper properties file
- VisiBroker Console

Note

GateKeeper is a stand-alone process. It does not require any of the CORBA IDL compilers.

Starting GateKeeper

The choice of the directory in which to start GateKeeper is determined by how it is being used.

- As an IIOP proxy server for a firewall

- As a Web Server to support HIOP
- In combination with a separate Web Server to support HIOP for VisiBroker for Java

If you use GateKeeper as an IIOP proxy, consult your firewall administrator because the firewall administrator typically is in charge of proxies.

If you use GateKeeper as a complementary Web Server, Borland recommends that you start GateKeeper in the same directory as the Java applets' code base. You can either start GateKeeper at the command line or as a Windows/NT service with the first two features listed above.

If you use GateKeeper in combination with a separate Web Server, you can start GateKeeper as a servlet in the Web Server.

Starting GateKeeper from the command line

Use the following command to start GateKeeper:

```
prompt> gatekeeper
```

Note

Before you can start GateKeeper from the command line, you must first ensure that your `CLASSPATH` setting includes `servlet.jar` in its path.
You can locate `servlet.jar` under the Tomcat installation included with VisiBroker, for example:

```
<installdir>/lib/tomcat/common
```

where `<installdir>` represents the root directory location in which VisiBroker is installed, such as: `C:\visibroker` on Windows.

On Windows, for example, specify `CLASSPATH` as an environment variable and include `servlet.jar` in the search path.

When you start GateKeeper, you will see a start up message followed by a series of messages indicating which services are being started. An example of this series of messages follows.

```
Sun Feb 16 23:43:28 2003: Starting GateKeeper for VisiBroker ...
Sun Feb 16 23:43:31 2003: Request Forwarding Service is started.
Sun Feb 16 23:43:31 2003: Administrative Service is started.
Sun Feb 16 23:43:31 2003: IOR is stored in GateKeeper.ior.
Sun Feb 16 23:43:31 2003: GateKeeper for VisiBroker is started.
```

Command line options

When using the `gatekeeper` command, the following command line options are allowed::

Option	Description
`-props file_name`	Indicates the name of the GateKeeper's properties file. You can include the entire path when you specify the file name. The default location for this file is the directory where you installed GateKeeper. The default name for this file is `GateKeeper.properties`.
`-J-D<Property-name>=<value>`	Specifies a property of GateKeeper at startup.
`-h, -help, -usage, -?`	Displays usage information.
`-quiet`	Specifies for GateKeeper to not generate output.

Running GateKeeper as an NT service

You can install GateKeeper as an NT service. Before you do so, make sure that you can run GateKeeper from a DOS prompt on your target NT platform.

To install GateKeeper as an NT service, type the following command at a command line, where `servicename` is the name of the GateKeeper you are installing.

```
gatekeeper -install "servicename"
```

If you use the `-props` option to specify a properties file, make sure you include the full path name of the properties file you specify.

After you've installed GateKeeper as an NT service, you can start it using the standard Services control panel.

Removing GateKeeper as an NT service

To remove a GateKeeper NT service, use the following syntax at a command prompt:

```
gatekeeper -remove "servicename"
```

Running GateKeeper as a servlet in a Web Server

GateKeeper can run as a servlet inside any Web server that supports servlets. GateKeeper is started with a special HIOP listener whose purpose is to generate the right HIOP component in the GateKeeper's IOR. The HIOP component should contain the Web server's host, port and the path to the GateKeeper servlet. The client will send HIOP requests to the GateKeeper as specified in the HIOP component. The benefit of this feature is in deployment and packaging to allow tighter integration with other components of the system such as a Web server and Borland Partitions.

Generally, there is no significant performance benefit in running GateKeeper as a servlet under the Web Server because all tunnelled requests still go through GateKeeper in the same way they do when GateKeeper is run as a stand-alone process.

Note

If you run GateKeeper as a servlet instead of from the command line, you will lose some administrative capabilities as well as GateKeeper output capabilities.

Managing GateKeeper

The VisiBroker Console enables you to set GateKeeper's properties to meet the requirements of your networked system. GateKeeper's properties are kept in a properties file that GateKeeper references at startup.

2

Configuring GateKeeper and internetworking devices

This section describes how to configure GateKeeper and internetworking devices to allow communications between client objects and server objects across networks, starting with a explanation of where GateKeeper can be deployed.

Where to deploy GateKeeper

This section describes some basic principles used to identify the correct location of where to deploy GateKeeper.

Gather the following information:

- client location
- server location and the server's listener port
- networks connecting the client and server
- firewall, router, and gateway configurations in the connecting networks

Find a connecting path between the client and server; the path may cross multiple networks. To enable the client to contact the server, there must be a connecting path. Otherwise, the client cannot communicate with the server.

Client and server on the same network

When the client and server are located on the same network, the client can always contact the server directly. GateKeeper, however, may still be required in some circumstances; as in the two cases shown in the following examples. If GateKeeper is required, deploy GateKeeper on any host in the same network.

Case 1: Restricted client transport type

Transport types that a client can use to connect to a server can be restricted using the client side properties. GateKeeper is required when:

- a client always connects through a proxy (`vbroker.orb.alwaysProxy`)
- a client always use HTTP-tunneling mode (`vbroker.orb.alwaysTunnel`)

See “Configuring client properties” for details.

Case 2: Java sandbox security

Java sandbox security prevents unsigned Java applets from communicating with server objects located on servers other than the ones running on the host from which the applets were downloaded. In this case, GateKeeper is required as a gateway between the client and server to overcome the restriction of Java sandbox security.

Client and server on adjacent networks

When the client and server networks are adjacent to each other, the two networks are connected using an internetworking device such as a gateway or router. In some cases, a firewall may exist in either network or both networks. To simplify the description, we will consider the firewall as part of the internetworking device. The internetworking device is responsible for forwarding and routing the messages between the two networks. It can also block certain messages from crossing the networks; this is the role of a firewall. The transport types that a client uses to connect to a server can be restricted using the client's property.

GateKeeper is required when

- a client always connects through a proxy
- a client always uses HTTP-tunneling mode

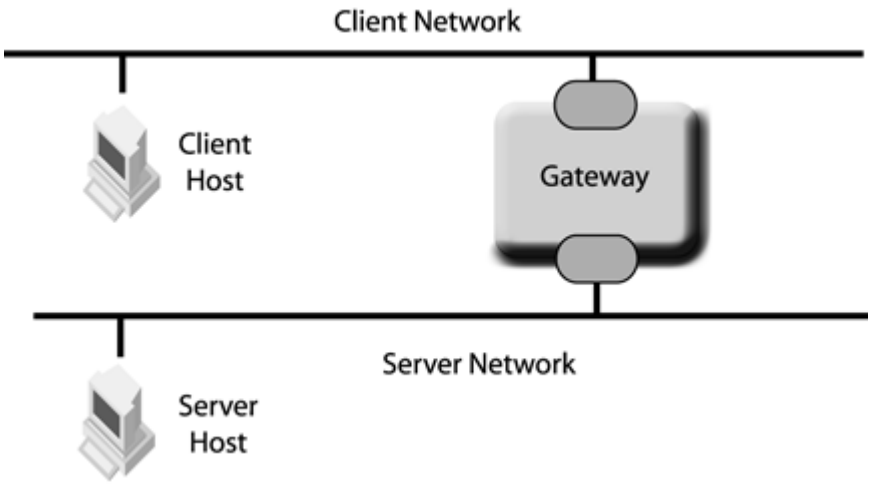
See “Configuring client properties” for details.

Case 1: Java sandbox security

Java sandbox security prevents unsigned Java applets from communicating with server objects located on servers other than the ones running on the host from which the applets were downloaded. In this case, GateKeeper is required as a gateway between the client and server to overcome the restriction of Java sandbox security.

The following figure shows the client and server on adjacent networks.

Figure 2.1 Client and server on adjacent networks



Case 1: Restricted client transport type

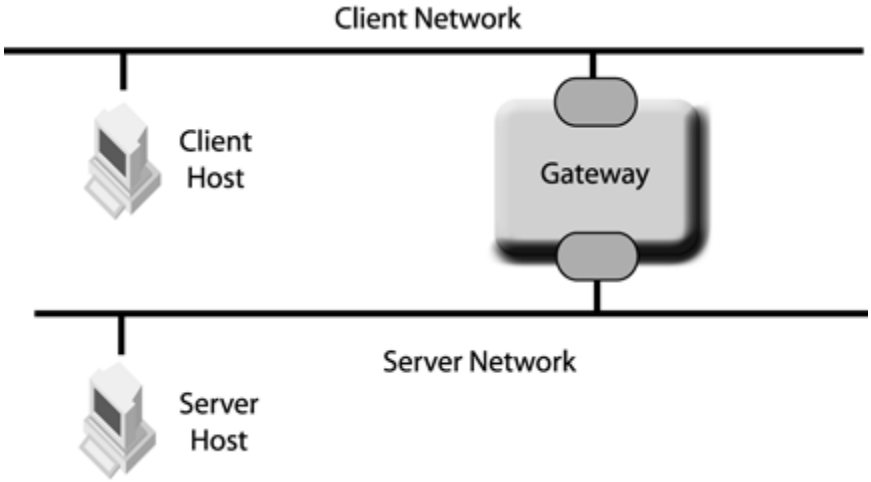
For a client's message to reach the server, the internetworking device must forward the message from the client network to the server network. To find an appropriate location to deploy GateKeeper, determine the type of messages that the internetworking device can forward from the client network to the server network.

The following cases illustrate all the possible locations to deploy GateKeeper for adjacent client and server networks.

Case 1: No GateKeeper required

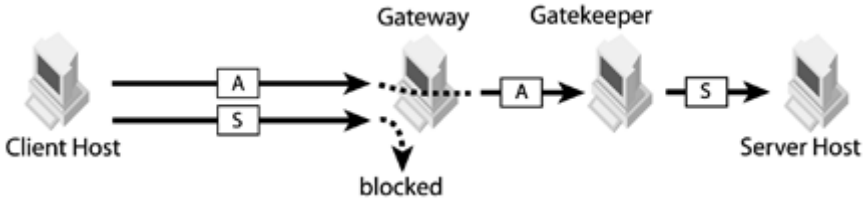
GateKeeper is not required when the gateway can forward all client messages from the client network to the server network.

The following diagram shows a client which sends a message of type A to the server, which listens to type A messages. The gateway forwards the message (type A) to the server network. The server then receives the message (type A). Common examples of type A messages are IIOP and IIOP/SSL.



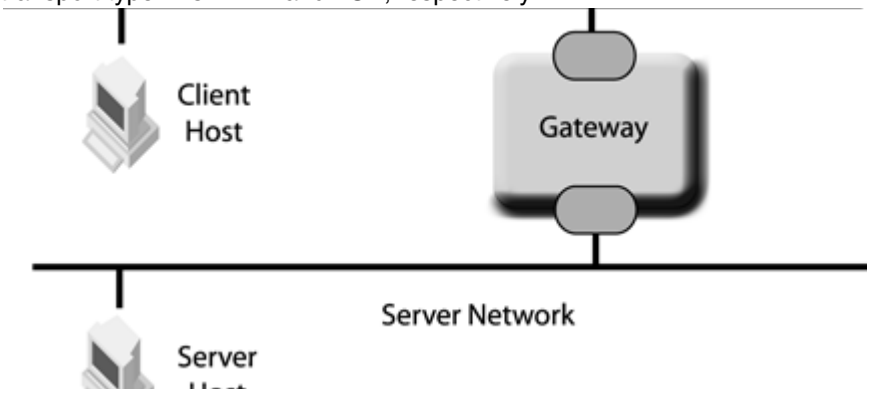
Case 2: GateKeeper in a server network

The following diagram shows a server that listens to messages of type S. The gateway blocks messages of type S but can forward messages of type A from the client network to the server network. If the client sends a message of type S to the server, it will be blocked by the gateway. Instead, the client has to send messages of type A so that the gateway can forward the message to the server network. GateKeepePr is required in the server network to act as a proxy. The client communicates with GateKeeper using type A message and GateKeeper in turn communicates with the server using type S message. An example of type A and S is HTTP and IIOP, respectively. An example for this scenario is HTTP Tunneling mode, where IIOP packets are not allowed, but HTTP packets are allowed by the Gateway/Firewall.



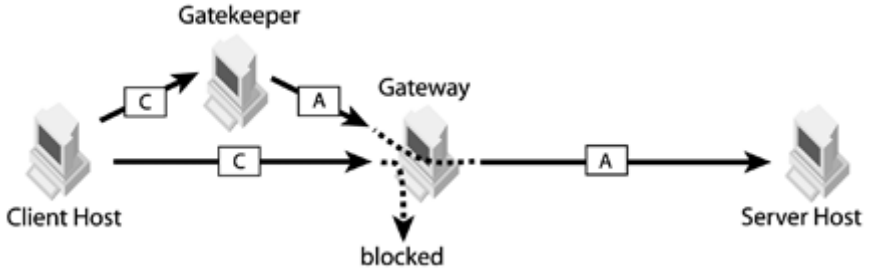
Case 3: GateKeeper in a client network

Server listens to messages of type A but client can only use transport type C to communicate with the server. The gateway blocks messages of type C but forwards messages of type A. A GateKeeper is needed in the client network. Client communicates with GateKeeper using transport type C and GateKeeper communicates with the server using transport type A. The gateway forwards the type A message from the GateKeeper to the server network. An example of transport type C and transport type A is HTTP and IIOP, respectively.



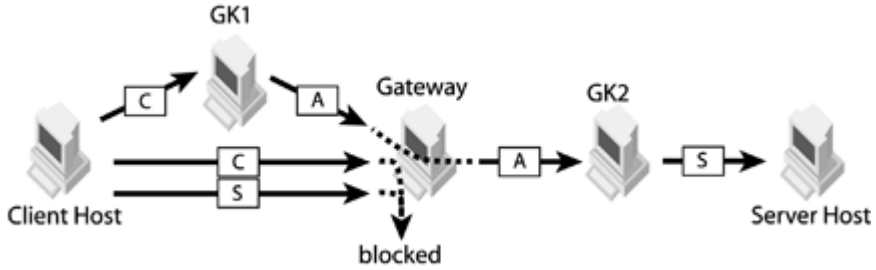
Case 4: GateKeeper in both networks

The gateway blocks both the messages (type C) sent by clients and the messages (type S) that the server can listen to. The gateway can forward another type of message (type A). Therefore, GateKeeper is required in both client and server networks. The client communicates with GK1 using message type C . GK1 communicates with GK2 using message type A, which can be forwarded by the GK2 which in turn communicates with the server using message type S. An example of message type C is HTTP, message type A is SSL and message type S is IIOP.



Case 5: GateKeeper in internetworking device (dual-homed)

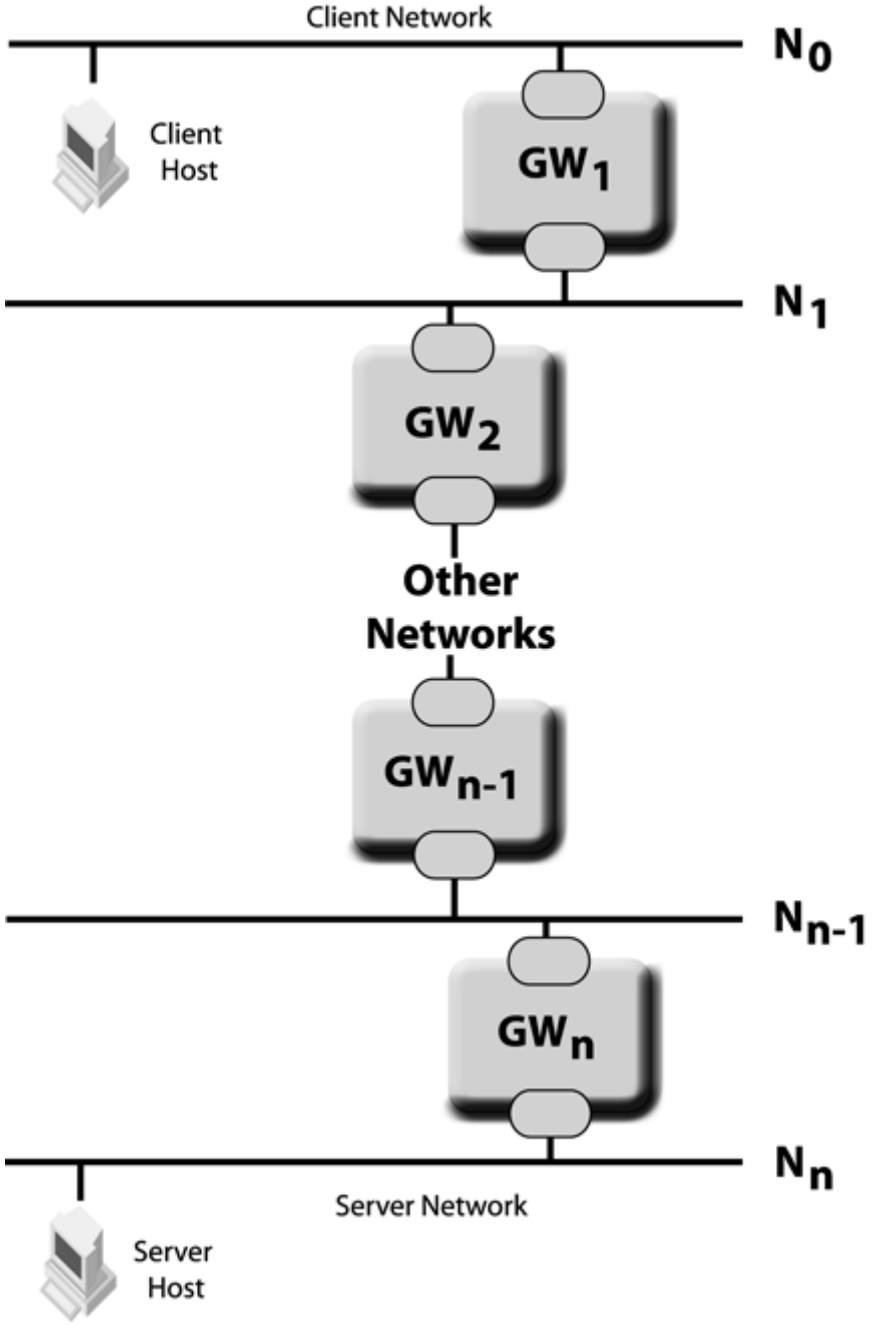
Installing GateKeeper on a dual-homed host works similar to deploying GateKeeper on the server network (case 2). The difference is that GateKeeper always listens to the exterior network for client messages. If a client located in the interior network needs to bind to a server using the same GateKeeper, the message must first be forwarded to the exterior network before it can reach the GateKeeper listener. An example of type A and type S messages are HTTP and IIOP, respectively.



Multiple networks between client and server

In a more complex environment, multiple networks exist between the client and the server networks. Each pair of adjacent networks is connected using an internetworking device.

Figure 2.2 Multiple networks between client and server.



For illustration purposes, the client network will be numbered as N_0. The network adjacent to the client network will be numbered as N_1, the next adjacent network as N_2 and so on until the server network. The server network will be numbered as N_n in the following discussions. Replace n with the actual number depending to the network configuration. Also, the internetworking device between network N_{n-1} and N_n is numbered as GW_n.

Clients can use different transport types to communicate with servers. Examples of transport types are IIOP, IIOP/SSL, HTTP and HTTPS. For each valid transport type, locate the furthest network that the client message can reach. The client located in network N_0 sends a message to network N_0. GW_1 may or may not forward the message to network N_1. The message can reach network N_1 if GW_1 can forward the message from N_0 to N_1. Subsequently, GW_2 may or may not forward the message to network N_2. Traverse the networks starting from the client network, then moving towards the server network. Mark the last network that the message can reach as N_c. In other words, GW_{c+1} cannot forward the message to the network N_{c+1}.

A server has one or more listener ports. Each port listens to one type of messages from clients. As an example, a server with an IIOP listener port and an SSL listener port will use the IIOP port to listen to IIOP messages and the SSL port to listen to IIOP over SSL messages. For each listener port, find the furthest network from the server to which a client message can reach the server. Mark the furthest network as N_s. In other words, a client located in network N_s is able to send a message to the server.

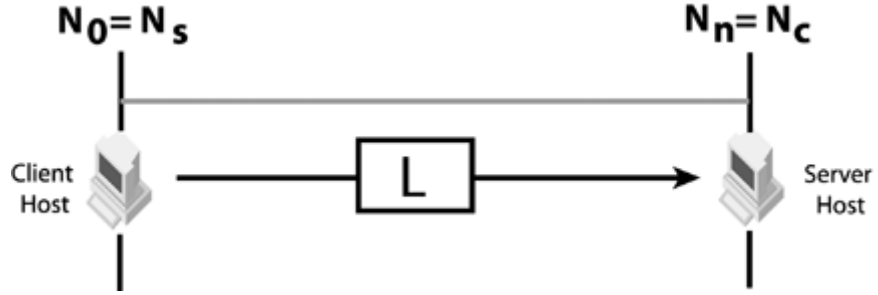
Note

If callback for VisiBroker 3.x style is required, an additional condition is required for N_c and N_s. The callback message from the server (from network N_n) must be able to reach the network N_s. When GateKeeper is used, the client must be able to set up a callback communication channel to network N_c.

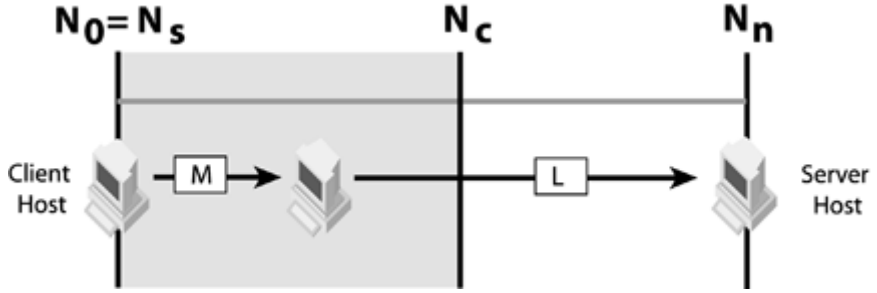
Case 1: Server can receive messages from the client network, s=0

Assume the server listens to transport type L. Messages of transport type L from the client network can reach the server network and subsequently the server.

If the client can send messages using transport type L, then GateKeeper is not required because client messages of type L can be forwarded to the server network. For example, the server listens to IIOP and the client can send IIOP messages. The client's IIOP messages can be forwarded to the server without being blocked by any firewalls, gateways or routers.



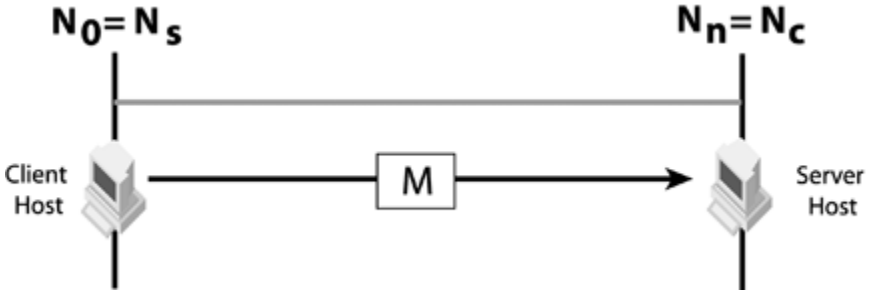
If the client cannot send messages using transport type L, deploy GateKeeper on a network within N_0 and N_c to proxy client messages of other transport types (M) to transport type L. For example, the server listens to IIOP and the client can only communicate using IIOP over HTTP.



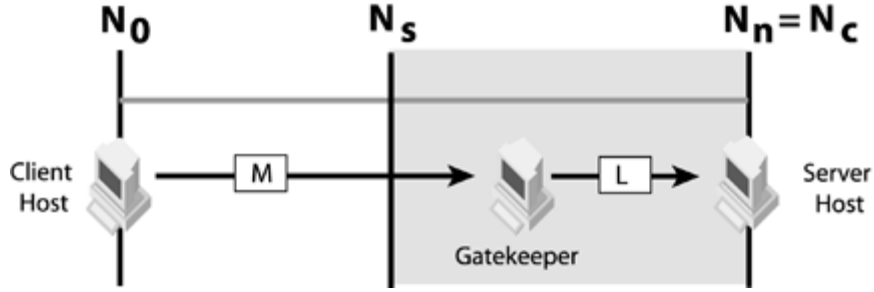
Case 2: Client messages can reach the server network, c = n

Client messages of a particular transport type (M) can reach the server network. GateKeeper is not required if the client transport type is one of the server listening

transport types. For example, the client sends IIOP messages and the server also listens to IIOP.

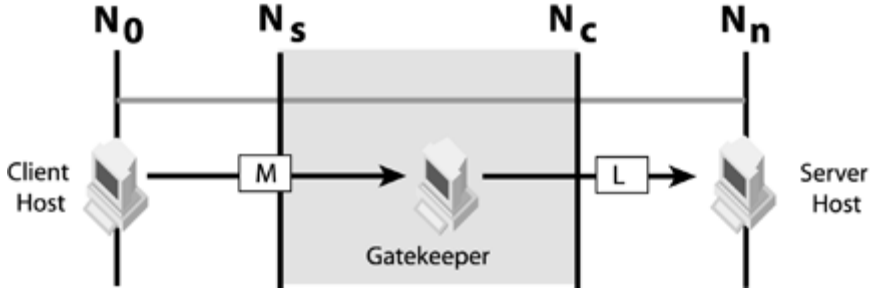


If the server does not listen to the client's message transport type, GateKeeper is required in any network within N_s and N_n. GateKeeper acts as a proxy to relay client messages of type M to one of the server listener types (L). For example, M (client message transport type) is IIOP over HTTP and L (server listener type) is IIOP.



Case 3: Overlapping of reachable networks by client and to server, c >= s

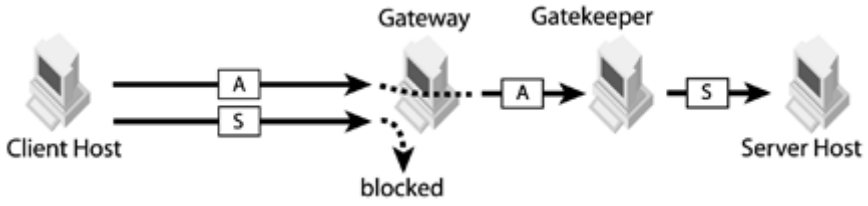
When c >= s, the client transport type (M) and the server listener type (L) must be different. Deploy GateKeeper in any network between N_s and N_c inclusively. In this case, GateKeeper acts as a proxy to relay client messages of type M to the server listener port of type L. As an example, the client's IIOP over HTTP messages can reach the networks up to N_c. IIOP messages sent from any network between N_s and N_n can reach the server. Deploying GateKeeper in between N_s and N_c will help bind the client's IIOP over HTTP messages to the server's IIOP listener port.



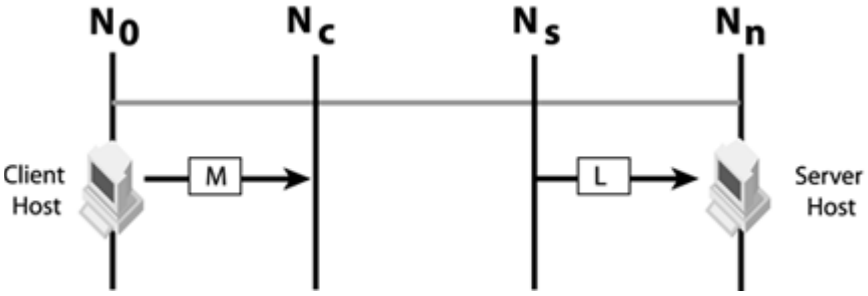
Case 4: No overlapping of reachable networks by client and to server, c < s

Check if GateKeeper chaining is possible or not. See “Chaining of GateKeepers” for details of GateKeeper chaining. GateKeeper chaining is possible only when there is another transport type (K) available for the two GateKeepers to communicate successfully from N_c to N_s. Deploy one GateKeeper on network ($N_{c)}$ and another GateKeeper on network N_s. After which, chain them together. For example, client sends IIOP over HTTP messages, the server listens to IIOP messages and both GateKeeper instances can use SSL to communicate with each other. The client

connects to GateKeeper 1 using HTTP, GateKeeper 1 communicates with GateKeeper 2 using SSL, and GateKeeper 2 communicates with the server using IIOP.



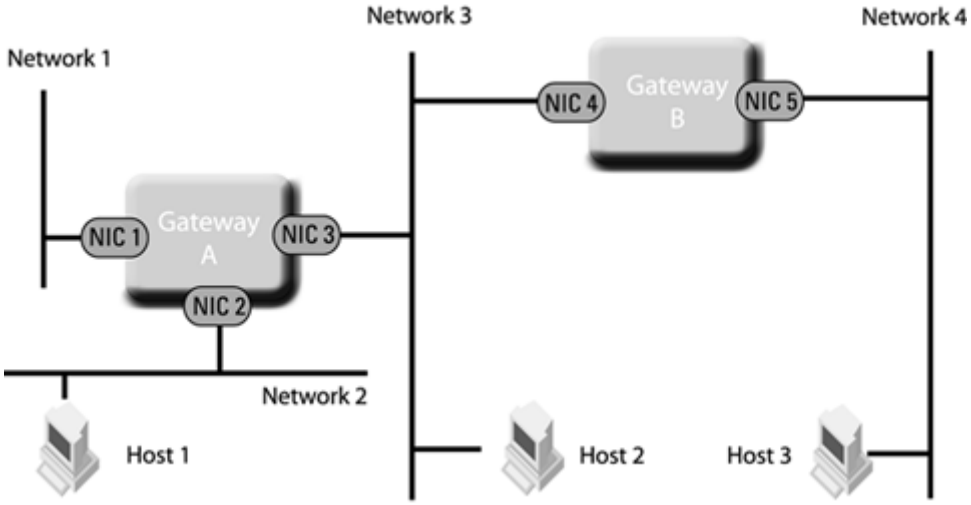
If chaining is not possible, there is no suitable network to deploy GateKeeper. The internetworking devices connecting networks N_c and N_s must be reconfigured so that the appropriate type of messages can be forwarded from N_c to N_s. After which, locate the new N_c and N_s, and refer to the previous cases accordingly.



Configuring a multi-homed host

A multi-homed host or router connects two or more physical networks. It has multiple network interfaces; also known as Network Interface Cards (NIC). Each NIC connects to one network. The multi-homed host allows communication between the connected networks. The following diagram shows a network configuration with two multi-homed hosts (Gateway A and Gateway B).

Figure 2.3 Multi-home machine network configuration.



