



AGETOR®

COM Integration

Table of contents

1	COM Integration	3
1.1	Requirements.....	3
1.2	Installation.....	3
1.3	Architecture.....	4
1.3.1	Service ORB:.....	4
1.3.2	ClientORB:	5
1.4	Message reading/writing:	6
1.4.1	Reading methods:	6
1.4.2	Reading sequence continuation:.....	6
1.4.3	Reading proxy status:	6
1.4.4	Writing methods:.....	7
1.4.5	Writing sequence continuation:	7
1.5	Implementing the IDL defined protocol	7

1 COM Integration

This document describes how to interact with AGETOR through COM. This means that any COM program will be able to send requests (invoke methods) to any AGETOR service as well as reply to requests (execute methods) from any AGETOR client.

The integration consists of a COM DLL, acting as a proxy to the AGETOR Broker, communicating with this through sockets.

Clients send a message and await the reply. Likewise will services await a request message, execute some code and return a reply message. Messages consist entirely of the simple types defined in IDL. However upon this COM object stub and skeleton code may be generated in a host language to allow method invocation syntax and user defined IDL types.


1.1 Requirements

The following elements must be present on the system:

- AGETOR Development Kit (ADK) version 2.0.1 or later.
- Java development kit 1.3 or later

1.2 Installation

This section gives you step-by-step information on how to install ORBLib.

 The newest version of ORBLib is always available at the Bording Data download center at <http://www.agetor.dk>.

The existing configuration will be retained in the new installation, but please consult the release.txt for any changes you might need to incorporate.

Download the newest version of the ORBLib and place it in your "AGETOR_HOME/install/packages" directory. If you are upgrading an existing installation the existing configuration will be retained and do the following:

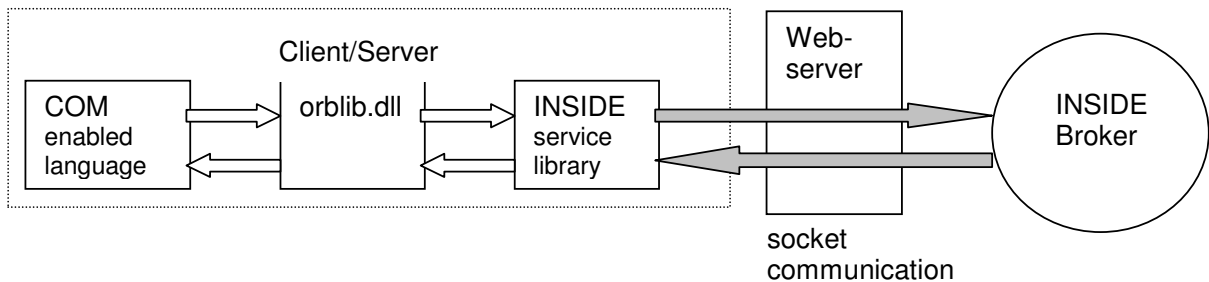
- Open a command window with "AGETOR_HOME/bin/prompt.bat" and run "installer.bat".
- After the AGETOR® installtool has started open a Internet browser and point to "http://localhost:8020".
- If prompted for login and password, please type in your login and password.
- Under "Product(s) ready to install" click on "ORBLib 2.x.x" and answer the few questions.

After you have installed the newest version you will have to setup the new dll with the "ORBSetup.exe" program located in AGETOR_HOME\app\com.

Please note, that new properties might have been added and you should always consult the release.txt for any changes you might need to incorporate.

1.3 Architecture

The orblib.dll consists of wrapper functions written in Java and utilizes the AGETOR service library to read and write simple types from the network communication.



1.3.1 Service ORB:

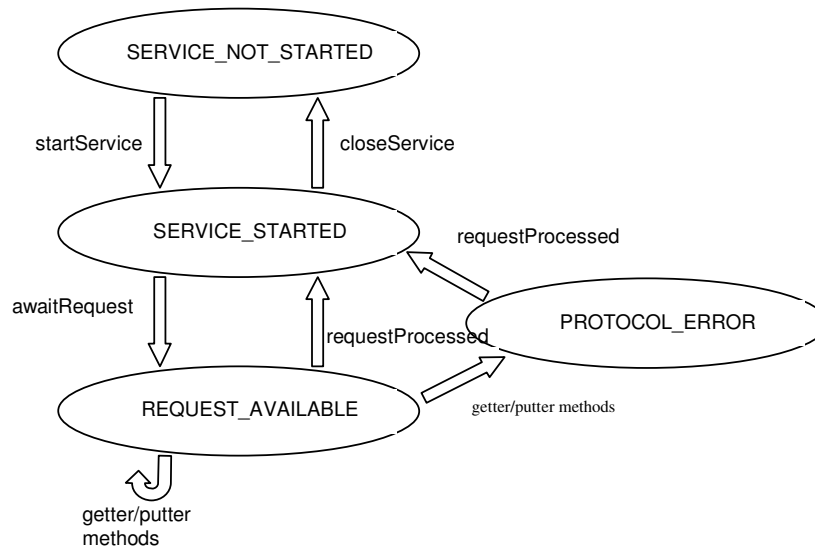
The service loads the orblib.dll and instantiates a ServiceORB object. It starts the AGETOR service library by indicating the port number from which it will be servicing requests. This port number must correspond to the port number assigned to the service on the AGETOR broker. The broker will now forward any requests to this service to the indicated port number.