To enable a multi-homed host to route data packets from one network to another correctly, IP-forwarding must be enabled and its routing table must be configured correctly. Similarly, the routing tables on the hosts must be configured correctly.

Assuming a client located on Host 1 is trying to communicate with a server located on Host 3, the client on Host 1 will first send the message to Host 3 on Network 2. Gateway A will accept the message on NIC 2 and route it to Network 3 using NIC 3. Gateway B will then accept the message on NIC 4 and route it to Network 4 using NIC 5. The message will then reach the server object on Host 3. This communication can happen only if IP-forwarding is enabled and all the routing tables are configured correctly.

Enable IP-forwarding

The multi-homed host must enable IP-forwarding to forward data packets from one network to another. If IP-forwarding is disabled, the multi-homed host cannot forward or route data packets from one network to another.

Routing table

One entry of the routing table is used for one destination host or network. Every entry must contain: information about the:

- destination host or network
- gateway it should contact.
- interface where the data packets should be sent out.

The following tables show examples of routing tables for the sample network configuration.

Destination	Gateway	Interface
Network 1	Gateway A	NIC 1
Network 2	Gateway A	NIC 2
Network 3	Gateway A	NIC 3
Network 4	Gateway B	NIC 3

Destination	Gateway	Interface
Network 1	Gateway A	Host 1
Network 2	Host 1	Host 1

Destination	Gateway	Interface
Network 3	Gateway A	Host 1
Network 4	Gateway A	Host 1

Destination	Gateway	Interface
Network 1	Gateway A	Host 2
Network 2	Gateway A	Host 2
Network 3	Host 2	Host 2
Network 4	Gateway B	Host 2

A routing table in the multi-homed host stores the routing information about which NIC to forward data packets to. The gateway information is used to contact the next gateway in the route. (Refer to the routing table for Gateway A in the example described above.) Using NIC 3, Gateway A has to contact Gateway B to route packets to Network 4.

Hosts also have their own routing table. The gateway information is essential for the host to contact the correct gateway which can route the packet correctly. (Refer to the routing table for Host 2). Host 2 needs to contact Gateway A to reach Network 1 and Network 2. But, Network 2 has to contact Gateway B in order to reach Network 4.

Use the following methods to verify if the routing table is configured correctly:

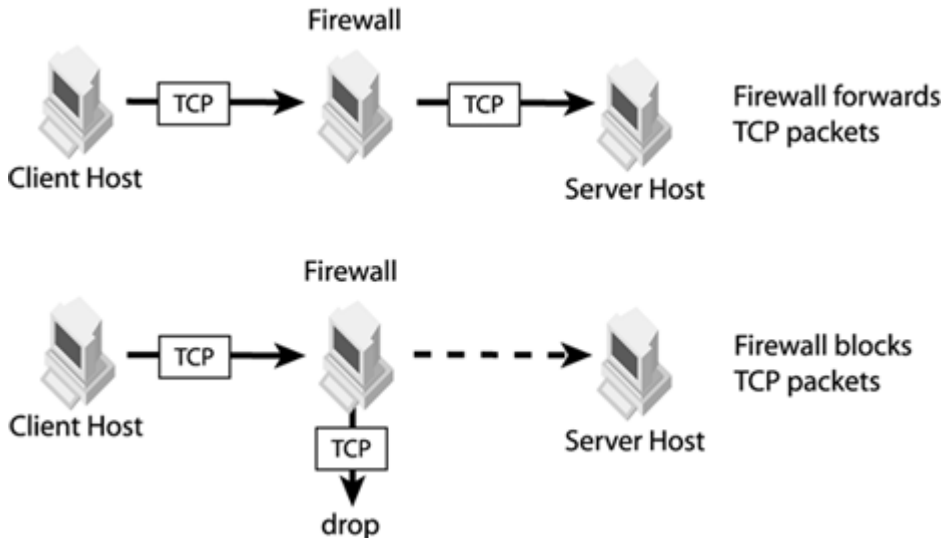
- Print the routing table.
- Ping the relevant host.
- Perform a trace route to the relevant host.

Configuring the firewall

A firewall is a network device that performs filtering of data packets. A firewall inspects every data packet it receives and then either forwards the packet or drops it depending on the firewall's security policy.

Case 1: Restricted client transport type

The following figure shows an example of firewall packet filtering.



The firewall's security policy usually inspects the message type, message source, and message destination to perform filtering. Firewalls are capable of applying packet-filter rules based on the type of service (example: stream-oriented or datagram-packets) and the underlying protocol type (example: IP, ICMP, TCP, UDP). Suppose that the firewall identifies the communication path as a TCP packet stream, then the firewall

can apply the packet-filtering rule defined in the security policy to decide if the packet should be allowed or dropped. The TCP packet streams can carry different kinds of data or payloads (example: HTTP, IIOP, FTP, SSL, etc). In general, each stream is assigned a unique port number, and it carries only one class or type of message. For example, IIOP messages can be carried on TCP Port 683 packet stream. Similarly, HTTP messages can be carried on TCP Port 80. The firewall may allow TCP Port 80, but may not allow TCP Port 683 depending on the packet-filtering rules. Using special techniques, a TCP packet stream can carry more than one type of messages. GateKeeper uses a special technique, called HTTP Tunnelling, to embed IIOP messages within HTTP messages to be carried over TCP packet streams.

When a firewall exists in the communication path between the client and server, the firewall may either forward or drop the data packets sent from the client to the server. For a successful communication between the client and server, the firewall must forward the client's messages to the server. The server can be a user application, GateKeeper, or other VisiBroker service providers such as the Smart Agent and the Naming Service. Configure the firewall to forward client's messages sent to the server's listener port.

Using Network Address Translation (NAT)

A multi-home host, router, and firewall can also perform NAT in addition to their specialized functions. NAT can translate the source host address, source port number, destination host address, and destination port number found in every network packet.

On the client side, the firewall usually translates the source host address. This method is commonly used to share a limited number of internet IP address.

On the server side, the firewall may translate the destination host address and/or the destination port number. This hides the real destination host address from external parties. It provides the flexibility to change the destination host address without notifying all external parties that must access the server. This flexibility holds true for the port number as well.

GateKeeper supports only static NAT, it does not support dynamic NAT. In static NAT, the translation is based on a predefined mapping table in which every address and port is always translated to a fixed value. In dynamic NAT, some rules can be set to translate addresses and ports to a range of values where the exact translated address of the network packet cannot be pre-determined because it can be any address within a given range.

See “Configuring user programs” for details on how to configure server objects to use TCP firewall with NAT. Be sure that the NAT translation mappings are added into the NAT device for successful communication between client and server objects.

With NAT, the routing tables for all the gateways involved must be configured to account for any fake network addresses in use. If not, the data packets having fake destination addresses will not be routed correctly. In addition, firewalls must be configured to forward messages to any fake destination host addresses and fake ports used in NAT. If firewalls block the fake address or fake port, a packet will not reach its destination.

Configuring GateKeeper

The following sections describe how to configure GateKeeper ports and services. Listener ports are the most common parameters that must be configured. Different firewalls usually do not open the same range of ports for communications. GateKeeper has many services and some of them must be enabled before they can be used.

Listener ports

The following properties specify GateKeeper's exterior IIOP and HTTP listener port numbers. These are the ports on which GateKeeper listens to client requests.

```
vbroker.se.exterior.scm.ex-iiop.listener.port=683
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

If GateKeeper is deployed behind a firewall, external clients can only contact GateKeeper if the firewall allows forwarding of IIOP or IIOP over HTTP messages through ports 683 and 8088, respectively. If the firewall can only allow other port numbers because of security restrictions, the GateKeeper listener ports must be configured to use the authorized ports on the firewall.

Administrative service

GateKeeper's administrative service provides the ability for you to use the VisiBroker Console to manage and configure GateKeeper. The administrative service allows dynamic configurations of GateKeeper while GateKeeper is active. The following properties specify the administrative service port numbers; 0 and 9091 are the default values for IIOP port and HTTP port, respectively. The value 0 tells GateKeeper to pick a port at random when it starts.

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=0
vbroker.se.iiop_tp.scm.hiop_ts.listener.port=9091
```

Enabling callbacks (VisiBroker 3.x style)

The callback feature (VisiBroker 3.x style) has been replaced with bidirectional support in VisiBroker versions 4.x and later. For GateKeeper to support clients that still use VisiBroker 3.x callbacks, the following properties settings are required:

```
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
```

After setting the above properties, GateKeeper activates its interior server engine to receive callback messages from the server. The listener can be configured using the `in-iiop` and `in-ssl` SCMs. In addition, a callback listener is activated for a client to establish an additional communication channel for callback messages. See “VisiBroker 3.x style callback” for details on specifying the listener port and additional related information. Be sure the selected ports are reachable from the client and the server by ensuring that these ports are not blocked by any firewalls.

Enabling pass-through connections

The following property enables pass-through connections in GateKeeper.

```
vbroker.gatekeeper.enablePassthru=true
```

If the client requests a pass-through connection, GateKeeper will not examine any messages that pass between the server and client. When the above property is set to `false`, GateKeeper binds the client to the server using normal (non-pass-through) connections even when the client requests a pass-through connection. In this case, GateKeeper examines the exchanged messages for routing and binding purposes.

The following properties are provided to help configure pass-through connections in GateKeeper:

```
vbroker.gatekeeper.passthru.blockSize=16384
vbroker.gatekeeper.passthru.connectionTimeout=0
vbroker.gatekeeper.passthru.logLevel=0
vbroker.gatekeeper.passthru.streamTimeout=2000
vbroker.gatekeeper.passthru.inPortMin=1024
vbroker.gatekeeper.passthru.inPortMax=165535
vbroker.gatekeeper.passthru.outPortMin=0
vbroker.gatekeeper.passthru.outPortMax=65535
```

See “Support for pass-through connections” for more information about the above properties.

Caution

The pass-through feature heavily taxes the resources of GateKeeper. If you choose to use this feature, be sure to configure GateKeeper with sufficient memory and increased sockets.

Enabling the location service

GateKeeper provides a location service for clients, such as applets, that are unable to communicate directly with the Smart Agent (osagent) because of Java sandbox security or existing firewalls. The location service lets the clients “bind” to the server through GateKeeper.

```
vbroker.gatekeeper.locationService=true
```

Specifying the Smart Agent (osagent)

GateKeeper uses the Smart Agent to locate server objects. GateKeeper can automatically locate the Smart Agent if one is located on the same network. When there is no Smart Agent running on the same network where GateKeeper is running, the location of the Smart Agent must be specified explicitly. You can also specify additional Smart Agents running on other networks.

```
vbroker.agent.addr=<host>
vbroker.agent.addrfile=<filename>
vbroker.agent.port=<port>
```

The first property specifies the host IP address of the Smart Agent. The second property specifies the file that defines a list of hosts running Smart Agents. The third property specifies the `OSAGENT_PORT`. The default value for the first two properties is null, which tells GateKeeper to contact the Smart Agent running on the same network.

See “Using the Smart Agent” in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more details about Smart Agent settings and other methods of setting Smart Agent parameters.

Specifying the Object Activation Demon (OAD)

The OAD service enables GateKeeper to automatically start servers to which it needs to bind. In such cases, the server is registered with the OAD service, but is accessible only through GateKeeper (when an Applet invokes a server, for example). To use the OAD service, GateKeeper must load the OAD IOR. The following property tells GateKeeper where to locate the OAD IOR.

```
vbroker.oad.iorFile=<OAD IOR>
```

See “Using the Object Activation Daemon” in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more information about OAD.

Configuring GateKeeper server engines

GateKeeper contains a few default server engines. Each server engine contains at least one server connection manager (SCM).

- The exterior server engine enables GateKeeper to bind client objects to server objects. It contains two default SCMs which are named `ex-hiop` and `ex-iiop`.
- The interior server engine provides callback services and is only available when callback is enabled. It contains two default SCMs which are named `in-iiop` and `in-ssl`.
- The iiop_tp server engine provides the administrative service. It contains two default SCMs, which are named `hiop_ts` and `iiop_tp`.

See “Exterior server engine” , “Interior server engine” and “Administration” for the full list of properties for the above SCMs.

Security services

Start GateKeeper with the following properties to enable IIOP/SSL and IIOP over HTTPS:

```
vbroker.security.disable=false
vbroker.orb.dynamicLibs=com.borland.security.hiops.Init
vbroker.se.exterior.scms=ex-iiop,ex-hiop,ex-ssl,ex-hiops
```

- The `vbroker.security.disable=false` property enables the required security packages into the VisiBroker ORB of the GateKeeper.
- The `vbroker.orb.dynamicLibs=com.borland.security.hiops.Init` property loads the additional HIOPS package, which allows IIOP messages over HTTPS; it is loaded separately.
- The `vbroker.se.exterior.scms=ex-iiop,ex-hiop,ex-ssl,ex-hiops` property adds the SCM ex-ssl and ex-hiops into the exterior server engine.

The unused SCM can be removed from the SCM list so that only required SCMs are started. However, scm ex-iiop and in-iiop can not be removed from the list when they initially exist.

To make sure all communication is encrypted, you can disable the nonsecure listener ports such as IIOP and HTTP as follows:

```
vbroker.se.exterior.scm.ex-iiop.listener.type=Disabled-IIOP
vbroker.se.exterior.scm.ex-hiop.listener.type=Disabled-IIOP
```

The IIOP/SSL and HTTPS listeners can be configured using the SCM properties prefixed with `vbroker.se.exterior.scm.ex-hiops` and `vbroker.se.exterior.scm.ex-ssl`. For a comprehensive list of these SCM properties, refer to the "GateKeeper properties."

SSL Transport Identity and Trustpoint

For SSL, transport identity is optional as SSL negotiation still can make use of a Diffie Helman key agreement algorithm without someone's public key.

However, without transport identity clients configured with `peerAuthenticationMode require` and `require_and_trust` will not connect. Additionally, as an SSL server, if GateKeeper itself does not have a client transport identity, it may not require client transport identities.

Installing SSL Identity using Wallet properties

The simplest way of installing certificates in GateKeeper is by using the following wallet properties:

```
vbroker.security.wallet.type=Directory:<path_to_identities>
vbroker.security.wallet.identity=<username>
vbroker.security.wallet.password=<password>
vbroker.security.trustpointsRepository=Directory:<path_to_trustpoints>
```

Installing SSL Identity on GateKeeper using Certificate Login

Apart from using simple wallet and trustpoints property sets, SSL Identity can be installed on the GateKeeper during startup by means of credential acquisitions (login). In the acquisition, the user must answer questions about files and directories, where the certificates, private key and trusted root certificates are stored. The password to decrypt the private key will definitely be asked.

The files and directories asked in the login conversation vary based on the type of certificate storage. The default storage is determined by JDK security settings in the following file:

```
${JAVA_HOME}/jre/lib/security/java.security
```

Out of the JDK box, `jks` is set as java keystore (jks):

```
#
# Default keystore type.
#
keystore.type=jks
```

For PKCS#12 storage, the above can be changed to string `pkcs12`. This storage format is only a single file, which contains certificates, trusted certificates and a private key. Please consult the JDK keytool manual.

For certificate login, the followings needs to be explicitly set on GateKeeper:

```
vbroker.security.login=true
vbroker.security.login.realms=<realm list>
```

In the realm list, among other realms, there needs to be `Certificate#CLIENT` and/or `Certificate#SERVER` and/or `Certificate#ALL`.

- `Certificate#CLIENT` is an SSL identity that is used for outgoing SSL connections,
- `Certificate#SERVER` is for incoming SSL connections,
- and `Certificate#ALL` can be used for both.

One extreme example is when in the <realm list> there appears all three realms. In this case, three different sets of SSL identities will be acquired from the user during GateKeeper startup.

When opening an outgoing SSL connection:

1. first `Certificate#CLIENT` will be used.
2. If none is set in `Certificate#CLIENT`, then `Certificate#ALL` will be used.
3. If there is also none set in `Certificate#ALL`, the outgoing SSL connection will have no identity.

Note

Similar priority also applies to incoming (server) SSL connection.

The identity that is set using a simple wallet property set will always go into `Certificate#ALL`.

Setting peerAuthenticationMode

Use the `peerAuthenticationMode` policy as usual. Set the property as follows:

```
vbroker.security.peerAuthenticationMode=none
```

Applet and Java Webstart

The Java programming language is a powerful tool for the development of programs that are deployed and run on the fly from one central location. This becomes a very powerful feature when combined with CORBA, more specifically with VisiBroker for Java.

Clients code can be downloaded on the fly and installed from a website as either a Java applet or a Java webstart application utilizing Java Network Launching Protocol (JNLP).

VisiBroker settings on a typical applet client

If the client is an applet, the following additional property settings are required:

```
<applet archive=vbjorb.jar,vbsec.jar,lm.jar,sanct4.jar,
      sanctuary.jar,code="ClientApplet.class" width="200"
      height="80">
        <param name="vbroker.security.disabled" value="false">
        <param name="vbroker.orb.dynamicLibs"
            value="com.borland.security.hiops.Init">
```

```
...
</applet>
```

Note

1. All VisiBroker jars do not need to be in the GateKeeper http root directory (the current directory where you launch GateKeeper).
2. Licensing jars: `lm.jar,sanct4.jar,sanctuary.jar` are needed only when the applet code creates persistent POAs.
3. When VisiSecure functionality is involved, `vbsec.jar` is needed in the applet's archive list. The applet parameter that enables it is also needed. Optionally, when HIOPS functionality is involved, it needs to be loaded separately using `dynamicLibs` as above.

VisiBroker Application Deployed as a Java Webstart

A Java webstart application can run without a web browser because it has its own launcher, which can be launched directly from a command shell on UNIX or by double-clicking on Windows. This launcher is the default mime handler for application/x-java-jnlp-file which is associated automatically when installing JDK/JRE on Windows and by any other means on UNIX. Therefore, clicking a link on a web page that results in any http response with that mime will launch the installed Java webstart launcher for processing the content of that reply. The content is actually an XML containing information about where to locate the required jars and other information pertaining to running the application. For example, the required java security permissions.

For a typical VisiBroker application deployed as a java webstart, please see the `gatekeeper bank_jws` example.

3

Configuring user programs

This chapter shows how to configure the user programs (clients and servers) to use firewalls and GateKeeper. The settings are configured through the client and server properties. See Appendix A for information on how to set the properties.

Using objects behind firewalls

You may need to configure both programming and runtime environments so that objects can work behind firewalls. Configuring firewall policies for a specific Portable Object Adapter (POA) must be done programmatically. Setting the same firewall policies globally for all POAs, however, can be accomplished using a single property setting and does not require source code modifications.

Programming a single POA

To allow a server to traverse a firewall when you want to configure firewall policies for a specific POA, you must specify a firewall policy on the POA where the server is activated. In particular, the following code must be added to the server. (The following examples use the Bank example as a basis.)

To configure a single POA programmatically:

1 Create the firewall policy:

Java

```
org.omg.CORBA.Any fw_policy_value = orb.create_any();
com.inprise.vbroker.firewall.FirewallPolicyValueHelper.insert(
   fw_policy_value, com.inprise.vbroker.firewall.EXPORT.value);
org.omg.CORBA.Policy firewall_policy = orb.create_policy(
   com.inprise.vbroker.firewall.FIREWALL_POLICY_TYPE.value, fw_policy_value);
org.omg.CORBA.Policy[] policies = {
   firewall_policy,
   rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
};
```

C++
```
CORBA::PolicyList policies;
policies.length(2);
policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy
   (PortableServer::PERSISTENT);
CORBA::Any policy_value;
policy_value <<= Firewall::EXPORT;
CORBA::Policy_ptr fpolicy= orb->create_policy
   (Firewall::FIREWALL_POLICY_TYPE, policy_value);
policies[(CORBA::ULong)1] = fpolicy;
```

2 Apply the policy to the POA on which the server will be activated:

Java
```
POA bankPOA = rootPOA.create_POA("bank_agent_poa", rootPOA.the_POAManager(),
   policies);
```

C++
```
PortableServer::POA_var bankPOA = rootPOA->create_POA("bank_agent_poa",
   poa_manager, policies);
```

Only the root POA takes the default policy, so it can be used to activate any server that must be accessed behind a firewall. You must also create another POA to activate the Account server. Since the Account server should not be bound by clients directly, you should create the POA as a transient POA:

Java
```
 policies = new org.omg.CORBA.Policy[] {
   firewall_policy,
   rootPOA.create_lifespan_policy(LifespanPolicyValue.TRANSIENT)
};

POA accountPOA = rootPOA.create_POA(
   "account_agent_poa", rootPOA.the_POAManager(), policies);
```

C++
```
policies.length(2);
policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy
   (PortableServer::TRANSIENT);
policies[(CORBA::ULong)1] = fpolicy;
PortableServer::POA_var accountPOA = rootPOA->create_POA("account_agent_poa",
   poa_manager, policies);
```

Configuring the firewall policy for all POAs associated with a server

The following property lets you set the firewall policy for all POAs associated with a server:

```
-Dvbroker.orb.exportFirewallPath=true
```

If you specify the `exportFirewallPath` property, you do not need to add a firewall policy when creating a POA and therefore, you do not have to modify the source code.

Loading a firewall package at runtime

The clients and servers working with GateKeeper must load the firewall package and its properties at runtime when it first initializes the ORB which is when the following method is invoked.

Java
```
org.omg.CORBA.ORB.init(String[] args,java.util.Properties property);
```

C++
```
CORBA::ORB_ptr CORBA::ORB_init(int& argc, char *const *argv);
```

Java The following property causes the firewall package to be loaded into the VisiBroker for Java ORB. Gatekeeper does not need to load firewall.Init package as it has defined the firewall components.

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
```

C++ Support for GateKeeper and firewall has been incorporated into the VisiBroker for C++ ORB and no additional libraries are needed to enable it.

Configuring client properties

The method in which a client communicates with a server can be restricted. In particular, using policies and properties, you can specify whether:

- Clients use a GateKeeper as a proxy for the real server

Java:
- Clients communicate to servers in the HTTP-tunneled channel
- Clients communicate to servers through IIOP/SSL

Java:
- Clients communicate to servers through HTTPS-tunneled channel
- Messages pass between client and server connections through a GateKeeper entirely unexamined by the GateKeeper (called pass-through mode)
- Clients choose the preferred transport from those available

Note about HTTP-tunneled channel:

VisiBroker for Java extension for IIOP that is tunneled through http protocol is called HIOP. In this mode, the ORB is able to tunnel only one request at a time. So if there are concurrent threads in the client making simultaneous requests, the ORB serializes those requests. And thus, the requests coming into the server are also serialized. The number of simultaneous requests seen in the server will match the number of Orbs on the client, when client(s) use HTTP tunneling. This is a limitation of using HTTP tunneling on the client.

The following sections show the various policies and their property settings for the clients. You can also use different combinations of these policies to determine how you want your client to communicate to the servers.

Specify always proxy on a client

The following property setting forces clients to use GateKeeper to proxy requests to servers.

Client's Properties:

```
vbroker.orb.alwaysProxy=true
```

The above property is optional. If you do not set it, the client uses the server's IOR to determine whether or not the object is hidden behind a server-side firewall and traverses the firewall accordingly. It is sometimes better to not set the above property, for example, when a client invokes both local objects inside the trusted network and remote objects hidden behind the firewall. Not setting the property enables the client to be more efficient by invoking the local objects directly without going through GateKeeper.

Client's Properties:

```
vbroker.orb.gatekeeper.ior=<IOR>
```

Clients can also specify the GateKeeper IOR using the above property. This method is helpful when a client is not able to locate GateKeeper through a Smart Agent.

Specify HTTP tunneling on a client

The following property setting directs clients to communicate to servers in the HTTP-tunneled channel.

Client's Properties:

```
vbroker.orb.alwaysTunnel=true
```

Limitation of tunneling: The http protocol requests from being multiplexed on same connection. The ORB is able to tunnel only one request at a time. So if there are concurrent threads in the client making simultaneous requests, the ORB serializes those requests. And thus, the requests coming into the server are also serialized.

The number of simultaneous requests seen in the server will match the number of ORBs on the he, when client(s) use HTTP tunneling. This is a limitation of using HTTP tunneling on the client.

The above setting causes the client applet or application to communicate to the GateKeeper through IIOP over HTTP and GateKeeper relays the request to the actual server object through IIOP. Replies from the server object to GateKeeper are communicated through IIOP. GateKeeper then forwards those replies to the client through IIOP over HTTP.

Java Applets should set `vbroker.orb.alwaysTunnel` if the client will be performing HTTP tunneling. Applet clients must set the property `vbroker.orb.gatekeeper.ior` to get the GateKeeper's IOR using URL naming or using a stringified IOR. In addition, the applet clients must not set the `vbroker.locator.ior` property.

Note

You cannot use callbacks with HTTP tunneling.

Caution

HTTP tunneling may not work consistently with various types of proxy servers because of differences that may exist in the implementation of HTTP proxy servers. Please refer to the VisiBroker GateKeeper FAQ on Borland's web site for more information.

Specify secure connections on a client

Client's Properties:

```
vbroker.orb.alwaysSecure=true
```

Clients talk to servers through IIOP/SSL or IIOP over HTTPS.

Client's Properties:

```
vbroker.orb.alwaysSecure=true
vbroker.orb.alwaysTunnel=true
```

Clients only talk to servers using IIOP over HTTPS.

Specify pass-through connections on a client

In this type of connection, GateKeeper does not terminate connections or interpret messages. This type of connection is useful when GateKeeper does not have SSL or the associated certificates to establish trust with the client. In such cases, the client and server negotiate their SSL connection without going through GateKeeper. Therefore, GateKeeper does not interpret messages passed between the client and the server.

Client's Properties:

```
vbroker.orb.proxyPassthru=true
```

GateKeeper's Properties:

```
vbroker.gatekeeper.enablePassthru=true
```

- The `vbroker.orb.proxyPassthru` property sets the value of the ORB-level `PROXY_MODE_POLICY` property. If set to true, all objects using a proxy on the client will request pass-through connections. You can also set the `PROXY_MODE_POLICY` on specific objects so that only those particular objects request pass-through connections.
- The `vbroker.gatekeeper.enablePassthru` property instructs GateKeeper to accept pass-through connections. This property is global to GateKeeper and affects GateKeeper's behavior only.

The `vbroker.orb.proxyPassthru` property tells the client to attempt to acquire pass-through connections from GateKeeper. GateKeeper, however, grants pass-through connections only if the `vbroker.gatekeeper.enablePassthru` property is set to `true`. See “Enabling pass-through connections” for other GateKeeper's pass-through properties.

Enabling pass-through connections

If the `vbroker.gatekeeper.enablePassthru` property is set to `false`, GateKeeper does not allow pass-through connections to be established and clients can only obtain normal (non-pass-through) connections to the server. GateKeeper then examines the messages exchanged between the client and server for routing and binding purposes. The connection will fail if GateKeeper cannot provide an SSL authentication for an SSL message.

Specifying the client bid order

Client's Properties:

The client's bid order specifies the relative importance for the various transports used to connect to the server. The transports that appear first will have higher precedence. The following property setting instructs the client to try the transport with the higher precedence first, whenever it is available, in the server's IOR. When a transport fails, the client will try the next available transport.

```
vbroker.orb.bidOrder=inprocess:liop:ssl:iiop:proxy:hiop:locator
```

In the above example, if the IOR contains both LIOP and IIOP profiles, the client will first try LIOP. Only if LIOP fails will it try IIOP.

Client's Properties:

```
vbroker.orb.bidCritical=inprocess
```

The critical bid has the highest precedence no matter where it is specified in the bid order. If there are multiple critical bids, then their relative importance is determined by the bid order.

Specifying a client callback listener port (for VisiBroker 3.x style)

The following properties specify the listener port of the client for servers to establish VisiBroker 3.x style callback connection. The listener type is set to `Callback-IIOP` to differentiate it from a normal `IIOP` listener.

Client's Properties:

```
vbroker.se.iiop_ts.scm.iiop_tp.listener.port=<port>
vbroker.se.iiop_ts.scm.iiop_tp.listener.type=Callback-IIOP
```

Configuring server properties

Use server properties to construct the server's IOR that is used by clients to establish communication paths to the server.

Specifying the listener port of the server

The following sections describe the property settings used to specify a server's listener ports.

Random listener port

The following property has the default value of 0 (zero) which tells the system to pick a random port number when the server starts.

Server's Properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=0
```

Specific listener port

The following server property assigns the port on which the server will listen to IIOP messages from clients.

Server's Properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<port number>
```

Note

All clients on the same network can establish communication with a server using the port, as specified in the above example, directly. Messages sent by clients on different networks must be forwarded by the gateway or router. If a server allows connections by clients outside the subnet, the router or firewall should be configured to allow messages for the specified port. Conversely, if the server only allows connections from clients on the same subnet, the router or firewall should be configured to block messages for the specified port to prevent unauthorized access by foreign client objects.

Port translation (NAT)

If there is a port translation using Network Address Translation (NAT) from a fake port (also called the proxy port) to the server's real IIOP listener port, use the following property settings to publish the fake port in the server's IOR.

Server's Properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<real_port>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<fake_port>
```

The above settings tell the server to listen to the real port while clients send messages to the fake port. The default value of the `proxyPort` property is 0 (zero), which means no proxy port is used.

Note

A better method of specifying NAT is to use the TCP firewall properties described in following section.

Disabling the IIOP port

Setting the following property will disable the server's IIOP listener port which forces the server to allow client requests on a specified port, such as a secured port like IIOP/SSL. The server will not allow IIOP messages on the published IIOP port.

Server's Properties:

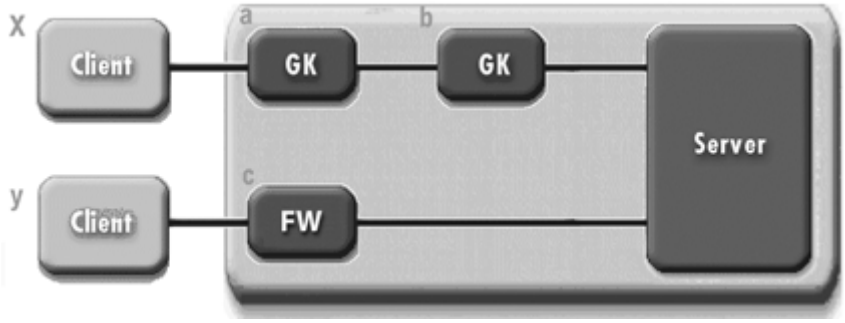
```
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
```

Specifying communication paths to the server

There may be multiple paths for a client's message to reach the server or different paths for messages originating from different clients to reach the same server. All these possible paths have to be configured in the server's properties so that the generated IOR has the information needed for clients to send messages to the server.

The following diagram illustrates two paths of firewall configurations and shows the communication paths to the server. Configuration X has a chain of two GateKeepers. Configuration Y has a single TCP firewall.

Figure 3.1 Communication paths to the server



To configure the server for the configurations shown in diagram above, enter the following information in the server's properties file:

1 Declare all firewall paths:

```
vbroker.se.iiop_tp.firewallPaths=x,y
```

2 Identify the components for each path:

```
vbroker.firewall-path.x=a,b
vbroker.firewall-path.y=c
```

The following sections describe how to specify the firewall components.

Specify the component of a proxy server

The following example shows an IIOP proxy across a firewall.

Server's Property:

```
vbroker.firewall-path.x=a,b
vbroker.firewall.a.type=PROXY
vbroker.firewall.a.ior=http://www.inprise.com/GK/GateKeeper.ior
vbroker.firewall.b.type=PROXY
vbroker.firewall.b.ior=IOR:<GateKeeper's stringified ior>
```

The first property defines the firewall components found in the path named `x`. The second and fourth properties specify the types of the component named `a` and `b`, respectively. Both component types are defined as `PROXY`, which identifies GateKeeper as an IIOP proxy server to forward all IIOP requests. The third property defines the IOR of *GateKeeper a* using URL naming. The fifth property defines the IOR of *GateKeeper b* using a stringified IOR.

Specify the component of a TCP firewall with NAT

Clients' messages may have to cross one or more TCP firewalls in order to reach the server. The TCP firewall components have to be defined when NAT is performed in the TCP firewall. If the TCP firewall does not perform NAT, the component can be ignored.

The following example shows how to use a router or firewall to forward an IIOP message at the TCP level.

Server's Property:

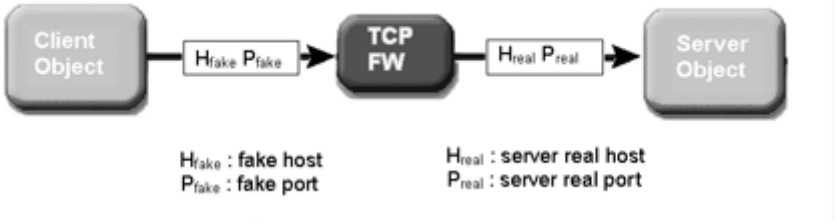
```
vbroker.firewall-path.y=c
vbroker.firewall.c.type=TCP
vbroker.firewall.c.host=<fake_host>
vbroker.firewall.c.iiop_port=<IIOP fake port>
vbroker.firewall.c.ssl_port=<SSL fake port>
vbroker.firewall.c.hiop_port=<HTTP fake port>
```

The first property defines the firewall components found in the path named `y`. The second property defines the type of component named `c` as `TCP`, which provides a predefined port to forward all IIOP, SSL and IIOP over HTTP messages on a router or other network device. The third property defines the fake host of the server. The

remaining last three properties define the fake port for the following message types: IIOP, SSL and HTTP.

The TCP firewall specified as component “c” in the above example is expected to perform host and port translation (NAT). The TCP firewall must be configured to translate the fake host to the server's real host and translate all the fake ports to the server's real listener ports.

Figure 3.2 TCP firewall with NAT



4

Advanced features

This section describes advanced features such as chaining GateKeepers, callbacks, access control, load balancing, fault tolerance and SSL. It also describes the factors that can improve the performance of GateKeeper.

Chaining of GateKeepers

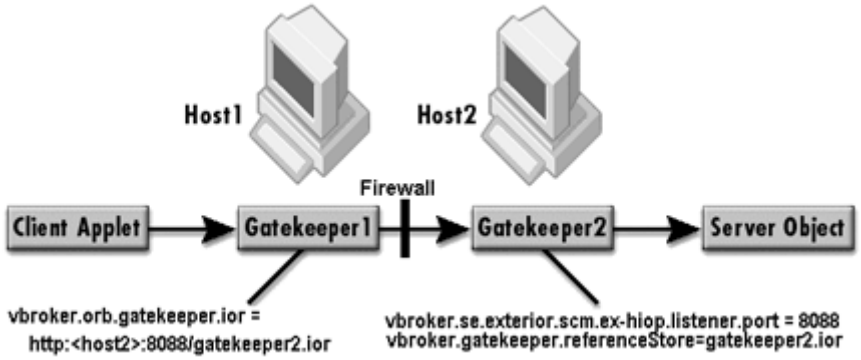
GateKeepers can be chained together to provide paths through the firewalls. There are two types of chaining

- static chaining
- dynamic chaining.

Static chaining of GateKeepers

In static chaining, the preceding GateKeeper is configured to forward messages to the next GateKeeper. The communication path is fixed and is therefore static.

Figure 4.1 Static chaining of GateKeepers



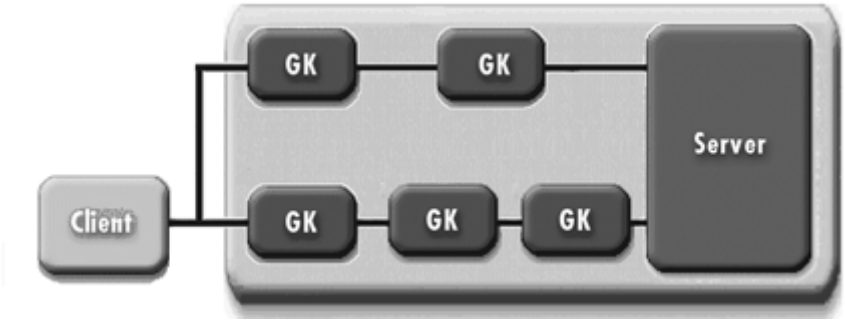
In the figure above, two chained GateKeepers are required to communicate across a firewall. The client applet sends messages to GateKeeper1 which will forward the messages across the firewall to GateKeeper2. GateKeeper1 is able to forward the message to GateKeeper2 because it has the Interoperable Object Reference (IOR) of

GateKeeper2. The IOR of GateKeeper2 specifies how to send messages from GateKeeper1 to GateKeeper2 and thus crosses the firewall.

Dynamic chaining of GateKeepers

In static chaining, the communication paths are specified in the GateKeeper's IOR. In dynamic chaining, the communication paths are specified in the server's IOR file. The client, if given the server's IOR file, can use the information in the server's IOR to select a path. The client tries the first path and the next if the first path fails, and so forth.

Figure 4.2 Dynamic chaining of GateKeepers



In the figure above, there are two paths from the client to the server. Both paths require chaining of GateKeepers. The two paths are specified in the server's IOR that the client reads, the first path is tried and on failure the second path is tried. The path is chosen dynamically at runtime.

The way to specify the paths to the server is described in “Specifying communication paths to the server”.

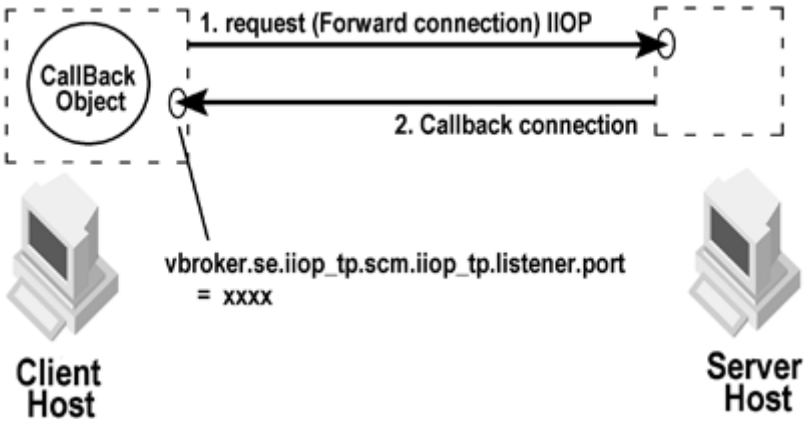
Callbacks

In most implementations, requests are initiated by the clients to which the servers reply back. There are also implementations where information must arrive at a client that is not in response to a request from the client which can be implemented by creation of callback objects. Callback objects can be implemented in the three methods described below.

Callbacks without GateKeeper

The implementation shown in the following figure is applicable to cases where the client and server can communicate in both directions. In these cases, there are either no intervening firewalls or the intervening firewalls do not hinder the communications between the clients and servers.

Figure 4.3 Using callbacks without GateKeeper

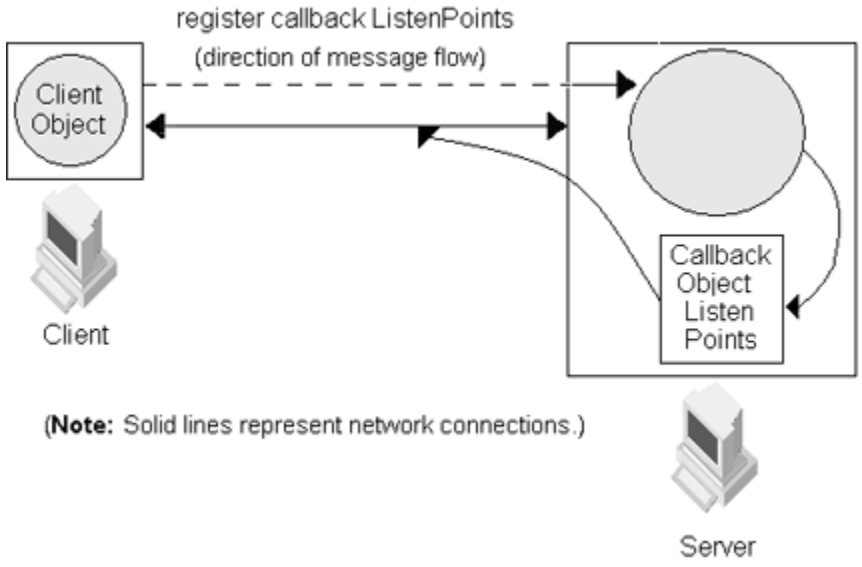


In the above example, the client creates an object, starts the listener, generates an IOR and sends a request and IOR to the server. The server calls the client's listeners and creates a callback connection. Subsequently, all messages to the callback objects will be channel through the callback connection.

Callbacks without GateKeeper using bidirectional GIOP

With bidirectional IIOP, servers use the client-initiated connections to transmit asynchronous information back to the clients. Servers need not initiate any connections to the client.

Figure 4.4 Using callbacks with bidirectional GIOP support without GateKeeper



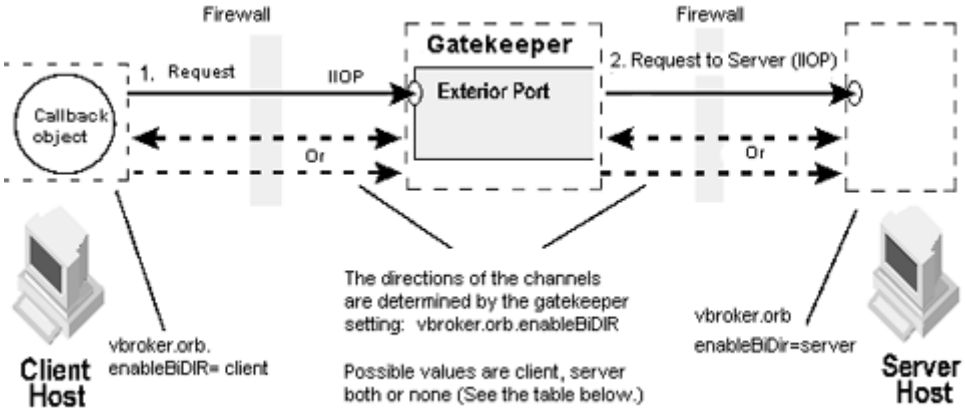
In the above figure, the client is able to establish a connection directly to the server, but the server is unable to establish a separate callback connection because of an intervening firewall. Therefore, the client and the server negotiate a bidirectional GIOP connection and share the initial connection established by the client for IIOP traffic in both directions.

The CORBA specification also adds a new policy to portably control this feature. For more information about bidirectional communications exclusive of GateKeeper, see the Borland AppServer *Developer's Guide*.

Callback with GateKeeper's bidirectional support

With bidirectional IIOP, servers use the client-initiated connections to transmit asynchronous information back to the clients. Servers need not initiate any connections to the client. The CORBA specification also adds a new policy to portably control this feature. For information about bidirectional communications exclusive of the GateKeeper, see the Borland AppServer *Developer's Guide* for more information.

Figure 4.5 Callback with GateKeeper's bidirectional support



In the figure above, GateKeeper sits between the client and server and therefore it acts as a server for the client and as a client for the server. The Client/GateKeeper and the GateKeeper/Server communication channels can be set to unidirectional or bidirectional connections.

You can also selectively set the channels to unidirectional or bidirectional. If the client defines `vbroker.orb.enableBiDir=client` and the server defines `vbroker.orb.enableBiDir=server`, the following table describes the type of channels for the different values of `vbroker.orb.enableBiDir` for GateKeeper.

vbroker.orb.enableBiDir=	Client GateKeeper	GateKeeper Server
client	unidirectional	bidirectional
server	bidirectional	unidirectional
both	bidirectional	bidirectional
none	unidirectional	unidirectional

Bidirectional connection example

An example that demonstrates GateKeeper's support for bidirectional connections is located in the `examples/vbroker/bank_bidir` subdirectory under the GateKeeper for Java installation.

The Bank BiDir example is similar to the Bank Callback example, except that in the BiDir example, bidirectional connections are established between the client, GateKeeper, and server. In other words, in the bidirectional implementation, the same connection is used for both forward invocations as well as callbacks.

This example demonstrates how to:

- Configure the client to enable bidirectional connections via the property file.
- Program the client to create callback objects that can be passed as arguments to invocations on server objects.
- Configure the server to set up firewall paths containing the server side inbound firewall via property file. It also demonstrates how to configure the server so that it can accept bidirectional connections.
- Program the server to export the firewall path in server objects IOR.
- Configure the GateKeeper so that it supports bidirectional connections.

The client

In this example, the client `Client.java`:

1. Creates a callback object on the POA named `callback_poa`. (This callback object will be invoked by the server through the GateKeeper.)
2. Binds to the **AccountManager** object.
3. Sends the object reference of this callback object to the server by opening a bank account by invoking `open()` and passing the callback object as an argument.
4. Queries the **Account** object reference obtained for the balance, again passing the callback object (this time it is passed to the balance method).

The server

In the example, the server `Server.java`:

1. Creates a persistent POA named `bank_poa` and a transient POA named `account_poa` with firewall policy value of `EXPORT`.
2. Creates an instance of the **AccountManager** servant.
3. Activates that servant on `bank_poa`.
4. Starts waiting for client requests.
5. Responds to the requests by invoking a method on the client-initiated callback object through the GateKeeper.

File name	Description
`server.properties`	Property file used to configure the bank server. In this example, the server is configured to accept listen points so that the connection between the GateKeeper and the server will be bidirectional. To make the connection unidirectional, either remove the property `vbroker.orb.enableBiDir` or set the value of this property to none. The other properties in this file are for loading the firewall package and then setting the firewall path so that server-side objects can be bound and called by the client.
`client.properties`	Property file used to configure the bank client. In this example, the client is configured to publish its listen points so that the connection between the client and the GateKeeper will be bidirectional. To make the connection unidirectional, either remove the `vbroker.orb.enableBiDir` property or set its value to none. As with the server, the vbroker.orb.dynamicLibs property is set to load in the necessary firewall library so that the client request can traverse the GateKeepers.
`GateKeeper.properties`	Property file used to configure the GateKeeper. In this example, the GateKeeper is configured to both publish the listen points and accept the listen points. Hence, both the client-GateKeeper and GateKeeper-server connections will be bidirectional. These connections can be converted into unidirectional by either removing the vbroker.orb.enableBiDir property or by setting this property to the value none.

Security considerations

Caution

Use of bidirectional IIOP may raise significant security issues. In the absence of other security mechanisms, a malicious client may claim that its connection is bidirectional for use with any host and port it chooses. In particular, a client may specify the host and port of security-sensitive objects not even resident on its host. In the absence of other security mechanisms, a server that has accepted an incoming connection has no way to discover the identity or verify the integrity of the client that initiated the connection. Further, the server might gain access to other objects accessible through the bidirectional connection. This is why the use of a separate, bidirectional SCM for callback objects is recommended. If there are any doubts as to the integrity of the client, it is recommended that bidirectional IIOP not be used.

Java

For security reasons, a server running VisiBroker for Java will not use bidirectional IIOP unless explicitly configured to do so. The property `vbroker.se.<sename>.scm.<scmname>.manager.importBiDir` gives you control of bidirectionality on a per-SCM basis. For example, you might choose to enable bidirectional IIOP only on a server engine that uses SSL to authenticate the client, and to *not* make other, regular IIOP connections available for bidirectional use. (See the Properties section for more information about how to do this.) In addition, on the client-side, you might want to enable bidirectional connections only to those servers that do callbacks outside of the client firewall. To establish a high degree of security between the client and server, you should use SSL with mutual authentication (set `vbroker.security.peerAuthenticationMode` to `REQUIRE_AND_TRUST` on both the client and server).

Access control

GateKeeper has a rules-based access controller built into it. This controller can deny or grant accessibility based on:

- operation
- signed by
- server's host/port
- server's subnet
- client's host/port
- client's subnet

All rules are evaluated in the order in which you specify them. Action is taken based on the first matched rule. If there is no matched rule, the default action you specify is taken. See “GateKeeper properties” for the syntax of the rules.

Custom-designed access control in GateKeeper

GateKeeper lets you plug-in custom designed Access Control mechanisms. The Access Control Manager invokes all Access Controllers specified using GateKeeper properties. The Access Control Manager uses the following interface for implementation of an Access Controller:

```
package com.inprise.vbroker.gatekeeper.security;
public interface AccessController {
  public void init(org.omg.CORBA.ORB orb, String prefix);
}
```

Access Controllers use the `TcpConnectionInfo` interface to get more information about the Client:

```
package com.inprise.vbroker.orb;
public interface TcpConnectionInfo {
  public String getLocalHostName();
  public int    getLocalPortNumber();
  public String getHostName();
  public int    getPortNumber();
  public long   getTotalBytesRead();
  public long   getTotalBytesWrote();
  public String name();
  public java.io.InputStream getInputStream();
  public java.io.OutputStream getOutputStream();
}
```

The Access Control Manager calls the `init` method to initialize the Access Controller. GateKeeper supports the following types of Access Controller interfaces:

- **ObjectAccessController:** The `isObjectAccessible()` method is invoked when the client requests GateKeeper to set up a proxy channel (communication path) to the server object. It should return `true` if the object is accessible:

```
package com.inprise.vbroker.gatekeeper.security;
import  com.inprise.vbroker.orb.TcpConnectionInfo;
import  com.inprise.vbroker.IOP.ServiceContext;
public interface ObjectAccessController extends AccessController {
  public boolean  isObjectAccessible(
    TcpConnectionInfo     clientInfo,
    org.omg.CORBA.Object  server,
    ServiceContext[]      contexts,
    byte[]                principal);
}
```

- **OperationAccessController:** The `isOperationAccessible()` method is invoked when the client sends requests through the GateKeeper. It should return `true` if a given operation is accessible:

```
package com.inprise.vbroker.gatekeeper.security;
import  com.inprise.vbroker.orb.TcpConnectionInfo;
import  com.inprise.vbroker.IOP.ServiceContext;
public interface OperationAccessController extends AccessController{
  public boolean isOperationAccessible(
    TcpConnectionInfo         clientInfo,
    TcpConnectionInfo         serverInfo,
    org.omg.CORBA.Object      server,
    String                    operation,
    ServiceContext[]          services);}
```

You can program an access controller (for example, `myAC`) and install it on GateKeeper using following properties:

```
vbroker.gatekeeper.security.accessControllers=myAC

vbroker.gatekeeper.security.acl.myAC.type=com.inprise.vbroker.gatekeeper.securi
ty.myACImpl
  vbroker.gatekeeper.security.acl.myAC.rules=
  vbroker.gatekeeper.security.acl.myAC.default=grant
```

An Access Controller can be implemented as follows:

```
package com.inprise.vbroker.gatekeeper.security;
import java.util.*;
import java.io.*;
import com.inprise.vbroker.orb.TcpConnectionInfo;
import com.inprise.vbroker.orb.ORB;
import com.inprise.vbroker.IOP.ServiceContext;
public class myACImpl implements
ObjectAccessController, OperationAccessController {
  public void init(org.omg.CORBA.ORB orb, String prefix) {
  }
  public boolean isObjectAccessible(
TcpConnectionInfo clt, org.omg.CORBA.Object svr,
ServiceContext[] contexts, byte[] principal) {
    return true;
  }
  public boolean isOperationAccessible(
TcpConnectionInfo clt, TcpConnectionInfo svr,
org.omg.CORBA.Object server, String operation,
ServiceContext[] services) {
    return true;
  }
}
```

The access control methods or rules can be defined by the implementation.

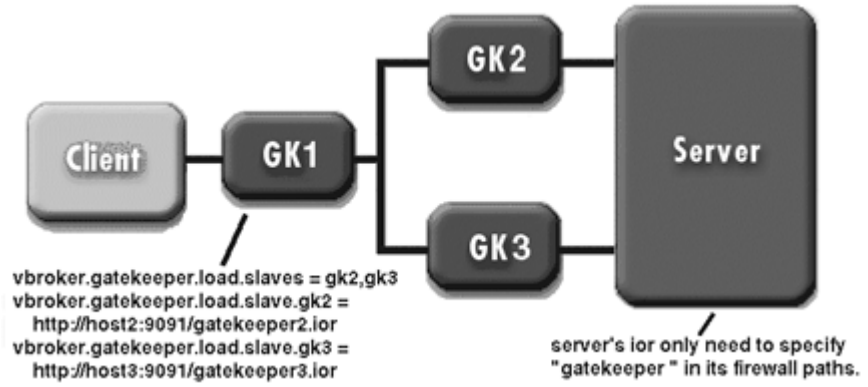
Load balancing and fault tolerance

GateKeeper is most often used to provide a single point of access to the internal network therefore it can become congested or become the single point of failure. These problems can be resolved by clustering GateKeepers to provide a degree of fault tolerance and scalability.

Load balancing

A master GateKeeper and one or more slave GateKeepers can be clustered together. The master GateKeeper is responsible for balancing the load among the slave GateKeepers. The server should export the master GateKeeper object reference only.

Figure 4.6 Load balancing using GateKeeper



The figure above shows the property setting for GateKeeper1 and the server. The master GateKeeper can balance the load between a slave GateKeepers on a per object level. On the object level, each client will be redirected to one of the slave GateKeepers based on the load balancing policy. In general, this will balance the load more evenly but potentially use more resources and slower.

The default load balance policy is round-robin. This policy, however, can be customized and is available as a standard package. Please contact Borland for more information.

Additionally, GK2 and GK3 can also have their own slave GateKeepers. In this configuration, a hierarchy of master and slaves can be stacked over one another.

Custom-designed load balancing in GateKeeper

The ORB default implementation of load distribution uses the **round-robin algorithm** in which the client request is shared among a server and GateKeeper in a sequential order. The following code example shows a distributor implementation:

```
package com.inprise.vbroker.gatekeeper.ext;
import java.util.Enumeration;
import org.omg.Firewall.GIOPProxy;
import com.inprise.vbroker.orb.*;
import com.inprise.vbroker.util.*;

public class MyDistributor implements Distributor {
       private Enumeration _enum;
       private UnGuardedVector _servers;
       public void init(ORB orb, UnGuardedVector v) {
              _servers = v;
```

```
                _enum = servers.elements();
        }
        public synchronized GIOPProxy next() {
        if (!_enum.hasMoreElements()) {
        _enum = _servers.elements();
        }
        return (GIOPProxy)_enum.nextElement();
        }
}
```

The Server Manager can collect current load related information of other GateKeeper instances in a master/slave configuration. Based on the available real-time information from the Server Managers, the master GateKeeper can reallocate client requests to other GateKeepers. In another scenario, a federation of GateKeepers can exchange load-statistics to distribute the load.

Fault tolerance

A master GateKeeper and one or more backup GateKeepers can be clustered together to be viewed as a single GateKeeper by the client. There are several ways to cluster the GateKeepers:

- Cluster the GateKeepers as different firewall paths to the server.

  This configuration is accomplished similarly to dynamic chaining of GateKeeper. It requires no changes to the GateKeeper configuration; you only need to configure the server to include all the backup GateKeepers as a firewall path on the server listener. This approach, however, makes the server configuration more complex.

- Fold all the backup GateKeeper's object references (profiles) into the master GateKeeper's object reference. When the master GateKeeper fails, the client would rebind to one of the other backup GateKeepers automatically. This approach can make the GateKeeper's object reference very large. The load balancing feature of GateKeeper follows this approach.

Scalability and performance guidelines

When assessing GateKeeper performance, it is useful to compare a GateKeeper scenario (Client-GateKeeper-Server) to a direct scenario (Client-Server).

Note

Here performance is represented as response time and scalability is represented as throughput.

The GateKeeper scenario requires two connections and thus two invocations. As a result:

- **Throughput is reduced:** It may be reduced by as much as 50 percent when compared to the direct scenario.
- **Response Time is slowed:** Response time will take longer when compared to the direct scenario. In some cases, it may take up to 200 percent longer.

GateKeeper performance tuning

GateKeeper does not introduce any new performance threshold or throughput threshold which means that GateKeeper will have the same performance and throughput profile as the VisiBroker ORB. Because GateKeeper is a CORBA application, it inherits the basic features of the ORB. As such, all ORB specific performance tuning parameters apply to GateKeeper as well. The following areas described below, however, can affect the performance of GateKeeper:

Bidding mechanism

The client-side ORB can be programmed to select specific bids based on the constraints set by the user. The order of selection of bids can be specified to speed up the process of connection establishment:

- **Constraint on Bid-Portfolio:** The following properties are useful for setting exclusive bids in the case of static chaining of GateKeeper:

```
vbroker.orb.alwaysProxy
vbroker.orb.alwaysTunnel
vbroker.orb.alwaysSecured
```

  For example, the vbroker.orb.alwaysProxy is useful when a specific GateKeeper is statically chained to another GateKeeper. If one is very sure that only HTTP Tunnelling will be used while chaining the GateKeepers, then set vbroker.orb.alwaysTunnel property can be set to avoid unnecessary bids. When the vbroker.orb.alwaysSecured property is set, then the GateKeeper will use secure communication path only while chaining. Please note that these properties are set on the outer GateKeeper.

- **Order of Bid-Selection:** The order of the bid can affect the speed of the selection of a specific bid. For example, if you are certain that most of the connections allowed on a specific GateKeeper will be of a secure type, you can place the SSL as the first entry in the string as follow:

```
vbroker.orb.bidOrder=inprocess:liop:ssl:iiop:proxy:hiop:locator
```

- **Specifying high-precedence Bid:** You can set the following property to the highest precedence bid. By default, it is set to `inprocess` in the ORB:

```
vbroker.orb.bids.critical=inprocess
```

Cache management

The following property sets the cache size of GateKeeper:

```
vbroker.gatekeeper.cache.size=100
```

Message marshalling

By setting the chunk size of the stream, you can increase the size of messages exchanged between GateKeeper and the client/server application. The chunk size can have significant impact on the performance of the applications, particularly using HTTP Tunnelling:

```
vbroker.orb.streamChunkSize=4096
```

You can try using values such as: 4096, 8192, or 16384. The performance of the applications may vary depending on the maximum size of the packets on your network.

Thread management

Depending on the response needs of GateKeeper, different techniques of thread management can be applied, such as thread pooling, thread-per-session, and so forth. By default, the request forwarding IIOP service uses `ThreadPool`, and the HIOP service uses `ThreadSession`:

```
vbroker.se.exterior.scm.ex-iiop.dispatcher.type=ThreadPool
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMax=100
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMin=0
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMaxIdle=300

vbroker.se.interior.scm.in-iiop.dispatcher.type=ThreadPool
vbroker.se.interior.scm.in-iiop.dispatcher.threadMax=100
vbroker.se.interior.scm.in-iiop.dispatcher.threadMin=0
vbroker.se.interior.scm.in-iiop.dispatcher.threadMaxIdle=300
```

```
vbroker.se.exterior.scm.ex-hiop.dispatcher.type=ThreadSession
```

Connection management

The bidirectional GIOP has advantages of using the same communication path for forward and backward communication. Therefore, Borland recommends that you use the `vbroker.orb.enableBiDir` property setting in callback scenarios. The following properties let you optimize connection resource usage (see Appendix A for more details):

```
vbroker.se.exterior.scm.ex-iiop.manager.connectionMax
vbroker.se.exterior.scm.ex-iiop.manager.connectionMaxIdle

vbroker.se.interior.scm.in-iiop.manager.connectionMax
vbroker.se.interior.scm.in-iiop.manager.connectionMaxIdle
```

`vbroker.ce.iiop.ccm.connectionMax` should not be used in the context of GateKeeper, because GateKeeper should be allowed to connect to as many servers as needed as it is an intermediate service to potentially many clients. GateKeeper must not stop already connected clients from proceeding with connections to servers just because the number of outgoing connections it can open is limited. Instead, GateKeeper can restrict the number of clients it is willing to service using the following property:

```
vbroker.se.exterior.scm.ex-iiop.manager.connectionMax
```

`vbroker.ce.iiop.ccm.connectionMaxIdle`, however, can be used to drop idle connections to servers. This is particularly useful when the number of servers the GateKeeper would potentially connect to is large, the number of connecting clients is small, and the clients mainly target only a few servers.

Impact of asynchronized invocation of GateKeeper

Asynchronized invocation of GateKeeper does not have a very significant impact on performance and scalability.

GateKeeper performance properties

There are many properties that affect GateKeeper's performance. Those properties related to connection, thread type, mode of operation and call type are described here.

For more information, see “Performance and load balancing” for additional properties that can be adjusted for better performance.

Connection settings

Connection related properties of GateKeeper are:

```
vbroker.se.<xxx>.scm.<yyy>.manager.connectionMax
vbroker.se.<xxx>.scm.<yyy>.manager.connectionMaxIdle
```

where <xxx> and <yyy> represent “exterior, ex-hiop”, “exterior, ex-iiop”, “exterior, ex-hiops”, “exterior, ex-ssl”, “interior, in-iiop” or “interior, in-ssl”.

The first property specifies the maximum number of active connections allowed. Limiting connections conserves GateKeeper resources, but may decrease client performance. The default is no limit.

The second property specifies how long an inactive connection is idle before it is closed. The default is 0 which means that inactive connections are never closed.

Thread related settings

When the dispatcher type is “ThreadPool”, the following properties of GateKeeper can be tuned:

```
vbroker.se.<xxx>.scm.<yyy>.dispatcher.threadMin
vbroker.se.<xxx>.scm.<yyy>.dispatcher.threadMax
vbroker.se.<xxx>.scm.<yyy>.dispatcher.threadMaxIdle
```

where <xxx> and <yyy> pair is “exterior, ex-iiop”, “exterior, ex-ssl”, “interior, in-iiop” or “interior, in-ssl”.