The server now awaits an incoming request, which means that control is left to the AGETOR service library, and only returned once a request is available. If the server is single threaded any Windows will be left non-responsive and not updated until a request is available and control is returned to the thread.

Once the AGETOR service library receives the request it will return control to service indicating the request type. The IDL defines which parameter types the request consists of and service must now read the input parameters and write the output parameters in the defined order. When this is done the service notifies the AGETOR service library that the response for the request has been completed. The AGETOR service library then returns the response to the broker. The service is now ready to process the next request and must await this.

startService	Starts the service on the given port number.
closeService	Closes the service and releases system resources.
awaitRequest	Block calling thread until request from client is available. Parameters may now be read from the request using the <code>get<type></code> functions. Also parameters may be written to the reply using the <code>put<type></code> functions. The reply is only sent when the <code>requestProcessed</code> function is called.

requestProcessed	<p>Send the generated reply. No more parameters can be read from the request or written to the reply, before a new request is available (awaitRequest). If protocol errors were encountered in the request, no reply is generated, and the status is cleared, thus preparing the service for a new request.</p>
------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------



1.3.2 ClientORB:

internalConnect	Connects to an AGETOR Broker on the specified host and port.
close	Close connection and release system resources.
makeInvocation	Makes a ready for a new invocation with the given environment, question number and the name of the function to invoke. After this you can set the parameters with the given put<type> methods.
doInvoke	Sends the invocation to the service. The calling thread is blocked until a reply from the service is available or the timeout expires. Return values can hereafter be read with the get<type> methods.
setTimeout	Sets the timeout for sendQuestion blocking.
getTimeout	Gets the current timeout for sendQuestion blocking.
brokerLogin	Given username, environment and password it can perform a login to the AGETOR broker.

1.4 Message reading/writing:

The methods for marshalling and demarshalling datastructures to and from a request/reply are done through 9 getter and putter methods for each of the simple types in IDL (including the extra Date type). Further two methods for testing for and indicating sequence continuation is provided. Finally a getStatus method is provided indicating if a mismatch has occurred when reading (getting) simple types from this request.

1.4.1 Reading methods:

<code>getBoolean</code>	Reads a boolean (0 or 1) from the request.
<code>getChar</code>	Reads a single char from the request.
<code>getDateAsString</code>	Reads a date from the request. The date is returned as a string with the format "yyyy-mm-dd hh:mm:ss"
<code>getDouble</code>	Reads a double floating point number from the request.
<code>getFloat</code>	Reads a floating point number from the request.
<code>getLong</code>	Reads a long (32-bit number) from the request.
<code>getOctet</code>	Reads a byte (8-bit number) from the request.
<code>getShort</code>	Reads a short (16-bit number) from the request.
<code>getString</code>	Reads a string from the request.

1.4.2 Reading sequence continuation:

<code>getMoreElements</code>	Returns whether a sequence contains more elements (true or false).
------------------------------	--------------------------------------------------------------------

1.4.3 Reading proxy status:

getStatus	Returns the status of the proxy ORB. OK (0) INVALID_PORT (1) CONNECTION_FAILED (2) INVOCATION_FAILED (3) INVOCATION_TIMEDOUT (4) SERVICE_NOT_STARTED (10) PROTOCOL_ERROR (11) SERVICE_STARTED (12) MESSAGE_AVAILABLE (13)
getStatusMessage	Returns a string describing the current status of the proxy ORB

1.4.4 Writing methods:

putBoolean	Write a boolean (0 or 1) to the reply.
putChar	Write a single char to the reply.
putDateFromString	Write a date number to the reply. The date is assumed to be in the format "yyyy-mm-dd hh:mm:ss"
putDouble	Write a double floating point number to the reply.
putFloat	Write a floating point number to the reply.
putLong	Write a long (32-bit number) to the reply.
putOctet	Write a byte (8-bit number) to the reply.
putShort	Write a short (16-bit number) to the reply.
putString	Write a string to the reply.

1.4.5 Writing sequence continuation:

putMoreElements	Indicates whether a sequence contains more elements (true or false).
-----------------	----------------------------------------------------------------------

1.5 Implementing the IDL defined protocol

Once an interface has been defined in IDL, the server and client must both adhere to this interface. This is done by writing and reading simple types in the correct order. Currently in Java, C, Visual Basic and ABF this is handled by auto-generated stubs and skeletons, which relieves the application programmer from the burden of strict protocol adherence. However this stubs and skeletons are generated in the host language to map types correctly, and the direct COM



integration requires the application programmer to get all input parameters and put all output parameters of requests.