The first property “threadMin” specifies how many threads are pre-created so that requests can be quickly serviced. The default is 0.

The second property “threadMax” specifies the maximum number of threads that can be created so that the system cannot be overloaded with too many threads. The default is 100. Any request that cannot be serviced because of too few threads will wait for the next available thread.

The third property “threadMaxIdle” specifies how long (in seconds) a thread is idle before it is destroyed. The default is 300 seconds.

GateKeeper modes

GateKeeper can run in normal and pass-through mode. The pass-through mode has lower performance because the content of the packets is not examined by GateKeeper but still consume GateKeeper's resources. In fact, each pass-through connection needs exclusive ports throughout the life-span of the connections. A client process can request exclusive connection using policies programmatically.

GateKeeper in its normal mode of operation gives the better performance.

The mode is normal unless the pass-through is enabled by setting the property vbroker.gatekeeper.enablePassthru=true.

Call types

There are three types of calls:

- normal forward calls
- bidirectional callbacks
- VisiBroker 3.x style callbacks

Bidirectional callbacks use a single connection for both forward calls and callbacks. It is more efficient than the VisiBroker 3.x style callbacks.

Bidirectional callbacks are as efficient as the normal forward calls.

GateKeeper and SSL

Note

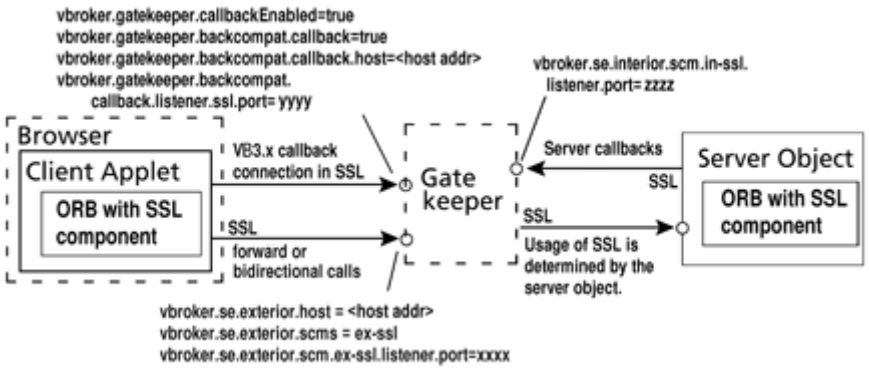
SSL is a separate optional package; therefore for applets and server objects to run in SSL mode, you must have a SSL component included in their ORB runtime.

GateKeeper with SSL provides the following security features:

- relay IIOP/SSL connections between client and server
- support HTTPS tunneling
- enable IIOP/SSL callback (VisiBroker 3.x Style and bidirectional)
- perform authentication on behalf of server
- forward credentials

Additional properties for setting SSL can also be found in "GateKeeper properties."

Figure 4.7 SSL connections to GateKeeper



SSL connections to GateKeeper

The server determines if the connection uses SSL or a regular IIOP connection. The client running in SSL mode may request the connection to be SSL. The server running in SSL mode, however, requires the client to connect to it in SSL mode.

If the client sets `vbroker.orb.alwaysSecure=true` in its property file, it will always connect to the Server or GateKeeper in SSL mode and will not first try other types of connections (which may fail if the Server or GateKeeper does not accept other types of connection). This shortens the time for connections.

Similarly, setting the same property will help GateKeeper when it connects to the Server.

SSL for forward and bidirectional calls

You can set the following GateKeeper properties to enable SSL for calls from the client (applet) to the server object (via the GateKeeper):

```
vbroker.se.exterior.host = <host address>
vbroker.se.exterior.scms = ex-iiop, ex-hiops, ex-ssl
vbroker.se.exterior.scm.ex-ssl.listener.port = <port address>
```

The applet client opens an SSL connection to the GateKeeper. The client-GateKeeper communication channel is in SSL mode. However, the mode of the GateKeeper-server communication channel is determined by the server. If the server's scm is set to SSL mode, the GateKeeper-server communication channel will be in SSL mode.

Bidirectional calls use the same forward communication paths. However, there is additional property setting for bidirectional callbacks.

Enabling the Security Service in GateKeeper

While security is turned on by default, this feature applies to the licensing of the Security Service only. There are, however, no license checks to turn on the Security Service.

In Borland VisiBroker, the security is turned off by default as specified in the following property:

```
vbroker.security.disable=true
```

By setting the following property in VisiBroker, the application will prompt for username and password for authentication:

```
vbroker.security.login=true
```

You must create `*.config` files (examples shown below) to specify the authentication and realm related parameters.

As a generic example of a security enabled GateKeeper, the IIOP, IIOP/SSL, HIOP, and HIOPS listeners have been enabled in the following set of properties:

gatekeeper.config

```
  System
  com.borland.security.provider.authn.HostLoginModule required REALM=myrealm
PRIMARYIDENTITY=true;

  com.borland.security.provider.authn.ClientSideDataCollection required
REALM=testrealm;

  };
  myrealm
  com.borland.security.provider.authn.HostLoginModule required;
  };

  anotherrealm {
  com.borland.security.provider.authn.HostLoginModule required;
  };
```

gatekeeper.properties

```
vbroker.security.disable=false
vbroker.security.peerAuthenticationMode=none
vbroker.security.secureTransport=false
vbroker.security.trustpointsRepository=Directory:./trustpoints
vbroker.gatekeeper.referenceStore=./gkclnt.ior
vbroker.orb.enableBiDir=both
vbroker.orb.dynamicLibs=com.borland.security.hiops.Init

  vbroker.se.exterior.scms=ex-iiop,ex-hiop,ex-ssl,ex-hiops
  vbroker.se.exterior.host=143.186.142.21
  vbroker.se.exterior.scm.ex-iiop.listener.port=25000
  vbroker.se.exterior.scm.ex-hiop.listener.port=25001
  vbroker.se.iiop_tp.scm.hiop_ts.listener.port=25002
  vbroker.se.exterior.scm.ex-ssl.listener.port=25003
  vbroker.se.exterior.scm.ex-hiops.listener.port=25004

  vbroker.se.interior.scms=in-iiop,in-hiop,in-ssl
  vbroker.se.interior.host=143.186.139.226
  vbroker.se.interior.scm.in-iiop.listener.port=15001

  vbroker.se.interior.scm.in-hiop.listener.port=15002

  vbroker.se.interior.scm.in-ssl.listener.port=15003

  # Enable callback using this GateKeeper

  vbroker.gatekeeper.callbackEnabled=true

  # Enable VBJ3.x (old style) callback also
  vbroker.gatekeeper.backcompat.callback=true
  vbroker.gatekeeper.backcompat.callback.host=143.186.142.21
  vbroker.gatekeeper.backcompat.callback.listeners=iiop,ssl
  vbroker.gatekeeper.backcompat.callback.listener.iiop.port=16001
  vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
  vbroker.gatekeeper.backcompat.callback.listener.ssl.port=16002
```

```
  vbroker.gatekeeper.backcompat.callback.listener.ssl.proxyPort=0
  vbroker.gatekeeper.backcompat.callback.listener.ssl.type=SSLCallback

  # Optional: enable GateKeeper specific Access Control properties
  vbroker.gatekeeper.security.accessControllers=myAC
  vbroker.gatekeeper.security.acl.myAC.default=grant
  vbroker.gatekeeper.security.acl.myAC.rules=rule1
  vbroker.gatekeeper.security.acl.myAC.rule1=grant [operation="*"]

  # Optional: Identity of GateKeeper

  vbroker.security.wallet.identity=<username>
  vbroker.security.wallet.password=<password>
  vbroker.security.wallet.type=Directory:<path-to-identities>
```

The property settings in the following example tell the client to specifically request a secure transport using GateKeeper. The client application collects the username and password and sends this conformation to the server via GateKeeper.

client.config

```
  System {
  com.borland.security.provider.authn.ClientSideDataCollection required
REALM=myrealm;
  };

  Client {
  com.borland.security.provider.authn.ClientSideDataCollection required;
  };
```

client.properties

```
  vbroker.security.disable=false
  vbroker.security.login=true

vbroker.security.authentication.callbackHandler=com.borland.security.provider.a
uthn.HostCallbackHandler
  vbroker.security.authentication.config=client.config

  vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
  vbroker.se.iiop_tp.scms=iiop_tp,ssl
  vbroker.orb.alwaysProxy=true
  vbroker.orb.alwaysSecure=true
```

The property settings in the following example disables the IIOP listener and the server is assumed to be a secure application that uses SSL transport only:

server.config

```
  System {
  com.borland.security.provider.authn.HostLoginModule required REALM=myrealm
PRIMARYIDEHostITY=true;
  com.borland.security.provider.authn.ClientSideDataCollection required
REALM=testrealm;
  };
myrealm {
  com.borland.security.provider.authn.HostLoginModule required;
  };
anotherrealm {
  com.borland.security.provider.authn.HostLoginModule required;
  };
```

server.properties

```
vbroker.security.disable=false
vbroker.security.login=true
vbroker.security.authentication.callbackHandler=com.borland.security.provider.a
uthn.HostCallbackHandler
vbroker.security.authentication.config=server.config
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.host=143.186.142.21
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=25000
vbroker.se.iiop_tp.scm.ssl.listener.port=25005
vbroker.se.iiop_tp.firewallPaths=intranet
vbroker.firewall-path.intranet=first,second
vbroker.firewall-path.internet=first
vbroker.firewall.first.type=PROXY
vbroker.firewall.first.ior=http://localhost:16085/gatekeeper.ior
vbroker.firewall.second.type=TCP
vbroker.firewall.second.host=192.75.11.14
vbroker.firewall.second.iiop_port=32000
vbroker.firewall.second.hiop_port=32001
vbroker.firewall.second.ssl_port=32005
```

Enabling access to the Naming Service through GateKeeper

To start the Naming Service on a fixed IP address and port, you must set the following properties. In the following example, the Naming Service is running on the IP host address: 143.186.142.21 and listener port: 32101:

namingservice.properties

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
vbroker.orb.logger.output=ns_debug.log

vbroker.naming.logLevel=7
vbroker.naming.iorFile=ns.ior

vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.host=143.186.142.21
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=32010
```

gatekeeper.properties

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
vbroker.agent.enableLocator=false

vbroker.orb.initRef=NameService=corbaloc::143.186.142.21:32010/NameService
vbroker.gatekeeper.referenceStore=gkclnt.ior
vbroker.se.exterior.host=143.186.142.21
vbroker.se.interior.host=143.186.139.226
vbroker.se.exterior.scm.ex-iiop.listener.port=25000
vbroker.se.exterior.scm.ex-hiop.listener.port=25001
vbroker.se.iiop_tp.scm.hiop_ts.listener.port=25002
```

5

Troubleshooting GateKeeper

This section describes how to obtain debugging information from GateKeeper and its clients and servers. It also highlights potential problems such as incorrect environment and registry settings and describes some common tools useful for troubleshooting GateKeeper.

Preparation for troubleshooting

The following sections describe the preparations that must be done or observed before troubleshooting GateKeeper.

Getting debugging information

Comprehensive debugging information can be obtained by setting the properties of the client, server and GateKeeper. The following table shows the relevant settings and whether they are applicable to the client, server, or GateKeeper. The properties must be set in their respective properties file.

Log levels have numeric levels along with corresponding text values and either can be used to describe the situation. Levels 4 and higher are useful for debugging. The following table describes the log level values.

Log level value	Description
`0 or EMER`	System is unusable. A panic condition.
`1 or ALERT`	A condition that should be corrected immediately, such as a corrupted system database.
`2 or CRIT`	Critical conditions, such as hard device errors.
`3 or ERR`	Error conditions.
`4 or WARNING`	Warning conditions such as connection failures due to authentication problems, such as incorrect password and authorization failures. This is the default.
`5 or NOTICE`	Conditions that are not critical, but may require special handling configuration settings.
`6 or INFO`	Informational such as all user accesses to specific methods requested to the server.
`7 or DEBUG`	Debug information meant to be understood only by the developer. These messages are not internationalized

The following table describes the property settings useful for debugging GateKeeper.

Property	Description	Set in Property File
`vbroker.orb.logger.output=<filename>`	**Log file** Specifies the name of the file where the log is recorded. If not specified, the default is `gkdebugfile.log`. Log information can also be sent to `stdout`.	Server, client and GateKeeper
`vbroker.orb.bufferDebug=true`	**ORB** Forces the internal buffer manager to display the buffers used by ORB.	Server, client and GateKeeper
`vbroker.orb.debug=true`	**ORB** Displays debugging from the ORB	Server, client and GateKeeper
`vbroker.orb.warn=<warning level>`	**ORB** Specifies which warning of a particular level to be displayed by the ORB. The values can be 0, 1 or 2. Level 2 will display warnings of all levels.	Server, client and GateKeeper
`vbroker.orb.logLevel=<loglevel>`	**ORB** The log level can take one of the values described above.	Server, client and GateKeeper
`vbroker.orb.logger.appName=<Application Name>`	**ORB** Specifies the application name to be displayed in the log.	Server, client and GateKeeper
`vbroker.events.debug=true`	**Event Service** Displays Event Service diagnostic messages.	Event Service
`vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.trace.Init` `vbroker.gatekeeper.trace.demo=true`	**GateKeeper** Displays trace information from GateKeeper's built-in trace facility.	GateKeeper
`vbroker.agent.debug=true`	**GateKeeper and Smart Agent** Displays debugging information of interactions between GateKeeper and the Smart Agent.	GateKeeper
`vbroker.locationservice.debug=true`	**Location Service** Displays debugging information of the Location Service.	Server, client and GateKeeper
`vbroker.URLNaming.debug=true`	**URLNaming** Displays debugging information of the URLNaming service loaded in the ORB runtime. This setting is often used to detect if the correct IOR is retrieved.	Server, client and GateKeeper
`vbroker.poa.logLevel=emerg`	**POA** Displays debugging information from the POA. GateKeeper has exterior, interior and `iiop_tp` POA.	Server and Server Side of GateKeeper

Property	Description	Set in Property File
vbroker.gatekeeper.passthru.logLevel=<emerg>	**Pass-through** Displays debugging information of pass-through mode in GateKeeper.	GateKeeper
vbroker.security.logLevel=<logLevel>	**Security Service** Displays logging information of the security service such as SSL on GateKeeper.	GateKeeper

Starting GateKeeper in debugging mode

In addition to the properties described above, the `gatekeeper` and `vbj` command line utilities can output additional environment and parameter setting information at start-up. The `-VBJdebug` option produces this additional output. The following table shows examples of the debugging commands:

Component	Command line options example
Server	`vbj -VBJdebug -DORBpropStorage=server.prop Server`
GateKeeper	`gatekeeper -VBJdebug -J-Dvbroker.orb.debug=true -J-Dvbroker.orb.logLevel=7 -props gk.prop`
Client	`vbj -VBJdebug -DORBpropStorage=client.prop Client`

Note

The `-VBJdebug` option affects only the `gatekeeper` and `vbj` commands and has no relationship to the diagnostic property settings described above. The diagnostic properties will produce the same output regardless of whether or not the -`VBJdebug` option is used.

Environment settings

GateKeeper reads in the environmental variables at startup. On Windows, GateKeeper also reads settings in the registry. The precedence of the settings (UNIX and Windows) is as follows:

1. command line
2. properties file
3. environment settings
4. system default (Windows only)

The following table lists the common environment variables used by GateKeeper.

Environment Variable	Description
`CLASSPATH`	`CLASSPATH` should include the directories of the Java Development Kit and Java Servlet Development Kit. Specifically, `CLASSPATH` must include `servlet.jar`. For example, in Windows NT, at the DOS prompt enter: `set CLASSPATH=C:\visibroker\lib\tomcat\common\servlet.jar;c:\bes\jdk\jdk1.4.1\lib` If the classpath includes multiple versions of the JDK, compatibility issues may arise.
`JAVA_HOME`	Specifies the home directory of Java. If the Java directory is not defined in the CLASSPATH environment variable, the system will try to locate Java libraries from this directory. **Note:** On UNIX systems, this variable must be set. On Windows systems, it may be preset in the registry.

Environment Variable	Description
JDK_HOME	Specifies the home directory of JDK. If the JDK directory is not defined in the CLASSPATH environment, the system will try to locate JDK libraries from this directory. **Note:** On UNIX systems, this variable must be set. On Windows systems, it may be preset in the registry.
PATH	The PATH environment is set automatically during VisiBroker installation and should include the directory in which GateKeeper exists. For example, on Windows, at the command prompt enter: `set PATH = c:\bes\vbroker\bin`
VBROKERDIR	Specifies the home directory of the GateKeeper distribution. Note that VBROKERDIR points to the VisiBroker directory, but not the VisiBroker bin directory. Ensure that the PATH environment variable includes %VBROKERDIR%\bin.
BES_HOME	Obsolete (replaced by VBROKERDIR)
OSAGENT_PORT	Specifies the port used to contact the Smart Agent. Ensure the appropriate Smart Agent listens to this port and GateKeeper is able to reach this port (and the host).
BES_LIC_DIR	Specifies the path of the directory location in which the license data file exists.

Tools for troubleshooting

The following table describes some tools that are useful for troubleshooting GateKeeper.

Names	Description
osfind Windows and UNIX)	Included with the VisiBroker distribution. It is used to locate all objects registered within a given Smart Agent domain and to display information known to the agent. Normally, Smart Agents are restricted to a subnet only.
printior (Windows and UNIX)	Included with the VisiBroker distribution. It is used to print all the information encoded in an IOR into a human readable format..
ping (Windows and UNIX)	Part of TCP/IP networking tool set and usually included with the operating system. It is used to test where a packet is bounced back from a remote host to the current host and can be useful for firewall configuration verification.
tracert (Windows) traceroute (UNIX)	Part of TCP/IP networking tool set and distributed with some operating systems. It prints the route or path taken by a data packet to reach the destination host. It can help you identify a problem with firewalls as it shows where packets fail when being forwarded by routers.
route (Windows and UNIX)	Part of TCP/IP networking tool set and distributed with some operating systems. It prints the routing table and can be useful with firewall configurations.
netstat (Windows and UNIX)	Part of TCP/IP networking tool set and distributed with some operating systems. It displays protocol statistics and current TCP/IP network connections and can be useful to verify the state of connections and port availability.
nslookup (Windows and UNIX)	Part of TCP/IP networking tool set and distributed with some operating systems. It queries Internet domain name servers for hostname mapping.
vregedit (Windows)	Provided by Borland to edit Windows registry entries related specifically to Borland products.
regedit (Windows)	Part of Microsoft Windows operating system distribution and lets you to edit the Windows registry.

Getting information about the computer network

A good understanding of the computer network is needed to configure GateKeeper properly. You should work closely with the network administrators to identify problems that might arise from an improper configuration of GateKeeper and the firewall or the network itself. Many times configuration problems arise due to an incorrect configuration of the router or firewalls.

Firstly, you should try to understand the network diagram, firewall policies, routing tables, packet filters, and the location and configuration of basic TCP/IP stack servers. Most network administrators can provide you with logical network diagrams that show the physical wiring and the components in their network. When making deployment plans for GateKeeper , it recommended that you first start by analyzing and understanding these diagrams..

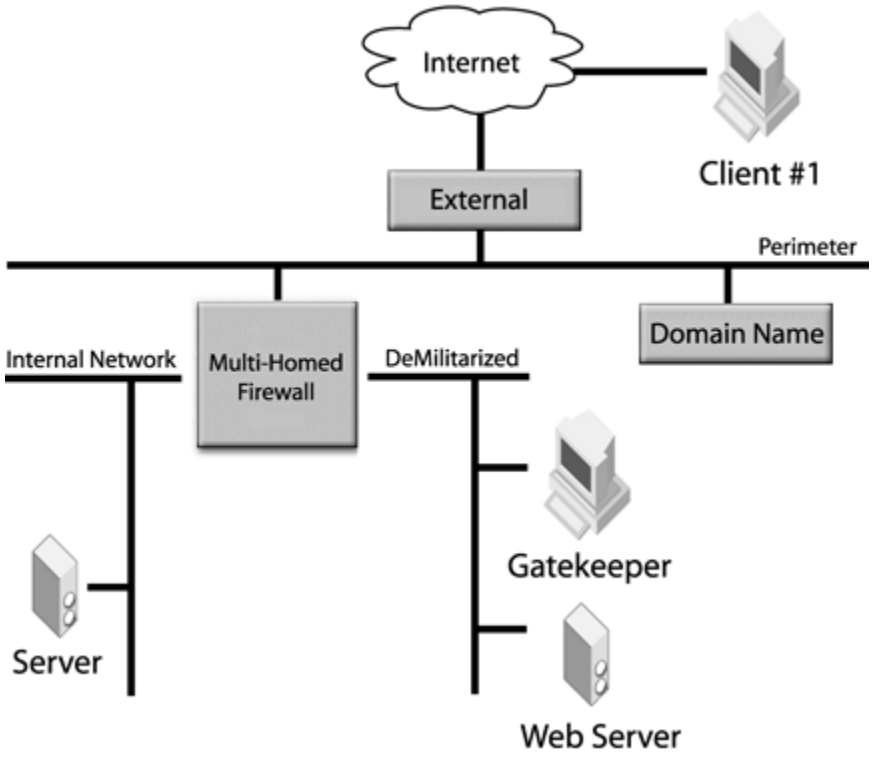
Next, you need to understand the firewall policies in place. Understanding the firewall policies and the physical network diagrams will help you determine whether messages from the client application are allowed to pass through various hops of the networks to reach the server and vice versa. This information in turn determines where the you should deploy GateKeeper and it will save a considerable amount of time when troubleshooting GateKeeper's configuration.

An external router routes packets to/from the Internet and perimeter networks. Additionally, the external router can be programmed so that only a restricted set of protocols can enter from the Internet to the perimeter network. This additional information is only available in the firewall policy. If the routes are not configured properly, the packets will be forwarded to the wrong destination or will be ignored. Whenever there is any change in the routing table or firewall policies, the network administrator should notify you.

A multi-homed firewall can filter and route packets from the perimeter network into the internal network and the de-militarized zone. Additionally, it may also perform Network Address Translation in which the real IP address of the internal network is replaced with the fake IP address and vice versa.

The following figure is an example of a network diagram that shows the physical wiring layout of three subnets; the Perimeter Internet, Demilitarized zone, and internal network.

Figure 5.1 Typical network diagram example



- The External Router routes packets to/from the Internet and Perimeter Internet. Additionally, the external router can be programmed such that only a small restricted set of protocols enter from Internet to the Perimeter Internet. This additional information is only available in the Firewall policy; that is the routing table.
- The Multi-Homed Firewall filters packets and routes packets from the Perimeter Internet into the two subnets; internal Network and Demilitarized Zone. Additionally, it may also perform Network Address Translation in which the real IP address of the internal network is replaced with the fake IP address and vice versa.

Note

The example above illustrates one of many possible network configurations and, therefore, it is very important to know where such information can be obtained before deploying GateKeeper.

Essential checks

GateKeeper acts like a proxy and problems can arise in the client, GateKeeper, or the server. The following sections describe some essential checks you can make when GateKeeper fails to work properly. The checks described below are not meant to be exhaustive and are not arranged in order of importance or performance sequence, but are provided here to serve as a guideline for preliminary troubleshooting.

Check the Smart Agent

GateKeeper uses the Smart Agent to locate server objects and GateKeeper can automatically locate the Smart Agent on the network. If the Smart Agent fails to detect the server object or if GateKeeper is unable to locate the Smart Agent automatically, you may use one of the following solutions to troubleshoot the Smart Agent:

- Check the environment variable settings described above.
- Start the Smart Agent in debugging mode:

```
osagent -v
```

- Find all Smart Agents that are reachable from where GateKeeper is installed. You may use the `osfind` command.
- Check if the IP and port addresses are set correctly in the client, GateKeeper, and the server.

Check the property files

Check the settings in the property files of the client, server and GateKeeper. The most common problem is setting the port and host addresses incorrectly.

Check the routing table

A multi-homed host allows communication between the connected networks. In order for the multi-homed host to route data packets from one network to another correctly, you must configure the routing tables correctly in the hosts. If the routing table fails to send data correctly, you may use the following methods to troubleshoot this program:

- Use `route print` and `traceroute` to check for routing tables. Identify where the communication breakdown and configure the routing tables correctly.
- Use tools such as `ping` and `tracert` can to examine and verify the communication paths.

Check pass-through connections

You may use one of the following methods to check if the pass-through connection is set correctly:

- If you are using GateKeeper in the pass-through mode, you must set the following properties in GateKeeper correctly:

  ```
  vbroker.gatekeeper.passthru.inPortMin
  vbroker.gatekeeper.passthru.inPortMax
  vbroker.gatekeeper.passthru.outPortMin
  vbroker.gatekeeper.passthru.outPortMax
  ```

  The *inPortMin* and *inPortMax* properties specify the range of ports a client uses to connect to GateKeeper. Therefore, you must ensure that the clients are able to overcome firewalls to connect to these ports.

  Similarly, the *outPortMin* and *outPortMax* properties specify the range of ports GateKeeper uses to connect to the server-side network. Therefore, you must ensure that GateKeeper is able to overcome the firewalls to connect to these ports on the server.

- Use tools such as `ping`, `tracert`, `traceroute`, and `route` to check if the destination is reachable.

Check the Java policy

Java: If the client is an applet using the java plug-in, make sure the following properties are added to the `java.policy` file. If these settings are not specified in the JRE's `java.policy` file, a security exception may occur. Note that these properties are the client's settings and “192.73.8.25:25001” is the IP and port address of GateKeeper's host and HIOP port.

```
grant codeBase "http://192.73.8.25:25001/*" {
   permission java.lang.reflect.ReflectPermission"suppressAccessChecks";
   permission java.io.SerializablePermission "enableSubclassImplementation";
   permission java.lang.RuntimePermission "accessDeclaredMembers";
};
```

Check SSL

If you are using SSL, ensure the certificate is installed properly in the client (Web browser), the server, and GateKeeper.

Check the IOR files

To check the content of an IOR file, use the following methods:

- Set the `vbroker.URLNaming.debug` property in the client, GateKeeper, or the server to trace which IOR files are retrieved.
- Use the `printior` command to print the content of an IOR file.

Check firewall settings

Firewall settings can be the most problematic settings.

- See “Configuring GateKeeper and internetworking devices,” “Configuring the firewall”, and “Configuring user programs.”
- Work closely with the network administrator to understand the firewall restrictions.
- Check the NAT (Network Address Translation) configuration.

Common errors and FAQs

1. A comprehensive list of Frequently Asked Questions, “VisiBroker GateKeeper FAQs” is maintained on the Borland web site. You may want to read this list for more information.
2. Common errors made while setting properties are spelling mistakes for property names, such as “vroker” instead of “vbroker”. Also, on Windows, some word processors automatically change the first character on a line into a capital letter. Therefore, `vbroker` becomes `Vbroker` which is not valid.
3. Socket binding errors can occur when the IP and/or port addresses are invalid or are already in use. The following table shows some typical errors:

Error Message	Error Type	Solution
`"Bind Exception: Cannot assign requested address"`	`Wrong IP Address`	Correct the IP and/or port address.
`"Bind Exception: Address in use"`	`IP port already in use`	Correct the IP and/or port address.
`"Invalid GIOP Proxy ior: Communication Problem"`	`Wrong IP Address`	Correct the IP and/or port address.
`"Invalid GIOP Proxy ior: Connect Exception"`	`Wrong Port Address`	Correct the IP addresses in the IOR file (property file)
`"Invalid GIOP Proxy ior: Invalid Object Ref, File Not Found Exception"`	`Wrong IOR Name`	Correct the name of the referenced IOR file.

Proxy servers and GateKeeper

GateKeeper can work in conjunction with HTTP proxy servers. These proxy servers are used by the HIOP protocol for the HTTP Tunneling feature of GateKeeper.

In general, the latest firewall products have a built-in capability to handle HTTP traffic. Certain firewalls have built-in HTTP proxy servers (such as Microsoft's ISA Server) while other firewalls can forward HTTP messages to an HTTP proxy server that can perform load balancing using proprietary mechanisms. In some cases, an HTTP proxy server uses caching techniques to increase performance. GateKeeper requests that HTTP proxy server caching is disabled for its messages.

When an HTTP proxy server is used in conjunction with GateKeeper, the HTTP proxy server acts like a NAT device for GateKeeper because the HTTP proxy server forwards packets. GateKeeper is hidden behind the HTTP proxy server and, as such, it is important to configure the proxy host properties or TCP firewall properties to specify the HIOP fake host/port.

Appendix

A

GateKeeper properties

This appendix describes the properties that may be set on GateKeeper with the exception of “Server's properties for firewall specifications”, which are properties set on the server.

Note

The following notations are used for the column “Default/Options” in the tables:

- Options are in bold; **gatekeeper.ior**
- <empty> is a blank space or an empty string.
- Options enclosed in angle brackets (<>) are user supplied values, for example, <port number> or <integer values>.

General properties

The following table lists the common properties used by GateKeeper.

Property	Default/Options	Description
`vbroker.gatekeeper.name`	**null** – no name is defined <a user defined name>	Specifies the name of a GateKeeper instance to differentiate it from other GateKeepers instances.
`vbroker.gatekeeper.referenceStore`	**gatekeeper.ior** – in the current directory of GateKeeper. <Relative pathname> <Full pathname>	Specifies the name of the GateKeeper IOR file. You may define the full path name if this file is not stored in the GateKeeper current directory.
`vbroker.gatekeeper.locationService`	**true** – enabled **false** – disabled	Enables or disables the location service using GateKeeper. This service is provided for clients such as applets that are not able to communicate with the Smart Agent (OSAgent) to do the bind. If this property is set to `false`, the client will get a `NO_PERMISSION` exception during the binding operation through GateKeeper.
`vbroker.gatekeeper.cache.size`	**64** (default) **1** – cache is disabled **0** – cache size is unlimited	Defines the size of the GateKeeper cache.

Property	Default/Options	Description
vbroker.gatekeeper.cache.timeout	**900** <an integer value>	Specifies the time in seconds for keeping the information in the cache. If the information is not used by the end of the timeout interval, garbage information is collected.
vbroker.gatekeeper.asynchronizedIO	**false** (default) – enabled **true** – disabled	Enables or disables the asynchronized IO feature in GateKeeper. The asynchronized IO feature is only useful in situations in which calling methods on the servers takes a long time and there are a lot of new incoming clients. In most situations, activating this feature does not provide any advantages. This feature exists mainly due to historical reasons and the use of it is discouraged.

Exterior server engine

The following table lists the properties used by the exterior server engine on the client side or Internet side of GateKeeper. Most of the important properties, however, are defined in each Server Connection Manager (SCM) which are described in the following sections.

Property	Default	Description
vbroker.se.exterior.scms	**ex-iiop,ex-hiop** – ex-iiop and ex-hiop Server connection managers in use. It can also be a list of ex-iiop, ex-hiop, ex-ssl, and ex-hiops separated by commas.	Defines the server connection managers for the exterior server engine.
vbroker.se.exterior.host	**null** – the primary host is used; that is the IP address of the primary Network Interface Card (NIC). <a host address>	The host address of the exterior host. The primary NIC is the exterior NIC. You can set this property in the <Basic Properties> panel of GateKeeper.
vbroker.se.exterior.proxyHost	**<empty>** – No proxy host. <a fake host address>	Network Address Translation (NAT) devices hide the actual IP address and/or port number in the network by changing the IP address and/or in the IP packet. Set this value to the value defined by the NAT. When you have callback enabled and GateKeeper sits behind a NAT, the callback proxy host (vbroker.gatekeeper.backcompat.callback.proxyHost) should be set to equal this property. This is used when GateKeeper sits behind a NAT. You can also set this property on the Basic Properties panel of the VisiBroker Console.
vbroker.se.exterior.type	**gatekeeper**	This setting says that the exterior server engine is in a “proxying” role. It means packet/messages forwarding function is enabled on this server engine. The user should not change this setting.

ex-hiop server connection manager (SCM)

Java

The ex-hiop server connection manager is responsible for servicing HTTP requests on the exterior server engine. Both the listener and dispatcher properties are configured using the property with the vbroker.scm.exterior.ex-hiop prefix.

The following vbroker.se.exterior.scm.ex-hiop properties specify the behavior of the ex-hiop listener. The ex-hiop listener is an HIOP listener. The default port is 8088. The threading policy is set to ThreadSession.

Note

All the properties related to an SCM are defined with the following prefix: `vbroker.se.<server engine name>.scm.<server connection manager name>.`

Some SCMs may define additional properties, but some properties, especially the properties related to threads and connections, have the same property names for all SCMs.

Property	Default	Description
`vbroker.se.exterior.scm.ex-hiop.root`	. <Full path of a file directory>	Sets to the root directory as its default. The default directory is where GateKeeper starts.
`vbroker.se.exterior.scm.ex-hiop.dispatcher.type`	**ThreadSession**	Specifies the dispatcher type for a ex-hiop scm request. The type should always be set to “Thread Session”.
`vbroker.se.exterior.scm.ex-hiop.listener.port`	**8088** <a port number>	Sets the default listener port for GateKeeper's client side (exterior) HIOP listener. Client applications or applets use this port primarily for HTTP tunneling support. This port may also be used on a limited basis for standard HTTP requests such as fetching the GateKeeper IOR file.
`vbroker.se.exterior.scm.ex-hiop.listener.proxyPort`	<empty> <a fake port number>	The `proxyPort` property is often used in conjunction with the server engine `proxyHost` property to mask the target port for this listener. If this property is set, the GateKeeper IOR file will contain the `proxyPort` value in the endpoint information for this listener. It is then the responsibility of the external NAT device to map the `proxyPort` to the listener's true port. The default is <empty> indicating the feature is disabled (the listener port will not be masked).
`vbroker.se.exterior.scm.ex-hiop.listener.type`	**HIOP**	Specifies the listener type for the ex-hiops SCM. HIOP indicates that this listener supports the proprietary HIOP protocol used for HTTP tunneling as well as the HTTP protocol. The listener type for pre-configured server connection managers such as ex-hiop should not be changed.
`vbroker.se.exterior.scm.ex-hiop.manager.connectionMax`	**0** – incoming connections are unlimited <an integer value>	Defines the maximum number of incoming connections allowed to the GateKeeper exterior HIOP listener.
`vbroker.se.exterior.scm.ex-hiop.manager.connectionMaxIdle`	**0** <an integer value>	Specifies the time in seconds that an inactive connection is idle before it can be closed.
`vbroker.se.exterior.scm.ex-hiop.manager.type`	**Socket**	It specifies the type of Server Connection Manager.
`vbroker.se.exterior.scm.ex-hiop.servletList`	**orb**	The servlet class type to be used.
`vbroker.se.exterior.scm.ex-hiop.servlet.orb.GET`	**true** – enabled **false** – disabled	Enables or disables the servlet GET operation. Note that the ORB is a predefined property.
`vbroker.se.exterior.scm.ex-hiop.servlet.orb.PUT`	**true** – enabled **false** – disabled	Enables or disables the servlet PUT operation. Note that the ORB is a predefined property.

Property	Default	Description
vbroker.se.exterior.scm.ex-hiop.servlet.orb.class	com.inprise.vbroker.HIOP.servlets.ORB.Servlet	It specifies the java class to be loaded for HIOP when GateKeeper is used as a servlet. The customer should not change this property to other values.
vbroker.se.exterior.scm.ex-hiop.servlet.orb.load	**false** – disabled **true** – enabled	Enables or disables the servlet Load operation. Note that the ORB is a predefined property.

ex-iiop server connection manager (SCM)

The ex-iiop server connection manager is responsible for servicing IIOP requests on the exterior sever engine. The listener and dispatcher properties can be configured using properties with the vbroker.se.exterior.scm.ex-iiop prefix. The following vbroker.se.exterior.scm.ex-iiop properties specify the behavior of the ex-iiop listener. The ex-iiop listener is an IIOP listener.

Property	Default	Description
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMax	100 <an integer value>	Specifies the maximum number of threads the server connection manager can create.
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMaxIdle	300 <an integer value>	Specifies the time an idle thread is idle before it is destroyed. The default is 300.
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMin	0 <an integer value>	Specifies the minimum number of threads the server connection manager can create.
vbroker.se.exterior.scm.ex-iiop.dispatcher.type	ThreadPool	Specifies the dispatcher type for the ex-iiop scm.
vbroker.se.exterior.scm.ex-iiop.listener.giopVersion	1.2	It sets the GIOP version to be used by the GIOP messages. This property can be used to overcome interoperability problems with older ORBs that cannot handle unknown minor GIOP versions correctly.
vbroker.se.exterior.scm.ex-iiop.listener.port	683 <a port number>	Sets as a default listener port for GateKeeper's client-side IIOP listener. Port 683 is the recommended setting for a deployed application since it is an OMG standard for IIOP and is registered with IANA. **UNIX:** On a UNIX platform, the default listener port number is in the range of 0 to 1024 which is reserved for privileged use. When running as a non-privileged user, the listener port can be set to a value greater than 1024 if desired.
vbroker.se.exterior.scm.ex-iiop.listener.proxyPort	**<empty>** – The proxy port feature is disabled. This indicates the feature is disabled (the listener port will not be masked). <a fake port number>	Specifies the proxy port number used with the proxy host name property.
vbroker.se.exterior.scm.ex-iiop.listener.type	IIOP	Specifies the listener type for the ex-iiop scm.

Property	Default	Description
vbroker.se.exterior.scm.ex-iiop.manager.connectionMax	**0** – incoming connections are unlimited <an integer value>	Defines the maximum number of incoming connections allowed to the GateKeeper exterior IIOP listener.
vbroker.se.exterior.scm.ex-iiop.manager.connectionMaxIdle	0 <an integer value>	Specifies the time in seconds in which a inactive connection is idle before it can be closed.
vbroker.se.exterior.scm.ex-iiop.manager.type	Socket	It specifies the type of Server Connection Manager. Currently, only “Socket” is available.

ex-hiops server connection manager (SCM)

The ex-hiops server connection manager is responsible for servicing HTTPS requests on the exterior server engine. Both the listener and dispatcher properties are configured using the property with the vbroker.scm.exterior.ex-hiops prefix.

The following vbroker.se.exterior.scm.ex-hiops properties specify the behavior of the ex-hiops listener. The ex-hiops listener is an HIOPS listener. The default port is 8089. The threading policy must always be ThreadSession.

Property	Default	Description
vbroker.se.exterior.scm.ex-hiops.root	. <Full path of a file directory>	Sets to the root directory as its default. The default directory is where GateKeeper starts.
vbroker.se.exterior.scm.ex-hiops.dispatcher.type	ThreadSession	Specifies the dispatcher type for ex-hiop scm request. The type should always be set to “ThreadSession”.
vbroker.se.exterior.scm.ex-hiops.listener.port	8089 <a port number>	Sets the default listener port for the GateKeeper's client-side (exterior) HIOPS listener. Client applications or applets use this port primarily for HTTPS tunneling support. This port may also be used on a limited basis for standard HTTPS requests such as fetching the GateKeeper IOR file.
vbroker.se.exterior.scm.ex-hiops.listener.proxyPort	**<empty>** <a fake port number>	The proxyPort property is often used in conjunction with the server engine proxyHost property to mask the target port for this listener. If this property is set, the GateKeeper IOR file will contain the proxyPort value in the end point information for this listener. It is then the responsibility of the external NAT device to map the proxyPort to the listener's true port. The default is <empty> indicating the feature is Disabled (the listener port will not be masked).
vbroker.se.exterior.scm.ex-hiops.listener.type	HIOPS	Specifies the listener type for the ex-hiops SCM. HIOPS indicates that this listener supports the proprietary HIOPS protocol used for HTTP tunneling as well as the HTTPS protocol. The listener type for pre configured server connection managers such as ex-hiops should not be changed.

Property	Default	Description
vbroker.se.exterior.scm.ex-hiops.manager.connectionMax	**0** – the cached connections are unlimited <an integer value>	Defines the maximum number of cached connections available to the GateKeeper exterior HIOPS listener.
vbroker.se.exterior.scm.ex-hiops.manager.connectionMaxIdle	**0** <an integer value>	Specifies the time in seconds in which a inactive connection is idle before it can be closed.
vbroker.se.exterior.scm.ex-hiops.manager.type	Socket	It specifies the type of Server Connection Manager.
vbroker.se.exterior.scm.ex-hiops.servletList	orb	The servlet class type to be used.
vbroker.se.exterior.scm.ex-hiops.servlet.orb.GET	**true** – enabled **false** – disabled	Enables or disables the servlet GET operation. Note that the ORB is a predefined property.
vbroker.se.exterior.scm.ex-hiops.servlet.orb.PUT	**true** – enabled **false** – disabled	Enables or disables the servlet PUT operation. Note that the ORB is a predefined property.
vbroker.se.exterior.scm.ex-hiops.servlet.orb.class	com.inprise.vbroker.HIOP.servlets.ORB.Servlet	It specifies the java class to be loaded for HIOPS when GateKeeper is used as a servlet. The customer should not change this property to other values.
vbroker.se.exterior.scm.ex-hiops.servlet.orb.load	**false** – disabled **true** – enabled	Enables or disables the servlet Load operation. Note that the ORB is a predefined property.

ex-ssl server connection manager (SCM)

The ex-ssl server connection manager is responsible for servicing SSL requests on the exterior sever engine. The listener and dispatcher properties can be configured using properties with the vbroker.se-exterior.scm.ex-ssl prefix.

The following `vbroker.se.exterior.scm.ex-ssl` properties specify the behavior of the ex-ssl listener. The ex-ssl listener is an ssl listener.

Property	Default	Description
vbroker.se.exterior.scm.ex-ssl.dispatcher.threadMax	100 <an integer value>	Specifies the maximum number of threads the server connection manager can create.
vbroker.se.exterior.scm.ex-ssl.dispatcher.threadMaxIdle	300 <an integer value>	Specifies the time an idle thread is idle before it is destroyed. The default is 300.
vbroker.se.exterior.scm.ex-ssl.dispatcher.threadMin	0 <an integer value>	Specifies the minimum number of threads the server connection manager can create.
vbroker.se.exterior.scm.ex-ssl.dispatcher.type	ThreadPool	Specifies the dispatcher type for the ex-iiop scm.
vbroker.se.exterior.scm.ex-ssl.listener.port	684 <a port number>	Sets as a default listener port for the GateKeeper's client-side SSL listener. Port 684 is the recommended setting for deployed application since it is an OMG standard for IIOP and is registered with IANA. **UNIX:** On a UNIX platform, the default listener port number is in the range 0 to 1024 which is reserved for privileged use. When running as a non-privileged user, the listener port can be set to a value greater than 1024 if desired.

Property	Default	Description
vbroker.se.exterior.scm.ex-ssl.listener.proxyPort	**<empty>** – The proxy port feature is disabled. This indicates the features is disabled (the listener port will not be masked). <a fake port number>	Specifies the proxy port number used with the proxy host name property.
vbroker.se.exterior.scm.ex-ssl.listener.type	SSL	Specifies the listener type for the ex-ssl scm.
vbroker.se.exterior.scm.ex-ssl.manager.connectionMax	**0** – cached connections are unlimited. <an integer value>	Defines the maximum number of cached connections available to the GateKeeper exterior SSL listener.
vbroker.se.exterior.scm.ex-ssl.manager.connectionMaxIdle	**0** <an integer value>	Specifies the time in seconds in which an inactive connection is idle before it can be closed.
vbroker.se.exterior.scm.ex-ssl.manager.type	**Socket**	Specifies the type of Server Connection Manager. Currently, only “Socket” is available.

Interior server engine

The following table lists the properties used by the interior server engine on the server-side or Intranet side of GateKeeper.

You may need to set some of the properties in the interior server engine in special cases, such as when GateKeeper runs on a dual-homed machine or if there is a Network Address Translation (NAT) between GateKeeper and the server.

Property	Default	Description
vbroker.se.interior.scms	**in-iiop in-hiop in-ssl in-hiops** <Combination of the above separated by commas.>	This defines the server connection managers for the server engine. The default is IIOP scm. However, you may choose to use other types of protocols, such as SSL by specifying its respective scms.
vbroker.se.interior.host	**Null** – the primary host is used; that is the IP address of the primary Network Interface Card (NIC). <a host address>	The interior server engine host address. You can also set this property on the Basic Properties panel of the VisiBroker Console.
vbroker.se.interior.proxyHost	**<empty>** – The proxy port feature is disabled. <a fake host address>	This property can be used if you have a NAT running between GateKeeper and the server to hide the server host's address. You can also set this property on the Basic Properties panel of the VisiBroker Console.

in-iiop server connection manager (SCM)

The in-iiop server manager is responsible for servicing IIOP requests on the interior server engine. The listener and dispatcher can be configured using properties with the `vbroker.se.interior.in-iiop` prefix.

The following `vbroker.se.interior.scm.in-iiop` properties specify the behavior of the in-iiop server connection manager.

Property	Default	Description
`vbroker.se.interior.scm.in-iiop.dispatcher.threadMax`	100 <an integer value>	Specifies the maximum number of threads the server connection manager can create.
`vbroker.se.interior.scm.in-iiop.dispatcher.threadMaxIdle`	300 <an integer value>	Specifies the time an idle thread is idle before it is destroyed. The default is 300.
`vbroker.se.interior.scm.in-iiop.dispatcher.threadMin`	0 <an integer value>	Specifies the minimum number of threads the server connection manager can create.
`vbroker.se.interior.scm.in-iiop.dispatcher.type`	**ThreadPool**	Specifies the dispatcher type for the in-iiop scm. The type should always be set to “ThreadPool”.
`vbroker.se.interior.scm.in-iiop.listener.giopVersion`	**1.2**	The protocol version number to be used for GIOP messages.
`vbroker.se.interior.scm.in-iiop.listener.port`	**0** – pick a random number <a port number>	Specifies the port number used with the host name property.
`vbroker.se.interior.scm.in-iiop.listener.proxyPort`	**<empty>** – The proxy port feature is disabled. <a fake port number>	Specifies the proxy port number used with the proxy host name property.
`vbroker.se.interior.scm.in-iiop.listener.type`	**IIOP**	Specifies the listener type for the in-iiop scm.
`vbroker.se.interior.scm.in-iiop.manager.connectionMax`	**0** – cached connections are unlimited. <an integer value>	Defines the maximum number of cached connections available to the GateKeeper IIOP listener.
`vbroker.se.interior.scm.in-iiop.manager.connectionMaxIdle`	0 <an integer value>	Specifies the time in seconds in which an inactive connection is idle before it can be closed.
`vbroker.se.interior.scm.ex-iiop.manager.type`	**Socket**	Specifies the type of Server Connection Manager. Currently, only Socket is available.

in-ssl server connection manager (SCM)

The in-ssl server manager is responsible for servicing SSL requests on the interior server engine. The listener and dispatcher can be configured using properties with the vbroker.se.interior.in-ssl prefix.

The `vbroker.se.interior.scm.in-ssl` properties listed below specify the behavior of the in-ssl server connection manager.

Property	Default	Description
`vbroker.se.interior.scm.in-ssl.dispatcher.threadMax`	100 <an integer value>	Specifies the maximum number of threads the server connection manager can create.
`vbroker.se.interior.scm.in-ssl.dispatcher.threadMaxIdle`	300 <an integer value>	Specifies the time an idle thread is idle before it is destroyed. The default is 300.

Property	Default	Description
vbroker.se.interior.scm.in-ssl.dispatcher.threadMin	0 <an integer value>	Specifies the minimum number of threads the server connection manager can create.
vbroker.se.interior.scm.in-ssl.dispatcher.type	**ThreadPool**	Specifies the dispatcher type for the in-iiop scm. The type should always be set to ThreadPool.
vbroker.se.interior.scm.in-ssl.listener.port	**0** – pick a random number <a port number>	Specifies the port number used with the host name property.
vbroker.se.interior.scm.in-ssl.listener.proxyPort	**<empty>** – The proxy port feature is disabled. <a fake port number>	Specifies the proxy port number used with the proxy host name property.
vbroker.se.interior.scm.in-ssl.listener.type	**SSL**	Specifies the listener type for the in-ssl scm.
vbroker.se.interior.scm.in-ssl.manager.connectionMax	**0** – incoming connections are unlimited. <an integer value>	Defines the maximum number of incoming connections allowed to the GateKeeper interior SSL listener.
vbroker.se.interior.scm.in-ssl.manager.connectionMaxIdle	0 <an integer value>	Specifies the time in seconds in which an inactive connection is idle before it can be closed.
vbroker.se.interior.scm.ex-ssl.manager.type	**Socket**	Specifies the type of Server Connection Manager. Currently, only Socket is available.

Administration

Java The following table lists the administration properties. Note that the default listener port number is 9091.

Property	Default/Options	Description
vbroker.se.iiop_tp.host	**null** – use host address from the system. <Host address>	Specifies the host address that can be used by this server engine.
vbroker.se.iiop_tp.ProxyHost	**<empty>** – use host address from the system. <Proxy host address>	Specifies the proxy host address that can be used by this server engine.
vbroker.se.iiop_tp.scms	**iiop_tp**, **hiop_ts**	Specifies the list of Server Connection Managers name(s).
vbroker.se.iiop_tp.scm.iiop_tp.listener-port	**<empty>** <a port number>	Specifies the IIOP administrative listener port.
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort	**<empty>** <a port number>	Specifies the proxy port for IIOP administrative listener port.
vbroker.se.iiop_tp.scm.hiop_ts.listener.port	**9091** <a port number>	Specifies the GateKeeper administrative listener port.
vbroker.se.iiop_tp.scm.hiop_ts.listener.proxyPort	**<empty>** – The proxy port feature is disabled <a fake port number>	Specifies the proxy port number for HIOP administrative listener port.
vbroker.se.iiop_tp.type	**normal**	This is applicable to the Administrative Server Engine.
vbroker.se.iiop_tp.scm.hiop_ts.servletList	**orb**	A virtual name given to the servlet class. It is used for specifying other properties like PUT, GET, LOAD and etc.
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.GET	**true** – enabled **false** – disabled	Enables or disables the servlet GET operation.
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.PUT	**true** – enabled **false** – disabled	Enables or disables the servlet PUT operation.

Property	Default/Options	Description
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.class	**com.inprise.vbroker.HIOP.servlets.ORBServlet**	The name of the servlet class.
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.load	**true** – enabled **false** – disabled	Enables or disables the servlet Load operation.

Access control

The following table lists the properties used to set security control in GateKeeper.

Property	Default	Description
vbroker.gatekeeper.security.accessControllers	**default**	Specifies the list of names of access controllers.
vbroker.gatekeeper.security.acl.<controllerName>.default	**null** – no default action **deny** – refused entry **grant** – permission granted	Specifies the default action for the control list. <controllerName> is specified above.

vbroker.gatekeeper.security.acl.<controllerName>.<xx>
where <xx> is the name of the given rule.
	null – no actions specified. See description for additional options.	Defines the action for the specific properties for the given rule. The definition is as follows:
		`<deny|grant>`
		`[operation="<operation name>"`
		`[signer by="<signer's company name>"]`
		`[server host="<hostname>"]`
		`[client host="<hostname>"]`
		`[server ip=aa.bb.cc.dd|<sub-mask>]`
		`[client ip=aa.bb.cc.dd|<sub-mask>]`
		`[object interface="<repId>"] ]`
		<deny|grant> defines the action related to the individual rule.
		operation="<**operation name**>" defines the related operation name based on the IDL.
		signer by="<**signer's company name**>" defines the signer's company name.
		server host="<**hostname**>" specifies the server hostname.
		client host="<**hostname**>" specifies the client hostname.
		server ip="<**aa.bb.cc.dd**>" specifies the IP address of the machine that the server resides on.
		client ip="<**aa.bb.cc.dd**>" defines the IP address of the machine that the client resides on.
		object interface="<repId>" defines the object type.
		Examples:
		`vbroker.gatekeeper.security.accessControllers=default`
		`vbroker.gatekeeper.security.acl.default.rules=rule1,rule2,rule3`
		`vbroker.gatekeeper.security.acl.default.rule1=grant`
		`[operation=\"*"\`
		`[server host=\"borland"\]]`
		`vbroker.gatekeeper.security.acl.default.rule2=deny`
		`[operation=\"*"\`
		`[client ip=192.168.100.40|255.255.255.0]]`
		`vbroker.gatekeeper.security.acl.default.rule3=deny`
		`[operation=\"*"\`
		`[server host=\"inprise"\]`
		`[client ip=192.168.100.88|255.255.255.0]]`
		Note: Variables in **Bold** are user-definable.

Property	Default	Description
vbroker.gatekeeper. security.acl. <controllerName>.type	**com.inprise.vbroker. gatekeeper.security. ACImpl**	Specifies the implementation class to be loaded by GateKeeper for Access Control. **Note:** User should not change this value.
vbroker.gatekeeper. security.acl. <controllerName>.rules	**null** – no rules specified. See description.	Specifies the names for the set of rules. Example: vbroker.gatekeeper.security.accessControllers=default vbroker.gatekeeper.security.acl.default.rules=rule1,rule2,rule3 where **rule1, rule2, rule3** are names defined by the user.

VisiBroker 3.x style callback

The following table lists the properties that can be set in VisiBroker 5.x to use the VisiBroker 3.x style callback.

Property	Default/Options	Description
vbroker.gatekeeper.callbackEnabled	**false** – disabled **true** – enabled	Enables or disables the callback mechanism through GateKeeper. The default is **disabled**.
vbroker.gatekeeper.backcompat.callback	**false** – disabled **true** – enabled	Enables or disables the VisiBroker 3.x style callbacks. If you wish to use the old style callbacks, set both the above and this property to **true**.
vbroker.gatekeeper.backcompat.callback.listeners	iiop ssl	Specifies a list of listeners.
vbroker.gatekeeper.backcompat.callback.listener.iiop.type	**IIOPCallback**	Specifies the type of IIOP callback listeners.
vbroker.gatekeeper.backcompat.callback.host	**<empty>** – the value is set to the primary host IP address and that is the IP address of the primary NIC. <a host address>	Defines the host IP address to bind for the callback listener.
vbroker.gatekeeper.backcompat.callback.proxyHost	<empty> – no proxy host is used. <a fake host address>	This property is often used in conjunction with the server engine proxyPort property to mask the target port for this listener. If this property is set, the callback IOR will contain the proxyHost value in the end point information for this listener. It is then the responsibility of the Network Address Translation (NAT) device to map the proxyHost to the listener's true port. The default is **<empty>** indicating that the feature is disabled (the listener port will not be masked in the callback IOR).
vbroker.gatekeeper.backcompat.callback.listener.ssl.type	**SSLCallback**	Specifies the type of SSL listener.
vbroker.gatekeeper.backcompat.callback.listener.ssl.port	**0** – a port picked at random <port number>	Defines the port on which the callback listener uses to listen for SSL.
vbroker.gatekeeper.backcompat.callback.listener.ssl.proxyPort	<empty> – no proxy port is used> <fake port number>	Use this property when a NAT sits between the client and GateKeeper in which the NAT hides the actual GateKeeper host address.

Property	Default/Options	Description
vbroker.gatekeeper.backcompat.callback.listener.iiop.port	**0** – a port number is picked at random. <a port number>	Specifies the port in which the callback listeners listen for the IIOP.
vbroker.gatekeeper.backcompat.callback.listener.iiop.proxyPort	**<empty>** – no proxy port is used. <a fake port number>	The proxyPort property is often used in conjunction with the server engine proxyHost property to mask the target port for this listener. If this property is set, the callback IOR will contain the proxyPort value in the end point information for this listener. It is then the responsibility of the Network Address Translation (NAT) device to map the proxyPort to the listener's true port. The default is **<empty>** indicating that the feature is disabled (the listener port will not be masked in the callback IOR).

Performance and load balancing

The following table lists the performance and load balancing properties to distribute and monitor the load between client and server.

Property	Default	Description
vbroker.gatekeeper.load.distributor	**com.inprise.vbroker.gatekeeper.ext.RoundRobinDistributor**	Specifies the Distributor class to be used by the Load Balance Manager. Users should not change this property unless they understand the distributor interface and implement their own distributor. The following implementation class is used as a default distributor: (Round Robin):com.inprise.vbroker.gatekeeper.ext.RoundRobinDistributor.
vbroker.ce.iiop.ccm.connectionMax	**0** – will not try to close any of the old active or cached connections. <an integer value>	Specifies the maximum number of total connections within a client. This value is equal to the number of active connections plus those that are cached.
vbroker.orb.gcTimeout	30 <an integer value>	Specifies the time in seconds that must pass before important resources that are not used are cleared.
vbroker.orb.fragmentSize	**0**	Specifies the GIOP fragment size. It must be a multiple of GIOP stream chunk size.
vbroker.orb.streamChunkSize	**4096** <a number which is the power of 2>	Specifies the GIOP message chunk size. It must be the power of 2.
vbroker.orb.bufferCacheTimeout	**6000** <an integer value>	Specifies (in milliseconds) the time in which a message chunk has been cached before it is discarded.

Property	Default	Description
vbroker.gatekeeper.load.slaves	**<empty>** – no slaves GateKeeper <a list of slaves> – see description.	Specifies the list of Slave GateKeepers to be clustered together for the purpose of Load Balancing by this Master GateKeeper. The list of names is separated by commas. For each name in the list, a property such as "vbroker.gatekeeper.load.slave.<slave_name>" is added to the property file. There is no default value for this property. Note that this property is used in the Master GateKeeper property only. For example: `vbroker.gatekeeper.load.slaves=abc,xyz.` **Note:** If you set this property, you must load the appropriate library. See the description of `vbroker.orb.dynamicLibs` in "Miscellaneous ORB properties".

vbroker.gatekeeper.load.slave.<slave_name>	**<empty>**
	<Specific Slave IOR> – see description.
		Specifies the IOR or a URL pointing to specific Slave GateKeeper to be clustered for the purpose of Load Balancing by this Master GateKeeper. **Note** that this property is used in Master GateKeeper property only. For example:
		`vbroker.gatekeeper.load.slave.abc=http://host1:9091/GateKeeper.ior`
		`vbroker.gatekeeper.load.slave.xyz=http://host2:9091/GateKeeper.ior.`
		Note: If you set this property, you must load the appropriate library. See the description of `vbroker.orb.dynamicLibs` in "Miscellaneous ORB properties".

vbroker.gatekeeper.load.balancer	**<empty>** <master>	Specifies that the master GateKeeper's sole purpose is to do load balancing and it will never take a turn in serving clients. In the default mode, the master itself is also a slave, in the sense that it is included in the list of available GateKeepers and will take turn among the slaves. When a particular GateKeeper is unavailable, the client will come back to the master to obtain the next slave GateKeeper in turn (this can be one of the slaves or the master itself). **Note:** If you set this property, you must load the appropriate library. See the description of `vbroker.orb.dynamicLibs` in "Miscellaneous ORB properties".

Support for bidirectional communications

The following table lists the properties that support bidirectional communications. These properties are evaluated only once, when the SCMs are created. In all cases, the `exportBiDir` and `importBiDir` properties on the SCMs are given priority over the `enableBiDir` property. In other words, if both properties are set to conflicting values, the SCM-specific properties will take effect. This allows you to set the `enableBiDir` property globally, and more importantly, turn off bidirectionality in individual SCMs.

Property	Default	Description
`vbroker.orb.enableBiDir`	both server and client: none See "Callback with GateKeeper's bidirectional support".	You can selectively make bidirectional connections. If the client defines `vbroker.orb.enableBiDir=client`, and the server defines `vbroker.orb.enableBiDir=server`, the value of `vbroker.orb.enableBiDir` in GateKeeper determines the state of the connection. **Note:** Just as you can selectively enable bidirectional communication on a per-SCM basis, you can also selectively enable bidirectional communication on GateKeeper. For example, if you set the `vbroker.se.exterior.scm.ex--iiop.manager.importBiDir` property to `true`, GateKeeper will accept bidirectional connections from the client. Setting the `vbroker.se.exterior.scm.ex--iiop.manager.exportBiDir` property to `true` causes GateKeeper to request bidirectional connections with the server.

Support for pass-through connections

Note

The `vbroker.gatekeeper.enablePassthru` property is the only property that supports pass-through connections.

Property	Default	Description
`vbroker.gatekeeper.enablePassthru`	**false** – disabled **true** – enabled	Specifies enabled or disabled Passthru mode in GateKeeper.
`vbroker.gatekeeper.passthru.blockSize`	**16384** <an integer value>	Specifies the buffer size that the channel uses for each read and write operation. A high value handles large messages with a single read and write, but increases the resources used by a single channel. A low value will optimize resource utilization, while degrading performance due to multiple reads and writes.
`vbroker.gatekeeper.passthru.connectionTimeout`	**300000 milliseconds** (5 minutes) <an integer value>	Specifies the amount of time in milliseconds a given channel will wait before it stops waiting for connections and shuts down the channel.
`vbroker.gatekeeper.passthru.inPortMin`	**1024** <a port number>	Used together with `vbroker.gatekeeper.passthru.inPortMax`. It specifies the start of a range of interior port for pass-through incoming connections.
`vbroker.gatekeeper.passthru.inPortMax`	**65535** <a port number>	Used together with `vbroker.gatekeeper.passthru.inPortMin`. It specifies the end of a range of ports for pass-through in-coming connections.

Property	Default	Description
vbroker.gatekeeper.passthru.logLevel	**0** – no logging <an integer value>	Enables the level of logging for the pass-through component.
vbroker.gatekeeper.passthru.outPortMin	**0** <a port number>	Used together with vbroker.gatekeeper.passthru.outPortMax. It specifies the start of a range of exterior port for pass-through outgoing connections.
vbroker.gatekeeper.passthru.outPortMax	**65535** <a port number>	Used together with vbroker.gatekeeper.passthru.outPortMin. It specifies the end of a range of exterior port for pass-through outgoing connections.
vbroker.gatekeeper.passthru.streamTimeout	**2000** <an integer value>	Specifies the amount of time in milliseconds an established channel will wait for messages before it shuts down.

Security services (SSL)

The following table lists the properties used in the Security Services.

Note

If you set this property then you load the appropriate library. See the description of vbroker.orb.dynamicLibs in "Miscellaneous ORB properties".

Property	Default	Description
vbroker.orb.alwaysSecure	**false** – disabled **true** – enabled	Specifies whether GateKeeper will make secure connections to the server.
vbroker.security.peerAuthenticationMode	See description.	Refer to the Security Properties for Java or the Security Properties for C++ section in the *Security Guide* for more details.
vbroker.security.trustpointsRepository	See description.	Refer to the Security Properties for Java or the Security Properties for C++ section in the *Security Guide* for more details.
vbroker.security.wallet.identity	See description.	Refer to the Security Properties for Java or the Security Properties for C++ section in the *Security Guide* for more details.
vbroker.security.wallet.password	See description.	Refer to the Security Properties for Java or the Security Properties for C++ section in the *Security Guide* for more details.

Location services (Smart Agent)

The following table lists the Smart Agent (OSAgent) properties used in the Location Service to locate server objects.

Property	Default	Description
vbroker.agent.addr	**null** – see description.	Specifies the IP address or host name of the host running the Smart Agent (OSAgent). The default value, **null**, installs VisiBroker applications to use the value from the OSAGENT_ADDR environment variable. If the OSAGENT_ADDR variable is not set,it is assumed that the Smart Agent is running on the local host or will be located by a broadcast message. Refer to “Using the Smart Agent” in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more details.
vbroker.agent.port	**null**	Specifies the port number that defines a domain within your network. VisiBroker applications and Smart Agent (OSAgent) work together when they have the same port number. This is the same property as the OSAGENT_PORT environment variable. Refer to “Using the Smart Agent” section in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more details.
vbroker.agent.addrFile	**null**	Specifies a file that stores information on where the IP address(es) or host names(s) of the Smart Agent may be found. Refer to “Using the Smart Agent” section in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more details.
vbroker.agent.failOver	**true** **false**	When set to **true**, allows a VisiBroker application to failover to another Smart Agent. Refer to “Using the Smart Agent” section in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more details.
vbroker.agent.enableCache	**true** **false**	When set to **true**, allows VisiBroker applications to cache object references. Setting this property to **true** improves performance when locating servers, but disables Smart Agent round-robin activity. Refer to “Using the Smart Agent” section in the *VisiBroker for C++ Developer's Guide* or the *VisiBroker for Java Developer's Guide* for more details.

Backward compatibility with VisiBroker 4.x and below

GateKeeper Version 5.x by default is not compatible with programs developed with VisiBroker 4.x and below. To make GateKeeper Version 5.x run properly with programs developed with VisiBroker 4.x and below, set the following property to **true**.

Note

Earlier versions of GateKeeper are by default compatible with older programs developed with Visibroker 4.x and below. With GateKeeper 5.x, however, you must explicitly set this property.

Property	Default	Description
`vbroker.orb.enableVB4backcompat`	**true** **false**	Specifies whether GateKeeper is compatible with older VisiBroker versions. Setting the property to **false** makes GateKeeper compatible with programs developed with VisiBroker 4.5.x onwards. Setting this property to **true** makes GateKeeper compatible with versions earlier than VisiBroker 4.5.x as well. (See Appendix B for more information.) **Note:** This value is set to **true** by default in GateKeeper. This value, however, is **false** by default on the client and server.

Server's properties for firewall specifications

Note

These properties should only be set in the property file for the server. If you set any of these properties then you load the appropriate library. See the description of `vbroker.orb.dynamicLibs` in the section "Miscellaneous ORB properties".

The following properties specify the communication paths from the client to the server. See "Specifying communication paths to the server" for examples of its usage.

Property	Default	Description
`vbroker.se.iiop_tp.firewallPaths`	**<empty>** <List of paths>	Specifies a list of communication paths from the clients to the servers. <List of paths> is a set of user defined names for the paths separated with commas. An example of the <List of paths> is: `vbroker.se.iiop_tp.firewallPaths=x,y`

`vbroker.firewall-path.<pathname>`	**<empty>**
	<List of components>
		Specifies the list of components in the firewall path <pathname>. For example,
		`vbroker.firewall-path.x=a,b vbroker.`
		`firewall-path.y=c`

`vbroker.firewall.<component>.type`	**<empty>**
	PROXY TCP
		Specifies the type of the components. For example:
		`vbroker.firewall.a.type = PROXY`
		`vbroker.firewall.b.type = TCP`

`vbroker.firewall.<component>.ior`	**<empty>**
	<Filename of ior file>
	<URL of the ior file>
	IOR:<GateKeeper's stringified ior>
		Specifies the ior of the component. This is specified together with `vbroker.firewall.<component.type>=PROXY`. Examples of the values are:
		`1. file:C:/GateKeeper/GateKeeper.ior`
		`2. http://www.inprise.com/GK GateKeeper.ior`
		`3. IOR:23984028417290734234972342342 34`

`vbroker.firewall.<component>.host`	**<empty>** <fake host name>	Specifying fake host of the server. This is specified together with `vbroker.firewall.<component>.types=TCP` and the component is a TCP Firewall with NAT.
`vbroker.firewall.<component>.iiop_port`	**<empty>** <fake IIOP Port>	Specifies a fake IIOP port for the server. This is specified together with `vbroker.firewall.<component>.types=TCP` and the component is a TCP Firewall with NAT.

Property	Default	Description
vbroker.firewall.<component>.ssl_port	**<empty>** <fake SSL Port>	Specifies a fake SSL port for the server. This is specified together with vbroker.firewall.<component>.type=TCP and the component is a TCP Firewall with NAT.
vbroker.firewall.<component>.hiop_port	**<empty>** <fake HIOP Port>	Specifies a fake HIOP port for the server. This is specified together with vbroker.firewall.<component>.type=TCP and the component is a TCP Firewall with NAT.

Miscellaneous ORB properties

These properties are common ORB objects and are directly and indirectly related to GateKeeper. They are not necessarily set in the GateKeeper property file, so please read each description carefully.

Property	Default	Description
vbroker.orb.gatekeeper.ior	**<empty>** <ior filename>	Specifies the URL of the GateKeeper IOR file. This property must be set in the HTML files with applets because of the change in the behavior of the Applet Viewer.
vbroker.orb.alwaysProxy	**false** – disable **true** – enabled	Specifies whether the client must always connect to the server via GateKeeper. This property can be set in the client or GateKeeper. If set in the client, the client will always connect to the server via GateKeeper. If set in GateKeeper, it will connect to the server via another GateKeeper. See the *VisiBroker Programmer's Reference* for more details.
vbroker.locator.ior.ior	**<empty>** <ior filename>	Specifies URL of the GateKeeper IOR file. This property is usually set in the client applet, but can also be set in an application. **Note:** GateKeeper provides limited location services. It cannot forward location requests to another GateKeeper. This is in contrast to the Smart Agent which is able to forward requests to another available Smart Agent.
vbroker.orb.alwaysTunnel	**false** – disabled **true** – enabled	Specifies whether the client must always make HTTP tunnel (IIOP wrapper) connections to the server. This property can be set in the client or GateKeeper. See the *VisiBroker Programmer's Reference* for more details.
vbroker.orb.dynamicLibs	**<empty>** <a list of libraries> See description	Specifies a list of libraries. Only libraries related to GateKeeper are described here. If the firewall component is specified, you must set this property in the properties of the client and server to: com.inprise.vbroker.firewall.Init If the load balancing component is specified, you must set this property in GateKeeper's property file to: com.inprise.vbroker.gatekeeper.ext.Init

Appendix

B

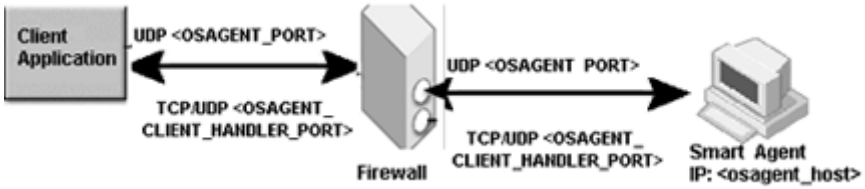
GateKeeper deployment scenarios

This appendix shows some common deployment scenarios in a multi-network environment with and without using GateKeeper.

TCP firewall (without GateKeeper)

Scenario 1.1: Smart Agent behind firewall

This scenario shows how to configure a client object to access a Smart Agent located behind a firewall.



Client's environment setting (using environment variables or Windows Registry):

```
OSAGENT_PORT
```

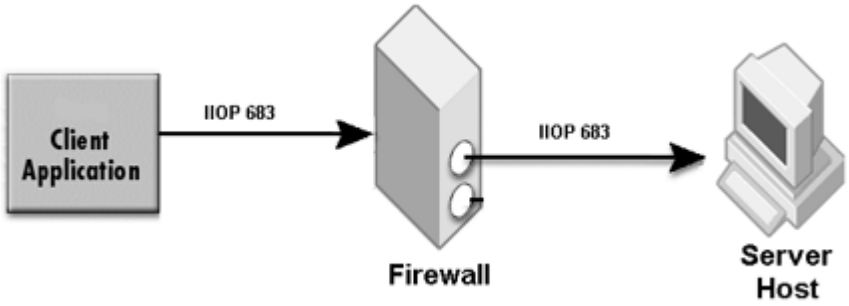
Client's properties:

```
vbroker.agent.addr=<osagent_host>
vbroker.agent.port=<OSAGENT_PORT>
```

Firewall settings:

- Allow UDP packets for both directions between the client host and the Smart Agent host on port <OSAGENT_PORT>.
- Allow TCP and UDP packets for both directions between the client host and the Smart Agent host on port <OSAGENT_CLIENT_HANDLER_PORT>.

Scenario 1.2: Using IIOP communication



Client's properties: none required

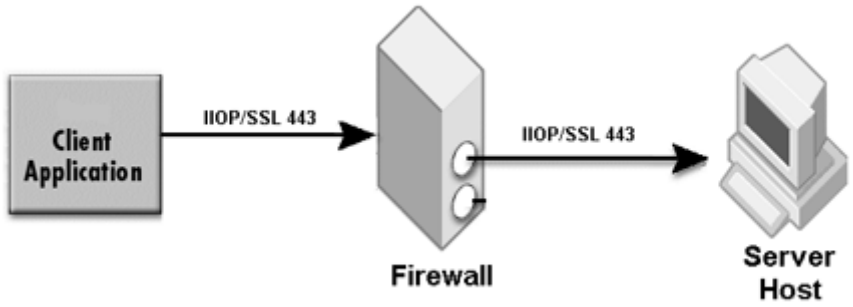
Server's properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
```

Firewall setting:

Allow TCP packet from client host to server host on port 683.

Scenario 1.3: Using IIOP/SSL communication



Secured Client's properties:

```
# Enabling Security Service
vbroker.security.disable=false

# Enforcing secure transport at client side
vbroker.security.alwaysSecure=true

# Setting peerAuthenticationMode
vbroker.security.peerAuthenticationMode=REQUIRE_AND_TRUST
vbroker.security.trustpointsRepository=Directory:./trustpoints
```

Secured Server's properties:

```
# Enabling Security Service
vbroker.security.disable=false

# Setting SSL Layer Attributes
vbroker.security.peerAuthenticationMode=REQUIRE_AND_TRUST
vbroker.security.trustpointsRepository=Directory:./trustpoints

# Set the ssl listener port at 443
vbroker.se.iiop_tp.scms=iiop_tp,ssl
vbroker.se.iiop_tp.scm.ssl.listener.port=443
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
```

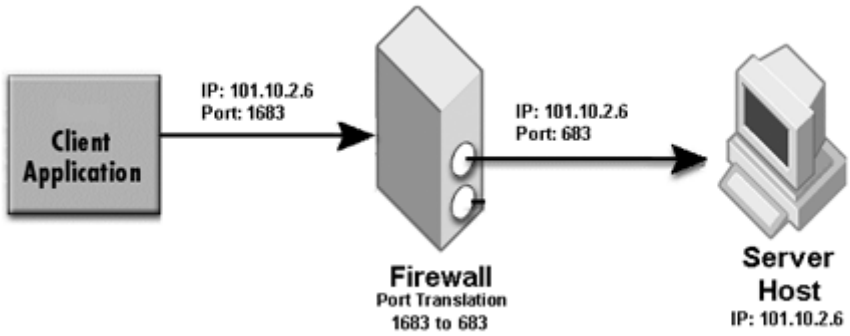
Note

The sample properties assume that valid certificate information has already been loaded into the Secured Client and Secured Server similar to the `<install_dir>/examples/vbroker/security/bank_ssl` example.

Firewall setting:

Allow SSL packet from client host to server host on port 443.

Scenario 1.4: Firewall performs address translation only



Firewall setting:

Address translation: 199.10.9.6 to 101.10.2.6

Server's properties: Use only one of the following two methods.

Method 1: Using IIOP profile

```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.proxyHost=199.10.9.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
```

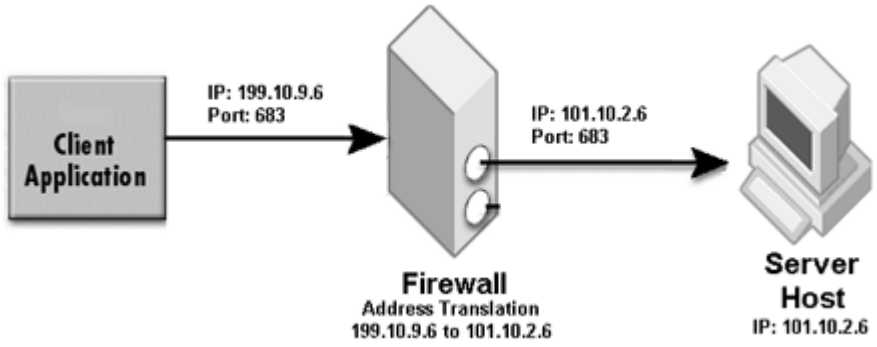
Method 2: Using firewall component

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=199.10.9.6
vbroker.firewall.fw.iiop_port=683
vbroker.firewall.fw.hiop_port=0
```

Note

Specify real port when there is no port translation, and 0 if the listener port is disabled.

Scenario 1.5: Firewall performs port translation only



Firewall setting:

Port translation: 1683 to 683

Server's properties: Use only one of the following two methods.

Method 1: Using IIOP profile

```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=1683
```

Method 2: Using firewall component

```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=101.10.2.6
vbroker.firewall.fw.iiop_port=1683
vbroker.firewall.fw.hiop_port=0
```

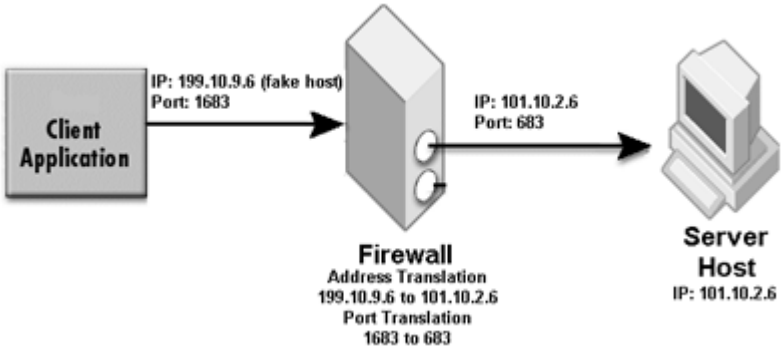
Note

Specify real host when there is no address translation.

When method 2 is used, add the following to the client's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
```

Scenario 1.6: Firewall performs both address and port translations



Combine the settings in Scenarios 1.4 and 1.5 when the firewall performs both address and port translation.

Note

For firewall component method, specify the firewall once combining both the fake host and fake port into the same firewall entry like the following:

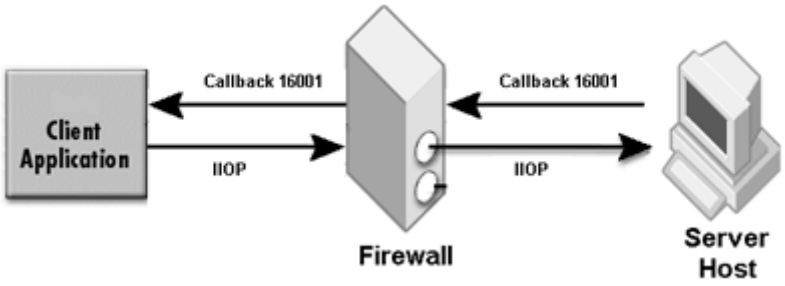
```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=199.10.9.6
vbroker.firewall.fw.iiop_port=1683
vbroker.firewall.fw.hiop_port=0
```

Note

For secure connection with NAT (Network Address Translation), use the security properties settings in Scenario 1.3.

Scenario 1.7: Callback without NAT

Refer to Scenario 1.2 for forward communication settings.



Client's properties:

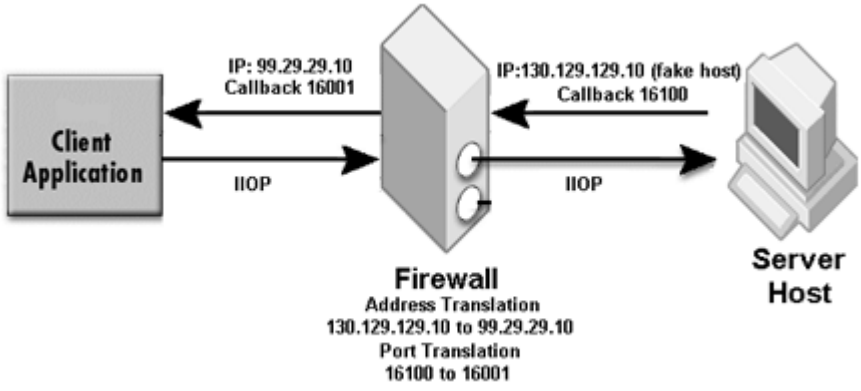
```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=16001
```

Firewall setting:

Allow TCP packet from server host to client host on port 16001.

Scenario 1.8: Callback with NAT

Refer to Scenario 1.2 for forward communication settings.



Firewall setting:

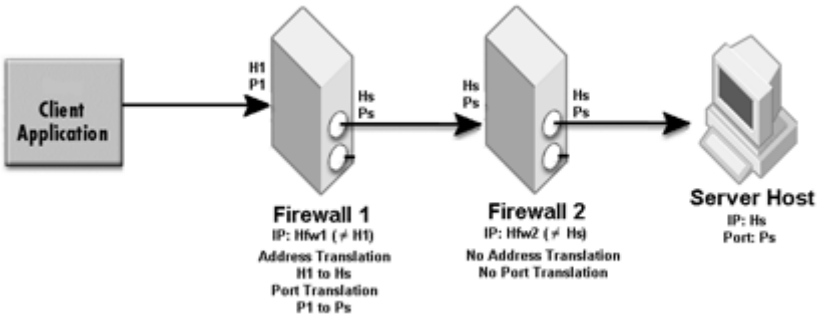
Address translation: 130.129.129.10 to 99.29.29.10 for packets from server network to client network. Port translation: 16100 to 16001 for packets from server network to client network.

Client's properties:

```
vbroker.se.iiop_tp.host=99.29.29.10
vbroker.se.iiop_tp.proxyHost=130.129.129.10
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=16001
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=16100
```

Scenario 1.9: Bidirectional communication

Use the settings in Scenario 1.2, 1.3, 1.4, 1.5, or 1.6 with the following additional settings to enable bidirectional communication.



In the figure above, the same connections are used for both forward and reverse communications paths.

Client's Properties:

```
vbroker.orb.enableBiDir=client
```

Server's Properties:

```
vbroker.orb.enableBiDir=server
```

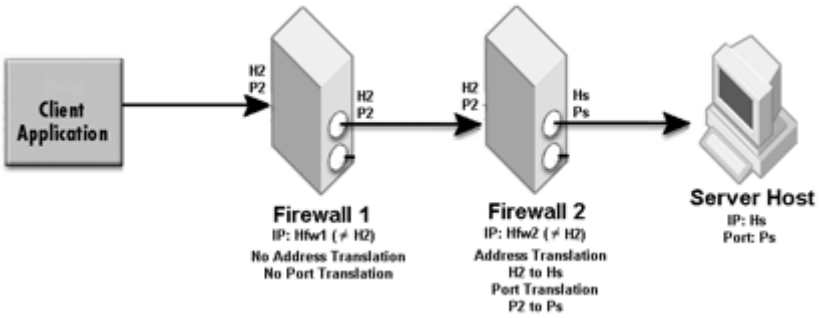
Scenario 1.10: Multiple firewalls in front of server

This scenario shows two firewalls in front of the server host. It can be extended similarly to more than two firewalls.

Both firewalls do not perform NAT

When both firewalls do not perform NAT, configure both the firewalls to allow TCP packets (for IIOP communication) on port Ps.

Only Firewall 1 performs NAT



Firewall performs the following NAT:

- Address translation: H1 to HsPort translation: P1 to Ps
- Firewall 2 must be configured to allow TCP packets on port Ps.
- Clients will send IIOP packets to host H1 on port P1.

Server's properties: Use only one of the following two methods.

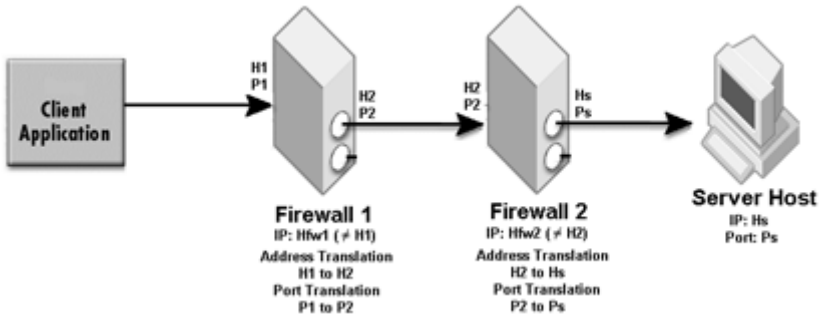
Method 1: Using IIOP profile

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.proxyHost=<H1>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<P1>
```

Method 2: Using firewall component

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw1
vbroker.firewall.fw1.type=TCP
vbroker.firewall.fw1.host=<H1>
vbroker.firewall.fw1.iiop_port=<P1>
vbroker.firewall.fw1.hiop_port=0
```

Only firewall2 performs NAT



Firewall2 performs the following NAT:

- Address translation: H2 to HsPort translation: P2 to Ps
- Firewall1 must be configured to allow TCP packets on port P2.
- Clients will send IIOP packets to host H2 on port P2.

Server's properties: Use only one of the following two methods.

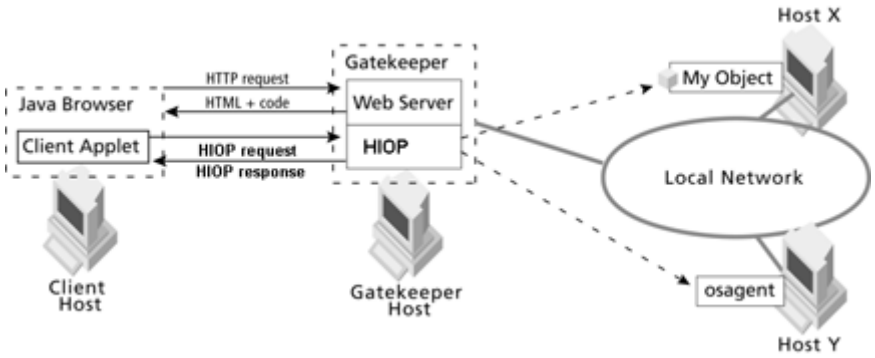
Method 1: Using IIOP profile

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.proxyHost=<H2>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<P2>
```

Method 2: Using firewall component

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw2
vbroker.firewall.fw2.type=TCP
vbroker.firewall.fw2.host=<H2>
vbroker.firewall.fw2.iiop_port=<P2>
vbroker.firewall.fw2.hiop_port=0
```

Both firewalls perform NAT



Firewall1 performs the following NAT:

- Address translation: H1 to H2Port translation: P1 to P2
- Firewall2 performs the following NAT:
- Address translation: H2 to HsPort translation: P2 to Ps
- Clients will send IIOP packets to host H1 on port P1.

Server's properties: Use only one of the following two methods.

Method 1: Using IIOP profile

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.proxyHost=<H1>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<P1>
```

Method 2: Using firewall component

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw1
vbroker.firewall.fw1.type=TCP
vbroker.firewall.fw1.host=<H1>
vbroker.firewall.fw1.iiop_port=<P1>
vbroker.firewall.fw1.hiop_port=0
```

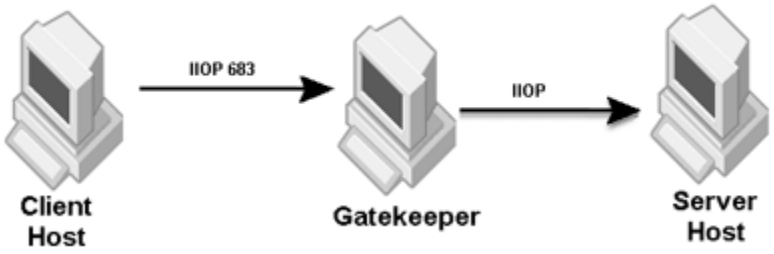
Note

The NAT information of Firewall2 does not need to be configured. The `proxyHost` and `proxyPort` specify only the first NAT fake host and fake port. For the firewall component and the firewall path, only specify the first NAT device.

GateKeeper deployment

Scenario 2.1: GateKeeper as Web Server

GateKeeper can acts as a Web Server to serve HTML pages, client applet and IOR files.



Set the GateKeeper HTTP listener using the following GateKeeper's properties:

```
vbroker.se.exterior.scms=ex-iiop,ex-hiop
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

From the web browser of the client host,

- Use the following to load an HTML file or client applet:

  ```
  http://gatekeeper:8088/ClientApplet.html
  ```

- Use the following to load GateKeeper's IOR:

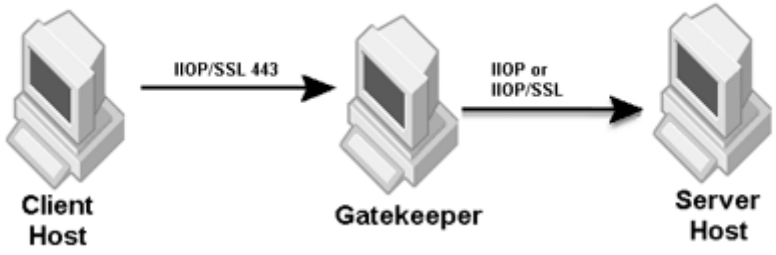
  ```
  http://gatekeeper:8088/gatekeeper.ior
  ```

Configure the client applet (ClientApplet.html) using the following example:

```
<applet archive=vbjorb.jar code="ClientApplet.class" width="200" height="80">
<param name="org.omg.CORBA.ORBClass" value="com.inprise.vbroker.orb.ORB">
<param name="vbroker.orb.alwaysTunnel" value="true">
<param name="vbroker.orb.gatekeeper.ior" value="http://gatekeeper:8088/
gatekeeper.ior">
</applet>
```

Any additional client properties needed can be set similarly using param name and value.

Scenario 2.2: GateKeeper as IIOP Proxy



Client's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.alwaysProxy=true
```

GateKeeper's properties:

```
vbroker.se.exterior.scm.ex-iiop.listener.port=683
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

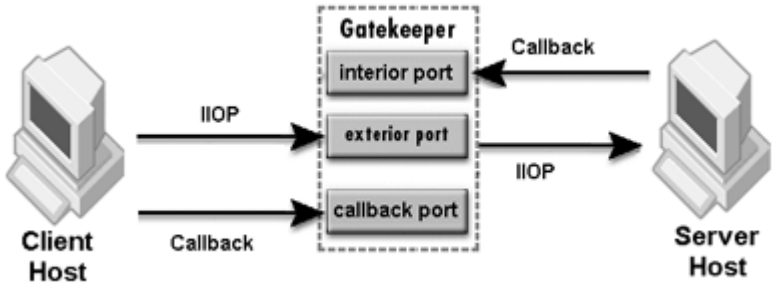
Server's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior
```

If the client is an applet that wants to use IIOP instead of HTTP Tunneling, use the following configuration, do not specify the property

```
<param name="vbroker.orb.alwaysTunnel" value="true">:
<applet archive=vbjorb.jar code="ClientApplet.class" width="200" height="80">
<param name="org.omg.CORBA.ORBClass" value="com.inprise.vbroker.orb.ORB">
<param name="vbroker.orb.alwaysProxy" value="true">
<param name="vbroker.orb.gatekeeper.ior" value="http://gatekeeper:8088/
gatekeeper.ior">
</applet>
```

Scenario 2.3: HTTP Tunneling Connection



Client's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init,com.inprise.vbroker.H
IOP.Init
vbroker.orb.alwaysTunnel=true
vbroker.orb.alwaysProxy=true
vbroker.orb.gatekeeper.ior=http://gatekeeper:8088/gatekeeper.ior
```

GateKeeper's properties:

```
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
vbroker.se.exterior.scm.ex-iiop.listener.port=683
```

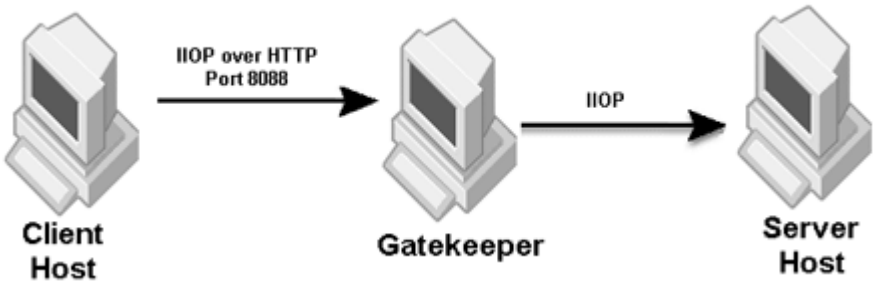
Server's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior
vbroker.orb.exportFirewallPath=true
```

If the client is an applet that wants to use HTTP Tunneling, use the following configuration:

```
<applet archive=vbjorb.jar code="ClientApplet.class" width="200" height="80">
<param name="org.omg.CORBA.ORBClass" value="com.inprise.vbroker.orb.ORB">
<param name="vbroker.orb.alwaysTunnel" value="true">
<param name="vbroker.orb.gatekeeper.ior" value="http://gatekeeper:8088/
gatekeeper.ior">
</applet>
```

Scenario 2.4: Secure connection (SSL)



Client's properties:

```
# Firewall related properties
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.alwaysProxy=true

# Set SSL related properties
vbroker.security.disable=false
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity= paul
vbroker.security.wallet.password= Paul$$$$
vbroker.security.trustpointsRepository=Directory:./trustpoints
```

GateKeeper's properties:

```
vbroker.se.exterior.scms=ex-iiop,ex-ssl
vbroker.se.exterior.scms.ex-iiop.listener.type=Disabled-IIOP
vbroker.se.exterior.scms.ex-hiop.listener.port=8088
vbroker.se.exterior.scms.ex-ssl.listener.port=443

# Set SSL related properties
vbroker.security.disable=false
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity= kevin
vbroker.security.wallet.password= Kevin$$$$
vbroker.security.trustpointsRepository=Directory:./trustpoints
vbroker.se.iiop_tp.scm.ssl.listener.port=<server SSL: listener port>
```

Server's properties:

```
# Firewall related properties
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior

# SSL related properties
vbroker.security.disable=false
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity= kevin
```

```
vbroker.security.wallet.password= Kevin$$$
vbroker.security.trustpointsRepository=Directory:./trustpoints

vbroker.se.iiop_tp.scms=iiop_tp,ssl
vbroker.se.iiop_tp.scm.ssl.listener.port=<server SSL listener port>
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
```

Scenario 2.5: Secure HTTP Tunneling

Use the client and server settings in Scenario 2.4 and add the following to the client's properties:

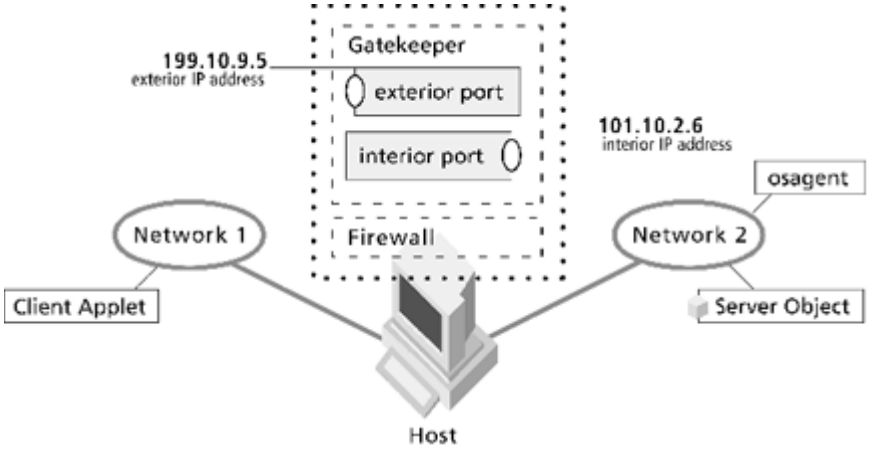
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init,
                        com.inprise.vbroker.HIOP.Init
                        com.inprise.security.Init,
                        com.inprise.security.hiops.Init
vbroker.orb.alwaysTunnel=true
```

GateKeeper's properties:

```
vbroker.orb.dynamicLibs=com.inprise.security.Init,
                        com.inprise.vbroker.gatekeeper.ssl.Init,
                        com.inprise.security.hiops.Init
vbroker.se.exterior.scms=ex-IIOP,ex-hiop,ex-hiops
vbroker.se.exterior.scm.ex-iiop.listener.type=Disabled-IIOP
vbroker.se.exterior.scm.ex-hiops.listener.port=443
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity=Kevin
vbroker.security.wallet.password=Kevin$$$
vbroker.security.secureTransport=true
vbroker.security.trustpointsRepository=Directory:./trustpoints
vbroker.security.peerAuthenticationMode=none
```

Scenario 2.6: Callback connection (for VisiBroker 3.x style)

Refer to Scenario 2.2 for forward communication settings.



Set the following client's property:

```
vbroker.se.iiop_tp.scm.iiop_tp.type=Callback-IIOP
vbroker.se.iiop_tp.scm.iiop_tp.listener.gatekeeper=http://gk_host:8088/
gatekeeper.ior
```

Enable GateKeeper callback (VisiBroker 3.x style) using the following GateKeeper's properties:

```
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
vbroker.gatekeeper.backcompat.callback.listeners=iiop
vbroker.gatekeeper.backcompat.callback.listener.iiop.port=<exterior callback
port>
vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
```

The interior port `in-iiop` is automatically enabled when callback is enabled. Only for secured callback, you need to add the SCM for `in-ssl`, `ex-ssl and ex-hiops as required`.

Scenario 2.7: Bidirectional communication

Use the settings in Scenario 2.2, 2.3, 2.4, or 2.5 with the following additional settings to enable bidirectional communication.

Client's Properties:

```
vbroker.orb.enableBiDir=client
```

Server's Properties:

```
vbroker.orb.enableBiDir=server
```

GateKeeper's Properties:

```
vbroker.orb.enableBiDir=both
```

Scenario 2.8: Pass-through connection

Use the settings in Scenario 2.2 or 2.4 with the following additional settings to enable pass-through connection.

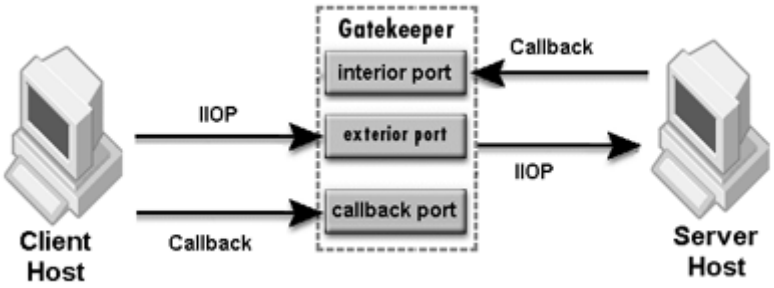
Client's Properties:

```
vbroker.orb.proxyPassthru=true
```

GateKeeper's Properties:

```
vbroker.gatekeeper.enablePassthru=true
```

Scenario 2.9: GateKeeper in dual-homed host configuration



Use the following GateKeeper's properties to configure:

- exterior host and interior host address

```
vbroker.se.exterior.host=199.10.9.5
vbroker.se.interior.host=101.10.2.6
```

- exterior listener ports

```
vbroker.se.exterior.scm.ex-iiop.listener.port=<exterior IIOP port>
vbroker.se.exterior.scm.ex-hiop.listener.port=<exterior HIOP port>
```

- interior listener ports (used for VisiBroker 3.x style callback)

```
vbroker.se.interior.scm.in-iiop.listener.port=<interior IIOP port>
```

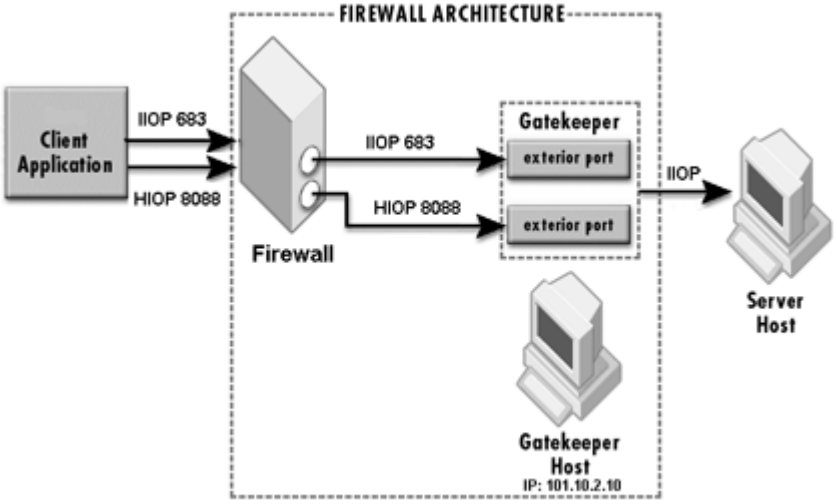
GateKeeper with server-side firewall

Note

Routers can also perform the function of a firewall.

Firewall in front of GateKeeper

Scenario 3.1: Firewall performs packet-filtering without NAT



GateKeeper's properties:

```
vbroker.se.exterior.scm.ex-iiop.listener.port=683
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

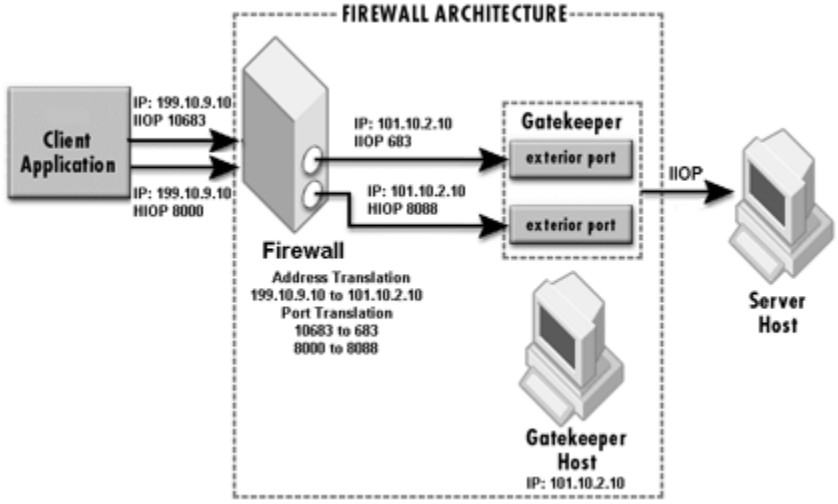
Firewall setting:

Allow routing of TCP packets on port 683 and HTTP packets on port 8088 from the external network to the internal network.

Server's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://101.10.2.10:8088/gatekeeper.ior
```

Scenario 3.2: Firewall performs NAT



Firewall NAT setting:

Address translation: 199.10.9.10 to 101.10.2.10 Port translations: 10683 to 683 and 8000 to 8088

There are two methods for specifying a NAT on a firewall in front of GateKeeper (use only one of the following two methods):

- Using GateKeeper's `proxyHost` and `proxyPort` configuration

  GateKeeper's properties:

  ```
  vbroker.se.exterior.host=101.10.2.10
  vbroker.se.exterior.proxyHost=199.10.9.10
  vbroker.se.exterior.scm.ex-iiop.listener.port=683
  vbroker.se.exterior.scm.ex-iiop.listener.proxyPort=10683
  vbroker.se.exterior.scm.ex-hiop.listener.port=8088
  vbroker.se.exterior.scm.ex-hiop.listener.proxyPort=8000
  ```

  Server's properties:

  ```
  vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
  vbroker.se.iiop_tp.firewallPaths=p
  vbroker.firewall-path.p=gk
  vbroker.firewall.gk.type=PROXY
  vbroker.firewall.gk.ior=http://101.10.2.10:8088/gatekeeper.ior
  ```

- Using the server's firewall component

  GateKeeper's properties:

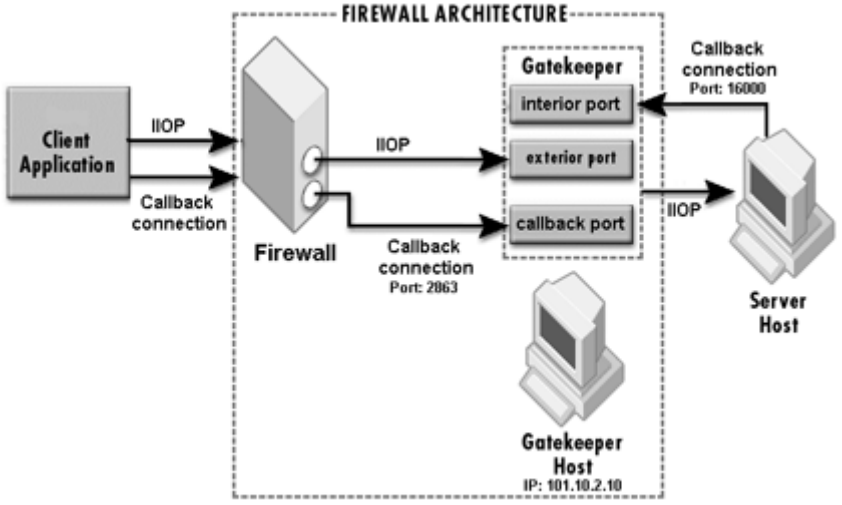
  ```
  vbroker.se.exterior.host=101.10.2.10
  vbroker.se.exterior.scm.ex-iiop.listener.port=683
  vbroker.se.exterior.scm.ex-hiop.listener.port=8088
  ```

Server's properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw,gk
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=199.10.9.10
vbroker.firewall.fw.iiop_port=10683
vbroker.firewall.fw.hiop_port=8000
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://101.10.2.10:8088/gatekeeper.ior
```

Scenario 3.3: Callback connection (for VisiBroker 3.x style)

Refer to Scenario 3.1 or 3.2 for forward communication settings.



Set the following client's property:

```
vbroker.se.iiop_tp.scm.iiop_tp.type=Callback-IIOP
vbroker.se.iiop_tp.scm.iiop.listener.gatekeeper=http://gk_host:8088/
gatekeeper.ior
```

Enable GateKeeper callback (VisiBroker 3.x style) and specify the callback port using the following properties:

```
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
vbroker.gatekeeper.backcompat.callback.host=101.10.2.10
vbroker.gatekeeper.backcompat.callback.listeners=iiop
vbroker.gatekeeper.backcompat.callback.listener.iiop.port=2683
vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
```

The firewall must allow the client to establish a callback connection (TCP protocol) to GateKeeper using port 2683.

Configure the interior ports using the following GateKeeper's properties:

```
vbroker.se.interior.scm.in-iiop.listener.port=16000
```

If the firewall performs NAT on the GateKeeper's host and callback port (address translation:199.10.9.10 to 101.10.2.10 and port translation: 12683 to 2683), add the following into GateKeeper's properties:

```
vbroker.gatekeeper.backcompat.callback.proxyHost=199.10.9.10
vbroker.gatekeeper.backcompat.callback.listener.iiop.proxyPort=12683
```

Scenario 3.4: Bidirectional communication

Use the settings in Scenario 3.1 or 3.2 with the following additional settings to enable bidirectional communication.

Client's Properties:

```
vbroker.orb.enableBiDir=client
```

Server's Properties:

```
vbroker.orb.enableBiDir=server
```

GateKeeper's Properties:

```
vbroker.orb.enableBiDir=both
```

Scenario 3.5: Pass-through connection

Use the settings in Scenario 3.1 or 3.2 with the following additional settings to enable pass-through connection.

Client's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.proxyPassthru=true
```

GateKeeper's Properties:

```
vbroker.gatekeeper.enablePassthru=true
vbroker.gatekeeper.passthru.inPortMin=<in_min_port>
vbroker.gatekeeper.passthru.inPortMax=<in_max_port>
```

Firewall setting:

Caution

Allow routing of TCP packets on port range `<in_min_port>` to `<in_max_port>` from the client host to the GateKeeper. The firewall must not perform port translation on this range of ports.

Firewall in front and behind of GateKeeper

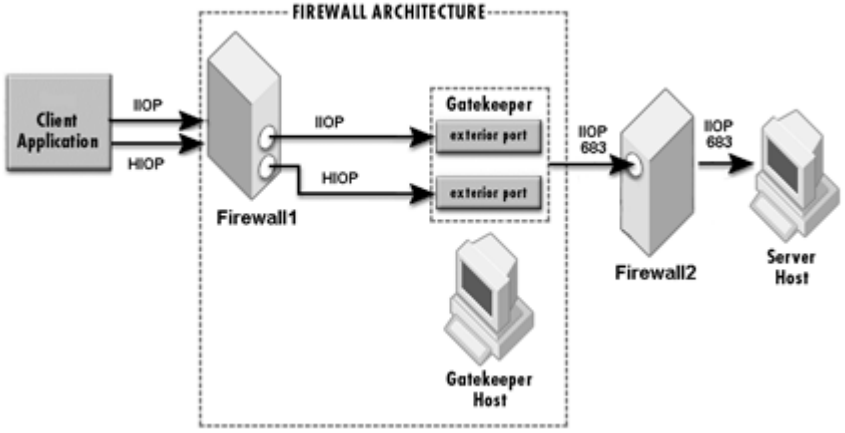
GateKeeper is deployed in the Demilitarized Zone (DMZ) while the servers are deployed in the internal network.

Note

Refer to the previous section for a configuration related to a firewall in front of GateKeeper. This section concentrates on the configuration related to the firewall between the GateKeeper and servers.

Scenario 4.1: Configuring firewall behind GateKeeper

Use the settings in Scenario 3.1 or 3.2 to configure the communication between clients and GateKeeper. The settings described here should be used in conjunction to the settings in Scenario 3.1 or 3.2.



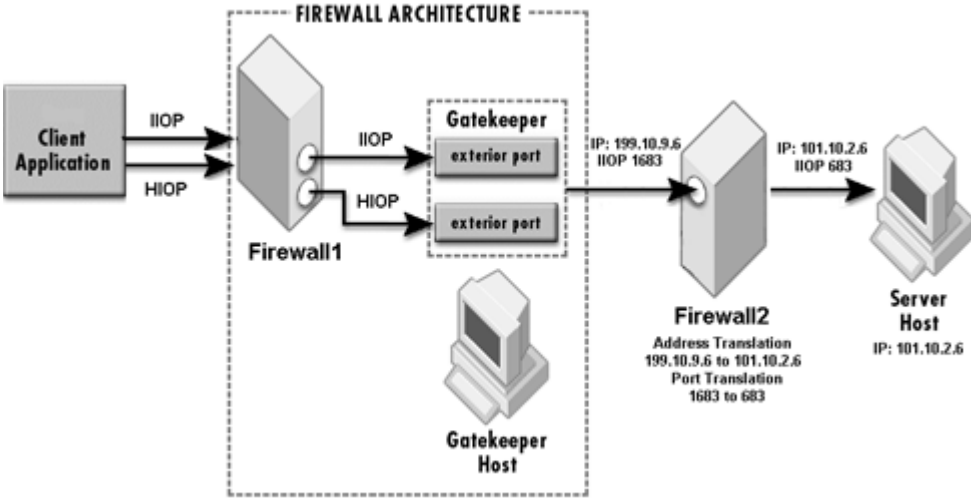
Specify the server IIOP listener port using the following server's properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
```

Configure Firewall2 to allow IIOP packet (TCP protocol) from GateKeeper to the server host on port 683.

Scenario 4.2: Firewall behind GateKeeper performs NAT

Use the settings in Scenario 3.1 or 3.2 to configure the communication between clients and GateKeeper. The settings described here should be used in conjunction to the settings in Scenario 3.1 or 3.2.



Firewall2 NAT setting:

Address translation: 199.10.9.10 to 101.10.2.10
Ports translation: 1683 to 683

There are two methods for specifying NAT on Firewall2. Use only one of the following two methods.

Server's properties:

Method 1: Using IIOP profile

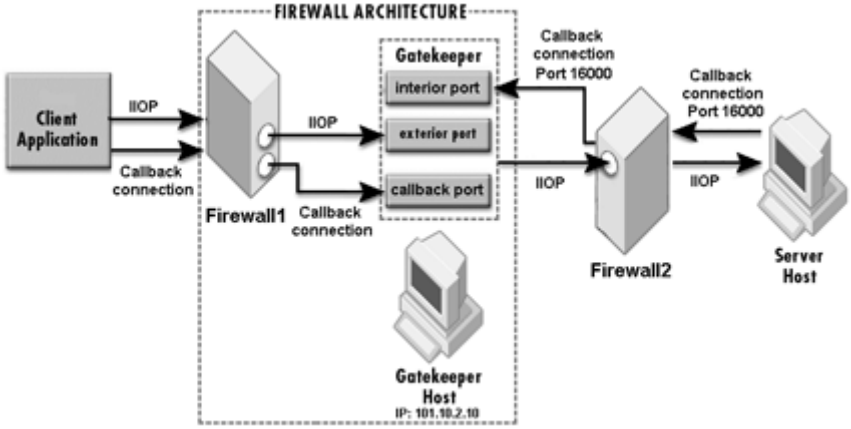
```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.proxyHost=199.10.9.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=1683
```

Method 2: Using firewall component

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk,fw2
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior
vbroker.firewall.fw2.type=TCP
vbroker.firewall.fw2.host=199.10.9.6
vbroker.firewall.fw2.iiop_port=1683
vbroker.firewall.fw2.hiop_port=0
```

Scenario 4.3: Callback connection (for VisiBroker 3.x style)

Use the settings in Scenario 3.3 callback connection between client and GateKeeper.



Configure the interior ports using the following GateKeeper's properties:

```
vbroker.se.interior.scm.in-iiop.listener.port=16000
```

Firewall2 must allow the server to communicate with GateKeeper on port 16000 using TCP protocol.

If Firewall2 performs the following NAT for packets routed from the server to GateKeeper:

Address Translation: 121.100.2.19 to 101.10.2.10Port Translation: 161000 to 16000

Then add the following properties to the GateKeeper's properties:

```
vbroker.se.interior.host=101.10.2.10
vbroker.se.interior.proxyHost=121.100.2.19
vbroker.se.interior.scm.in-iiop.listener.port=16000
vbroker.se.interior.scm.in-iiop.listener.proxyPort=16100
```

Scenario 4.4: Bidirectional communication

Use the settings in Scenario 4.1 or 4.2 with the following additional settings to enable bidirectional communication.

Client's Properties:

```
vbroker.orb.enableBiDir=client
```

Server's Properties:

```
vbroker.orb.enableBiDir=server
```

GateKeeper's Properties:

```
vbroker.orb.enableBiDir=both
```

Scenario 4.5: Pass-through connection

Use the settings in Scenario 4.1 or 4.2 with the following additional settings to enable pass-through connection.

Client's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.proxyPassthru=true
```

GateKeeper's Properties:

```
vbroker.gatekeeper.enablePassthru=true
vbroker.gatekeeper.passthru.inPortMin=<in_min_port>
vbroker.gatekeeper.passthru.inPortMax=<in_max_port>
vbroker.gatekeeper.passthru.outPortMin=<out_min_port>
vbroker.gatekeeper.passthru.outPortMax=<out_max_port>
```

Server's Properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<server IIOP port>
```

Caution

The value of `<server IIOP port>` must fall in the range of `<out_min_port>` and `<out_max_port>`.

Configure Firewall1 to allow routing of TCP packets in port range `<in_min_port>` to `<in_max_port>` from the client host to the GateKeeper. Configure Firewall2 to allow routing of TCP packets in port range `<out_min_port>` to `<out_max_port>` from the GateKeeper to the server host. The firewalls must not perform port translation on these ports.

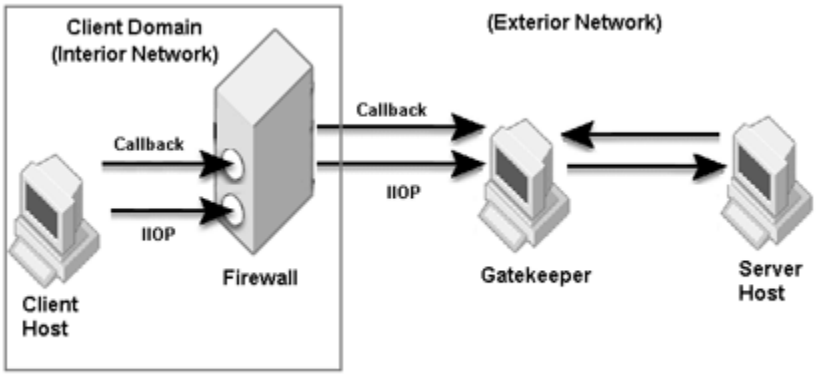
Scenario 4.6: Smart Agent in internal network

Use the settings in Scenario 1.1 assuming GateKeeper is the client.

GateKeeper with client-side firewall

Scenario 5.1: Firewall allows IIOP

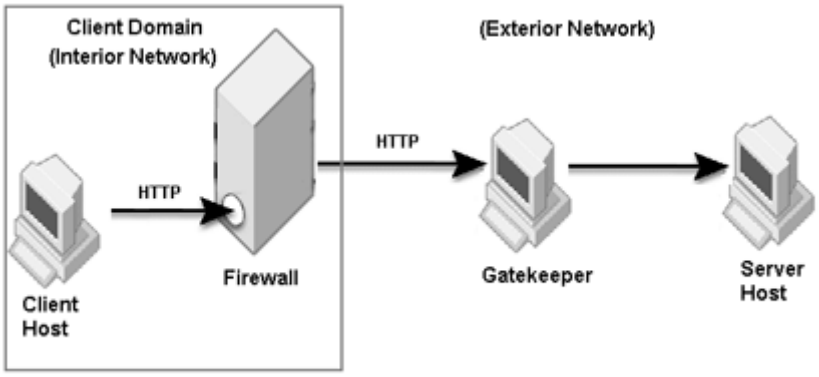
The client-side firewall allows clients from the internal network to send IIOP messages (TCP protocol) to the external network.



Refer to Scenarios 3.x replacing the server-side firewall in front of GateKeeper with a client-side firewall. As GateKeeper is outside the client domain, the client-side administrator who control the firewall, usually does not have the authority to modify the GateKeeper's configuration. The administrator has to collect the GateKeeper's listener ports information to configure the client-side firewall accordingly.

Scenario 5.2: Firewall allows HTTP only

The client-side firewall allows clients from the internal network to send HTTP messages only to the external network. IIOP message will be blocked by the firewall. Therefore, clients have to use HTTP tunneling to communicate with a GateKeeper outside the client-side firewall.



Set the following client's property to force the client to always use HTTP tunneling.

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.alwaysTunnel=true
```

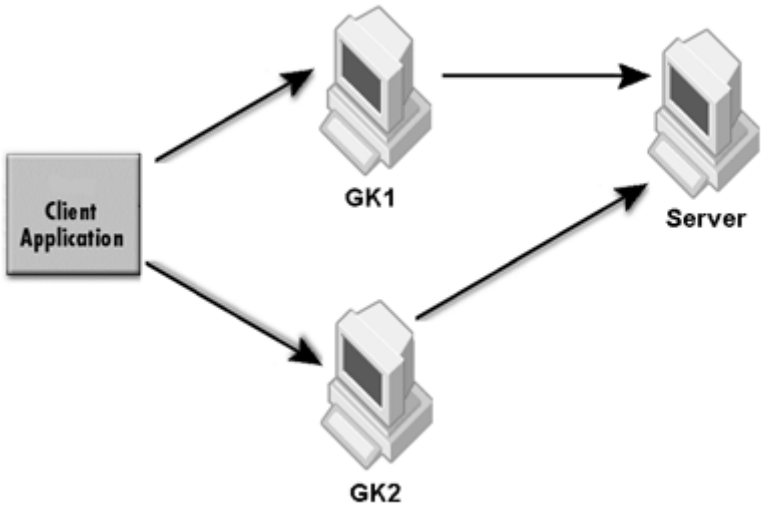
Note

HTTP tunneling does not support the VisiBroker 3.x style callback. If callback is required, use a bidirectional connection. Pass-through connection is also not available with HTTP tunneling.

GateKeeper load balancing and fault-tolerance

Scenario 6.1: Using multiple GateKeepers for fault-tolerance

Instead of relying on a single GateKeeper, you can deploy multiple GateKeepers for fault-tolerance. Assign more than one GateKeepers to a server to create redundancy.



The server's properties in this example are as follows:

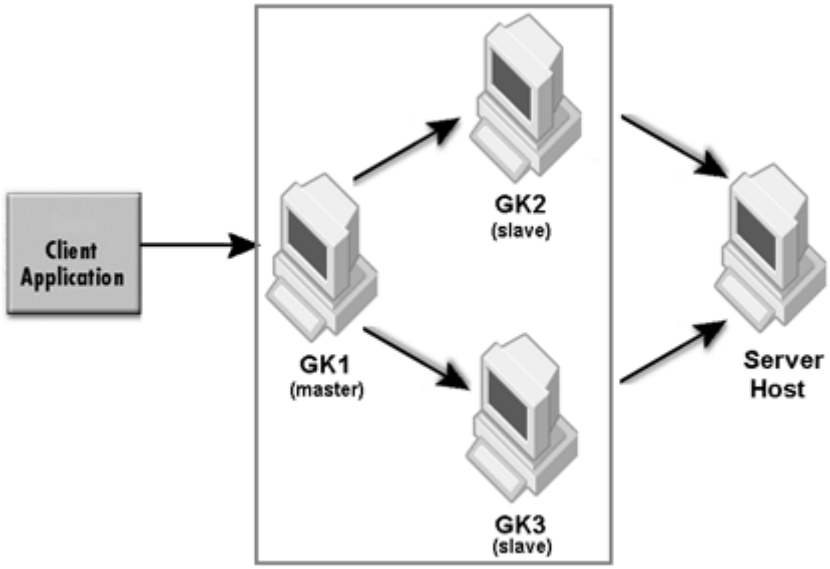
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p1,p2
vbroker.firewall-path.p1=gk1
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
vbroker.firewall-path.p2=gk2
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

The following property is required for both GK1 and GK2:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
```

Clients can use either GK1 or GK2 to communicate with the server. When one GateKeeper is down, the client can use the other one to communicate with the server.

Scenario 6.2: Master/Slave configuration for load balancing



The figure above shows a master/slave GateKeeper configuration with GK1 as the master GateKeeper while GK2 and GK3 are the slave GateKeepers

GateKeeper GK1 properties (master):

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
vbroker.gatekeeper.load.slaves=gk2,gk3
vbroker.gatekeeper.load.slave.gk2=http://gk2_host:8088/gatekeeper.ior
vbroker.gatekeeper.load.slave.gk3=http://gk3_host:8088/gatekeeper.ior
```

No additional properties are required for slave GateKeepers GK1 and GK2:

Server's Properties (specify only the master GateKeeper):

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk1
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
```

If the client is not able to obtain the server's IOR directly, then the client can specify a GateKeeper to contact using the following property:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.gatekeeper.ior=http://gk1_host:8088/gatekeeper.ior
```

This configuration also provides fault-tolerance. For each connecting client, the master GateKeeper assigns the next slave GateKeeper in turn to serve the client, but if that slave GateKeeper is down, the client will come back to the master GateKeeper to get assigned to the next available slave GateKeeper, and so on, until the client obtains a usable GateKeeper.

The master GateKeeper actually keeps a list of available GateKeepers, which it can assign to a connecting client. The list contains all slave GateKeepers as well as the master GateKeeper itself. Therefore, when its turn comes around, the master GateKeeper will assign itself to a client.

When the following property is set on the master GateKeeper, the master GateKeeper is not included in the list.

```
vbroker.gatekeeper.load.balancer=master
```

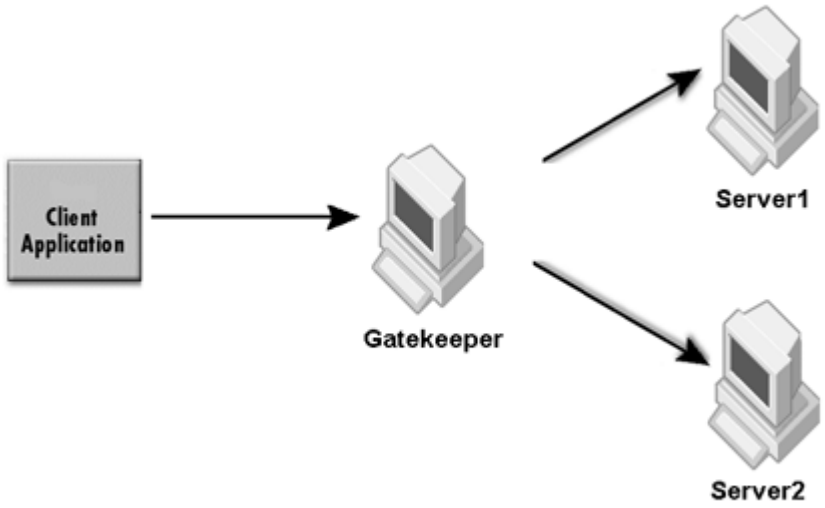
In the case that all slave GateKeepers are down, in order to prevent clients from coming back to the master GateKeeper for obtaining a usable GateKeeper again and again infinitely, the following property should be set on the client side:

```
vbroker.orb.rebindForward=N
```

where *N* must be less than the number of slave GateKeepers.

When the master GateKeeper itself is down, the rebind mechanism on the client ORB will make all clients connect through the first available slave GateKeeper, there will be no load balancing in this situation as the load balance functionality is in the master GateKeeper and the master is down. However, fault tolerance is preserved because clients still can get through and connect to the servers.

Scenario 6.3: Multiple instances of same server for load balancing



You can deploy multiple instances of the same server to provide load balancing and fault-tolerance for the server. For load balancing, the GateKeeper will direct the request to the multiple servers using a round-robin mechanism. For fault-tolerance, if one server is down, another server can continue to provide the same service.

Add the following property to the GateKeeper:

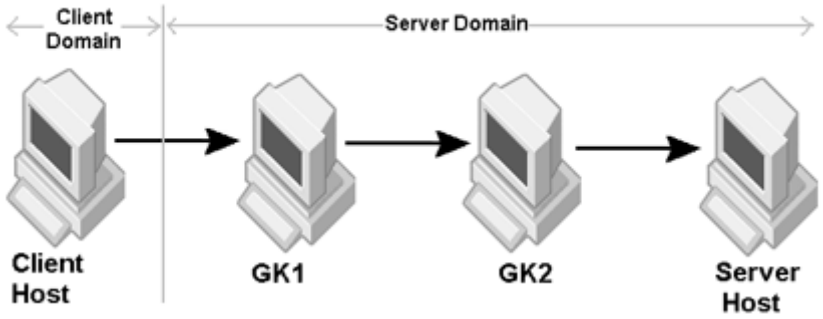
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
```

Server1 and Server2 properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gk_host:8088/gatekeeper.ior
```

GateKeeper chaining

Scenario 7.1: Server-side chaining

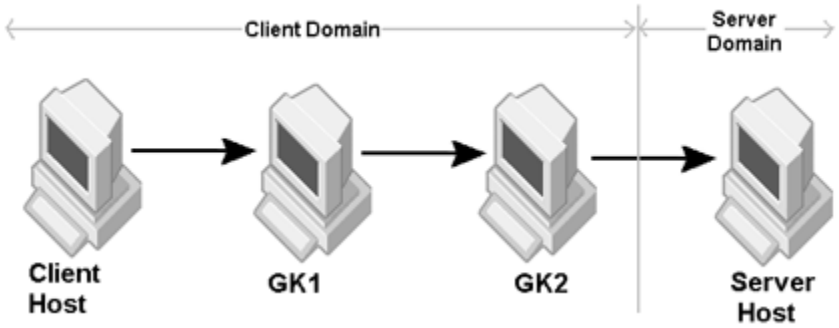


Use the following server's properties to specify the server-side GateKeeper chaining:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk1,gk2
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

When the client obtains the server's IOR, it will be able to use the GateKeeper chaining to communicate with the server.

Scenario 7.2: Client-side chaining



Client's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.gatekeeper.ior=http://GK1:8088/gatekeeper.ior
```

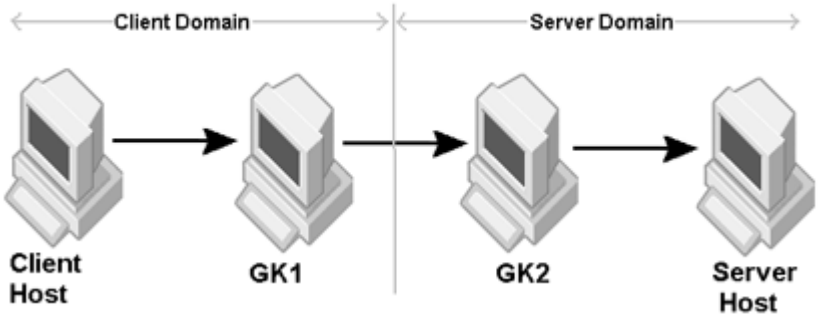
GK1's Properties:

```
vbroker.orb.gatekeeper.ior=http://GK2:8088/gatekeeper.ior
```

Note

In order for the client to communicate with the server using the chained GateKeepers, the last GateKeeper on the chain (GK2) must be able to obtain the server's IOR.

Scenario 7.3: Both server-side and client-side chaining



Client's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.gatekeeper.ior=http://gk_host:8088/gatekeeper.ior
```

Server's Properties:

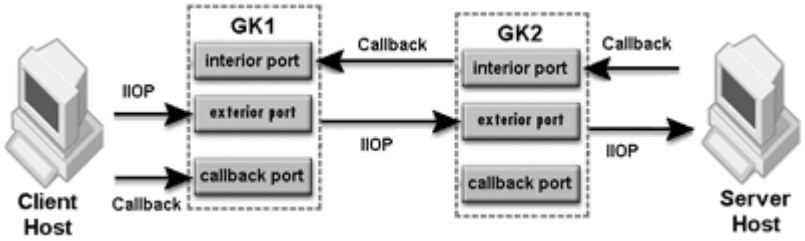
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk2
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

If GK1 always connect to GK2, you can chain GK1 to GK2 statically using the following GK1 property:

```
vbroker.orb.gatekeeper.ior=http://gk2_host:8088/gatekeeper.ior
```

Otherwise, GK1 must be able to obtain the IOR of the server or GK2 using a Smart Agent or Naming Services.

Scenario 7.4: Callback communication (VisiBroker 3.x style)



Set the following properties to allow VisiBroker 3.x style callback communication.

Client's Properties:

```
vbroker.orb.alwaysProxy=true
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Callback-IIOP
vbroker.se.iiop_tp.scm.iiop_tp.listener.gatekeeper=http://gk1_host:8088/
gatekeeper.ior
```

GK1 and GK2 properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
vbroker.gatekeeper.backcompat.callback.listeners=iiop
vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
vbroker.gatekeeper.backcompat.callback.listener.iiop.port=<exterior callback
port>
vbroker.gatekeeper.backcompat.callback.host=<GK exterior IP address>
vbroker.se.interior.scm.in-iiop.listener.port=<interior port>
```

Server's Properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<IIOP listener port>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.intranet=gk1,gk2
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

Scenario 7.5: Bi-directional connection

Refer to Scenario 7.1, 7.2, or 7.3.

Add the following additional settings to enable bi-directional communication.

Client's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.enableBiDir=client
```

Server's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.enableBiDir=server
```

GK1 and GK2 Properties:

```
vbroker.orb.enableBiDir=both
```

Scenario 7.6: Pass-through connection

Refer to the diagrams in Scenario 7.1, 7.2, or 7.3.

Client's Properties:

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.proxyPassthru=true
```

GK1 and GK2 Properties:

```
vbroker.gatekeeper.enablePassthru=true
vbroker.gatekeeper.passthru.inPortMin=<in_min_port>
vbroker.gatekeeper.passthru.inPortMax=<in_max_port>
vbroker.gatekeeper.passthru.outPortMin=<out_min_port>
vbroker.gatekeeper.passthru.outPortMax=<out_max_port>
```

Server's Properties:

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<server IIOP port>
```

Note

The value of `<server IIOP port>` must fall in the range of `<out_min_port>` and `<out_max_port> of GK2`. The range of `<in_min_port>` and `<in_max_port>` of GK2 must fall in the range of `<out_min_port>` and `<out_max_port> of GK1`.

If there is a firewall between any of the hosts, refer to the following table for the ports to be opened.

Firewall location	Range of open ports
Between Client and GK1	GK1 exterior IIOP and HIOP listener port, and GK1 <in_min_port> and <in_max_port>
Between GK1 and GK2	GK2 exterior IIOP and HIOP listener port, and GK2 <in_min_port> and <in_max_port>
Between GK2 and Server	GK2 <out_min_port> and <out_max_port>

Caution

The firewalls must not perform any port translation on the pass-through ports.

Using VisiBroker in a multiple firewall/subnet environment

VisiBroker can be used in multiple firewall scenarios. In general, VisiBroker provides two different approaches to cross firewalls.

First, the Network Address Translation (also called TCP firewalls) can be configured using the following properties:

```
vbroker.se.iiop_tp.host=www.realdomain.com
vbroker.se.iiop_tp.proxyHost=www.fakedomain.com
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=25000
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=32000
```

In the above-mentioned configuration, the real host / port information is lost in the IOR, which means only the fake host /port is available in the IOR. Another commonly deployed TCP firewall configuration is a server-side configuration. Since this configuration is an ORB built-in mechanism, it also applies to all types of services (for example GateKeeper, Naming Service, and so forth).

```
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths =Queen
vbroker.firewall-path.Queen=Atlantic
vbroker.firewall-path.Atlantic.type=TCP
vbroker.firewall-path.Atlantic.host=www.fakedomain.com
vbroker.firewall-path.Atlantic.iiop_port=32000
vbroker.firewall-path.Atlantic.hiop_port=32003
vbroker.firewall-path.Atlantic.ssl_port=32004
```

The advantage of the technique above is that the configuration information is not lost. The internal client can connect to the servers directly using the real IP host / port information. The risk involved with this configuration, however, is that both the IP host / port (real and fake) are exposed in the generated IOR file.

Second, GateKeeper can be run on the firewall server to act as a GIOP proxy server. Various mechanisms are available in GateKeeper that are designed for different purposes. For example:

- **Normal mode:** Used when the firewalls can allow at least one port for the GIOP Proxy Server (such as GateKeeper). It is an automatic mode which can switch to HTTP tunneling if required.
- **Pass through mode:** Used when the firewalls can allow a range of ports and packets exchanged between the client and server that is not to be interpreted by GateKeeper. In such a scenario, GateKeeper will act as a resource manager only. GateKeeper acts as a resource manager because it allocates IP ports to be used by the clients.

- **HTTP tunneling:** Used when the firewalls allow only HTTP traffic. In such a scenario, the GIOP Proxy Server cannot be run in the firewall. Instead, an HTTP proxy server sits before GateKeeper. The client ORB has a built in mechanism to convert GIOP messages into HTTP messages which will be sent to the HTTP proxy server or the firewall. Then, the HTTP proxy server (or the applicable firewall) forwards the HTTP messages to GateKeeper. Additionally, GateKeeper will convert the HTTP messages into GIOP messages and forwards it to the required target (such as server or another GateKeeper). This configuration is also useful in a client-side configuration when outgoing HTTP traffic is allowed by the firewall, but other types of TCP connections are not allowed.

Note

Multiple firewalls may need a combination of the above configuration techniques. Basically, the use of multiple firewalls is a deployment issue and, therefore, all possible combinations cannot be covered in this document. Some general guidelines are.

- The Smart Agent should be avoided because it is designed to work in a single domain only.
- CORBA Naming Service should be used to store and lookup CORBA object IOR.
- Domain Name Service (DNS) lookups should be avoided.
- Network Address Translation (NAT) or TCP Firewall configuration should be used only on the outermost firewall in the firewall enclaves. In such scenarios, even the internal clients are expected to behave as if they are outside the firewall.
- GateKeeper can be used wherever it is possible to run it in a firewall environment. HTTP Tunneling feature can be used if TCP connections are not allowed by the firewall.
- GateKeeper chaining can be used when multiple firewalls are involved. With GateKeeper chaining, multiple hops can be configured.
- Multiple GateKeepers should be used for load balancing and distribution.

Firewall and Smart Agent scenario

Within a firewall architecture, the Smart Agent is not expected to run on the firewall host. Instead, the Smart Agent can be run within the internal network. Usually, the Smart Agent should not be exposed to the external network because of security reasons. The Smart Agent uses an IPv4 UDP broadcast message to advertise itself. Since the firewall / routers can block broadcast messages from being forwarded to the next hop in the network, the Smart Agent is usually visible only in the local network. If the Smart Agent needs to be accessible from an external network, you must open specific a port on the firewall.

The following environment variables are used by the Smart Agent:

- `OSAGENT_PORT`
- `OSAGENT_CLIENT_HANDLER_PORT`

The VisiBroker-ORB requires the `OSAGENT_PORT` environment variable to be set to register and query CORBA objects using the Smart Agent. The default value of `OSAGENT_PORT` is 14000. By setting `OSAGENT_PORT` to an appropriate TCP/IP port, you can define a virtual domain. One can run any number of Smart Agents in a given subnet. Setting different `OSAGENT_PORT` values will create different domains, which means CORBA objects registered in one Smart Agent domain will not be visible to CORBA client querying from a different domain.

Set the following TCP/IP address or port to be used by CORBA application to reach the Smart Agent:

- `vbroker.agent.addr=143.186.142.21`
- `vbroker.agent.port=25873`

If the client, server, or GateKeeper does not require the use of the Smart Agent, set the following property in the respective property files to disable it:

```
vbroker.agent.enableLocator=false
```

GateKeeper is run on a multi-homed (or firewall) host. The Smart Agent can be run either on the multi-homed host on in the internal network. GateKeeper can be configured to use a designated Smart Agent using the following properties (for example):

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
```

The server in the internal network should register itself to the same Smart Agent as the GateKeeper if your client program expects GateKeeper to query the server objects to get the IOR using the following properties (for example).

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
```

GateKeeper can use only one Smart Agent domain at a time. The Smart Agent domain is determined by setting the `OSAGENT_PORT` value or the `vbroker.agent.port` property. All servers accessible through GateKeeper should register to the same Smart Agent domain or to the Naming Service. It is recommended to run the Smart Agent on the same subnet as GateKeeper in the internal network.

The following ports are required by the Smart Agent:

1. `OSAGENT_PORT` (UDP type)
2. `OSAGENT_CLIENT_HANDLER_PORT` (UDP type)
3. `OSAGENT_CLIENT_HANDLER_PORT` (TCP type)

The `OSAGENT_PORT` used by the ORB applications is a UDP port. The only TCP type of port (also known as the `OSAGENT_CLIENT_HANDLER_PORT`) used by the Smart Agent is assigned to the Location Service. The `OSAGENT_CLIENT_HANDLER_PORT` of the UDP type is used by the Smart Agent itself. Please note that `OSAGENT_CLIENT_HANDLER_PORT` should be set only on those hosts where the Smart Agent is running.

Using the Smart Agent in a firewall scenario

The Smart Agent has some built-in fail-over and load-balancing capabilities. The domain of the Smart Agent is defined by the `OSAGENT_PORT` in use. If an ORB application (such as a server) is registered to one of the Smart Agents in a given domain (the Smart Agent domain), other ORB applications (such as a client program) can query the server objects in that domain from any of Smart Agents in the same domain . Thus, Smart Agents can query within a domain to locate a server object without making the client application aware of the process. Therefore, if one Smart Agent fails, the ORB application can find another Smart Agent in the same domain, register itself again, and proceed.

By design, the load-balancing capability of Smart Agents is not extended to firewalls because each firewall has unique behavior. For example, a NAT (Network Address Translation) device is a type of firewall which changes the IP address/port. The Smart Agent is not designed to handle NAT scenarios. Also, some firewalls may allow only specific types of packets, others may require security and encryption, and some may not allow DNS lookups. Therefore, the Smart Agent should not be used in any kind of firewall or NAT configuration.

Although the Smart Agent is not designed to be used in a firewall scenario, there are few steps to follow if an application must access a Smart Agent sitting behind the firewall. The following steps, however, apply to inter-departmental firewalls only:

1. Open the `OSAGENT_PORT` and `OSAGENT_CLIENT_HANDLER_PORT` on the firewall. Certain firewalls may require that you set static forwarding routes so that the packets can reach the Smart Agent. All the intermediate firewalls between the applications should open these ports as well. Since firewalls may be on a multi-homed host, edit the localaddr (located, for example, in the `<instal_dir>var/defaults/adm/properties/services/osagentfile` folder) and set `OSAGENT_LOCAL_ADDR_FILE` to specify all the interfaces Smart Agent should bind to listen for request packets.

2 Set the Smart Agent IP address in the `agentaddr` file to allow the Smart Agent on one network to contact a Smart Agent on another network.

3 Set `OSAGENT_PORT` and `OSAGENT_CLIENT_HANDLER_PORT` on all hosts from where ORB applications may be launched. Please note that these ports should be the same as those opened on the firewalls.

Note

Even though it is possible to use the Smart Agent with the above settings, such usage is not recommend because this type of configuration may work with some firewalls, but will not work with all types of firewalls.

Behavior during the Smart Agent failure in a firewall scenario

If the Smart Agent fails, the ORB application is expected to switch to another Smart Agent in the same subnet. Because the Smart Agent `OSAGENT_PORT` is already fixed, the ORB application sends a UDP broadcast to locate another Smart Agent. If there is a firewall, the ORB application should have access to reach another host where the Smart Agent is running. The ORB application may not have the knowledge of the location of an alternative Smart Agent, so it can't do much. If the Smart Agent starts up again on the same host, the client may be able to contact it. Basically, it is important to understand that the Smart Agent uses a UDP broadcast-based technology. Some firewalls and routers may not forward UDP broadcasts and this is one of the reasons why the Smart Agent cannot be used across firewalls. Another Smart Agent, however, can be used if a Smart Agent in the same subnet fails.

Client behavior for using the Smart Agent

A client ORB application can be configured to use a specific range of ports to bind to the Smart Agent by setting the following properties:

```
vbroker.agent.clientPort
vbroker.agent.clientPortRange
```

The port range ensures that the client ORB uses the local ports in a given range only. The client port range is required because Windows/NT delays actual closure of ports resulting in limited use of the port range.

Using GateKeeper with other CORBA services

From a client's perspective, GateKeeper is transparent to all other CORBA Services. There is no distinction made between a usual server object and other CORBA services such as the Naming Service, Transaction Service, Notification Service, Event Service, and so forth..

In a server-side configuration, the server can be configured to specify the firewall component in its IOR which is identified by the client ORB and is used only when required. In such cases, the client fails over to bind to the server using GateKeeper only if a direct connection could not be established. Assuming that `iiop_tp` is the default server-engine used by the server, the following example shows a typical set of properties for a firewall configuration:

```
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths =Queen,King
vbroker.firewall-path.Queen=Atlantic,Pacific
vbroker.firewall-path.King=Indian
vbroker.firewall-path.Atlantic.type=TCP
vbroker.firewall-path.Atlantic.host=www.borland.com
vbroker.firewall-path.Atlantic.iiop_port=25000
vbroker.firewall-path.Atlantic.hiop_port=25003
vbroker.firewall-path.Atlantic.ssl_port=25004
```

```
vbroker.firewall-path.Pacific.type=PROXY
vbroker.firewall-path.Pacific.ior=http://www.mygk1domain.com/gatekeeper.ior
vbroker.firewall-path.Indian.type=PROXY
vbroker.firewall-path.Indian.ior=http://www.mygk1domain.com/gatekeeper.ior
```

In a client-side configuration, the GateKeeper IOR can be provided to the Client ORB. In such a scenario, the client makes all its operations using GateKeeper. In this case, the following properties are useful:

```
vbroker.orb.alwaysProxy=true
vbroker.orb.gatekeeper.ior=http://www.mydomain.com/gatekeeper.ior
```

In some cases, when the Smart Agent is not accessible by the client, GateKeeper is used to locate server objects. In such a scenario, it is recommended that you use the Location Service available through GateKeeper. In this case, the following property is used for GateKeeper:

```
vbroker.gatekeeper.locationService=true
```

Additionally, the following property is used on the client-side to locate objects:

```
vbroker.locator.ior=http://www.mydomain.com/gatekeeper.ior
```

Configuring GateKeeper with an HTTP proxy server

When an HTTP proxy server is running between the client and GateKeeper, GateKeeper needs to publish the HTTP proxy server's IP host/port address in its IOR. The technique described below can be used to achieve this goal. The following GateKeeper properties can be set which resemble a Network Address Translation configuration. In this case, the HTTP Proxy Server is acting as a NAT.

```
vbroker.se.exterior.proxyHost=142.186.142.21
vbroker.se.exterior.scm.ex-hiop.listener.proxyPort=32001
```

Note

Setting both of the above properties is not mandatory. In this scenario, GateKeeper appears to be behind a NAT device and, as such,.all clients trying to communicate with GateKeeper using HTTP tunneling will always pass their requests through the HTTP Proxy Server.

Additional server engines in GateKeeper

There are three in-built server engines available in GateKeeper:

- iiop_tp
- exterior
- interior

The iiop_tp server engine is used for administrative purposes only. The exterior and interior server engines are used for external and internal networks, respectively. When using TCP/IP networks, each server engine may be associated with a network IP host address, for example:

```
vbroker.se.exterior.host=142.186.142.21
vbroker.se.interior.host=142.186.182.30
vbroker.se.iiop_tp.host=192.73.8.25
```

This version of GateKeeper does not allow adding new server engines using the properties file.

Additional listeners or server connection managers in GateKeeper

GateKeeper can have more than one server connection manager (SCM) or listener for a given type of service. Usually, an SCM provides a specific type of service, such as IIOP, SSL, HIOP, HIOPS, and so forth. Each SCM is bound to a server engine, such as exterior or interior. To configure an SCM, you must assign a logical name (for example, `myscm`), and append this name to the following property:

```
vbroker.se.exterior.scms=ex-iiop,ex-hiop,myscm
```

Furthermore, the following properties must be added for each SCM (see Appendix A for more details):

```
vbroker.se.exterior.scm.myscm.manager.type=Socket
vbroker.se.exterior.scm.myscm.manager.connectionMax=0
vbroker.se.exterior.scm.myscm.manager.connectionMaxIdle=0
vbroker.se.exterior.scm.myscm.listener.type=IIOP
vbroker.se.exterior.scm.myscm.listener.port=683
vbroker.se.exterior.scm.myscm.listener.proxyPort=0
vbroker.se.exterior.scm.myscm.listener.giopVersion=1.2
vbroker.se.exterior.scm.myscm.dispatcher.type=ThreadPool
vbroker.se.exterior.scm.myscm.dispatcher.threadMax=100
vbroker.se.exterior.scm.myscm.dispatcher.threadMin=0
vbroker.se.exterior.scm.myscm.dispatcher.threadMaxIdle=300
```

GateKeeper stress/load metrics

Because GateKeeper is a Java based ORB service implementation, many Java tools can be used to obtain performance characteristics.

The VisiBroker Console provides certain real time performance characteristics about any ORB service (including GateKeeper). It can display information related to allocated memory, numbers of threads, connections, fragmentation, and so forth.

Deploying GateKeeper as a servlet

This section describes an example of deploying GateKeeper as a servlet into a Tomcat 5.0 web server. For earlier or later versions of Tomcat, some small modifications may be required.

This example makes use of the bank_agent example along with the supplied Client.properties, which among other things will force the client to connect to a server only via the gatekeeper servlet embedded in the web server. The bank_agent example is located in the following directory:

```
<install_dir>/examples/vbroker/basic/bank_agent
```

The additional files you will need to run the example in this scenario are:

- **web.xml**—the deployment descriptor for gatekeeper to be deployed as a servlet.
- **Client.properties**—the properties needed to set the bank_agent client to connect to the bank server via the GateKeeper embedded inside a web server as a servlet.

At the end of this section there are screen dumps of web.xml and Client.properties (see web.xml and Client.properties) which you can copy, paste, and save to designated files in specified directories.

Building the example

1 Download free copy of Tomcat web server from `http://jakarta.apache.org/tomcat/index.html`, and follow the instructions to install it. A properly functioning installation can be verified by launching a web browser for http://localhost:8080.

2 Copy, paste, and save web.xml (see web.xml) to <Tomcat root install>/webapps/gatekeeper_servlet/WEB-INF/web.xml. Create subdirectories as needed.

3 Open and edit the file

```
<Tomcat root install>/webapps/gatekeeper_servlet/WEB-INF/web.xml
```

and change the following portion to correctly refer to your osagent Tomcat ports.

```
<init-param>
    <param-name>vbroker.agent.port</param-name>
    <param-value>YOUR OSAGENT PORT</param-value>
</init-param>
 ...
<init-param>
    <param-name>
        vbroker.se.exterior.scm.ex-hiop.listener.port
    </param-name>
    <param-value>
        TOMCAT HTTP PORT. OUT OF TOMCAT BOX, THIS MUST BE 8080
    </param-value>
</init-param>
```

4 Copy the following jars from

```
<install_dir>/lib/
```

to

```
<Tomcat install root dir>/shared/lib
```

Putting the jars into the Tomcat shared/lib/ directory will make them available to all web applications deployed in the container. If this is not desired, consult the Tomcat documentation for the other lib directories.

- lm.jar
- sanctuary.jar
- vbjorb.jar
- sanct4.jar
- vbjclientorb.jar
- vbsec.jar

1 Copy, paste, and save Client.properties (see Client.properties) to

```
<install_dir>/examples/vbroker/basic/bank_agent
```

and open and edit the following settings.

```
vbroker.orb.gatekeeper.ior=http://<host>:<port>/gatekeeper_servlet/
gatekeeper.ior
```

where <host> is the IP of machine on which Tomcat is running and <port> is the HTTP port to which Tomcat is listening. This must be the same number as the port in the web.xml above. For out-of the box Tomcat installations this must be set to 8080.

Running this example

1 Set the proper environment for the existing VisiBroker (that is, execute ${VBROKERDIR}/vbroker.sh on UNIX platform).

2 Build the basic/bank_agent example if necessary.

3 Make sure osagent is running.

4 Make sure that ${JAVA_HOME} and ${PATH} consistently refer to the desired JDK.

5 Start Tomcat by executing the following command:

Windows

```
<Tomcat root install>/bin/startup.bat
```

UNIX

```
<Tomcat root install>/bin/startup.sh
```

6 Navigate to the example basic bank agent directory:

```
<install_dir>/examples/vbroker/basic/bank_agent
```

7 Start the bank server by executing the following command:

```
vbj Server
```

8 Start the client by executing the following command:

```
vbj -DORBpropStorage=Client.properties Client
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//
EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>

  <display-name>GateKeeper Servlet</display-name>

  <description>GateKeeper as a servlet example</description>

  <servlet>

    <servlet-name>GateKeeperServlet</servlet-name>

    <servlet-class>
      com.inprise.vbroker.gatekeeper.servlet.Servlet
    </servlet-class>

    <load-on-startup />

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-hiop.listener.path
      </param-name>
      <param-value>
        /gatekeeper_servlet/servlet
      </param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.agent.port</param-name>
      <param-value>PUT YOUR OSAGENT PORT</param-value>
    </init-param>

<!-- Some setups may not allow UDP broadcast to locate osagent
     In that case, uncomment and set the following correctly
    <init-param>
      <param-name>vbroker.agent.address</param-name>
      <param-value>
        PUT IP OF THE MACHINE, ON WHICH OSAGENT IS RUNNING
      </param-value>
    </init-param>
```

```
-->

    <init-param>
      <param-name>vbroker.gatekeeper.referenceStore</param-name>
      <param-value>
        webapps/gatekeeper_servlet/gatekeeper.ior
      </param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.se.exterior.scms</param-name>
      <param-value>ex-iiop,ex-hiop</param-value>
    </init-param>



<!-- If you want Visibroker log messages, uncomment this.
     Log messages will go to the specified file below, relative
     to Tomcat root install dir

    <init-param>
      <param-name>vbroker.orb.debug</param-name>
      <param-value>true</param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.orb.logLevel</param-name>
      <param-value>7</param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.orb.warn</param-name>
      <param-value>2</param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.orb.logger.output</param-name>
      <param-value>webapps/gatekeeper_servlet/log.txt</param-value>
    </init-param>
-->

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-iiop.listener.type
      </param-name>
      <param-value>Disabled-IIOP</param-value>
    </init-param>

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-hiop.listener.port
      </param-name>
      <param-value>8080</param-value>
    </init-param>

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-iiop.listener.port
      </param-name>
      <param-value>0</param-value>
```

```
    </init-param>

  </servlet>

  <servlet-mapping>
    <servlet-name>GateKeeperServlet</servlet-name>
    <url-pattern>/servlet</url-pattern>
  </servlet-mapping>

</web-app>
```

Client.properties

```
# The following line is only one (single) line
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init,com.inprise.vbroker.H
IOP.Init

vbroker.orb.alwaysTunnel=true
vbroker.orb.alwaysProxy=true

# The following line is only one (single) line
vbroker.orb.gatekeeper.ior=http://host:8080/gatekeeper_servlet/gatekeeper.ior

# Uncomment the following lines for debug messages
# vbroker.orb.debug=true
# vbroker.orb.warn=2
# vbroker.orb.logLevel=7
```

Index

H

I

Index

S

Index