



**AGETOR®**

Visual Basic  
Integration

## Table of contents

1	Visual Basic Integration.....	3
1.1	Overall .....	3
1.2	Requirements.....	3
1.3	Installation.....	3
1.3.1	Architecture .....	4
1.4	Example.....	5
1.4.1	Service ORB:.....	6
1.4.2	Internal ORB:.....	7
1.4.3	Using a AGETOR service from a scripting environment.....	8

## 1 Visual Basic Integration

This section describes how to interact with AGETOR server- and client-side components using Visual Basic.

This means that any Visual Basic program will be able to invoke and implement remote procedure calls, i.e. act as either an AGETOR service or AGETOR client.

### 1.1 Overall

The integration is based on the COM integration, which provide message semantics and transmission of simple IDL types. By adding Visual Basic code, this is hidden behind a method invocation abstraction. For each IDL module, a code-generator (idl2vb) generates a VB DLL containing classes matching IDL structures, interfaces matching IDL interfaces as well as client side stubs and server side skeletons.


### 1.2 Requirements

The following elements must be present on the system:

- ADK 2.0.2.
- Java development kit 1.3 or later
- The COM integration - ORBLib version 2.0.1.

### 1.3 Installation

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

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

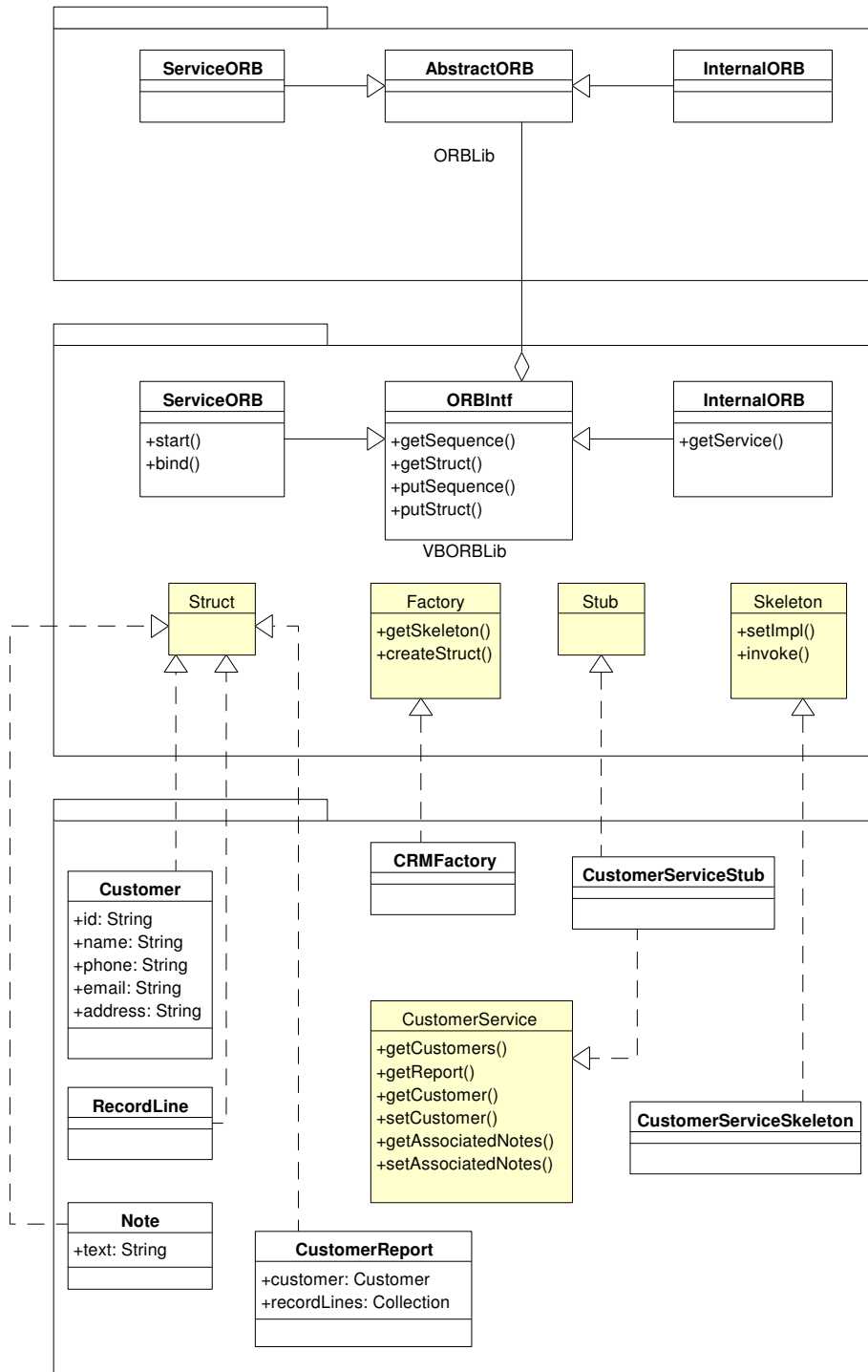
Download the newest version of the VBORBLib and copy the package into your "AGETOR\_HOME/install/packages" directory. If you are upgrading an existing installation the existing configuration will be retained and do the following:

- 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 "VBORBLib 2.x.x" and answer the few questions.

After you have installed the newest version you will have to setup the new dll by running the setup program "VBSetup.exe" 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.

### 1.3.1 Architecture



## 1.4 Example

To explain the integration an example IDL is constructed. The following IDL illustrates some simple operations within a Customer-relations-management system (CRM).

```
module dk.bording.idl.crm {
    struct Customer {
        string<10> id;
        string<40> name;
        string<20> phone;
        string<50> email;
        string<50> address1;
        string<50> address2;
    };
    typedef sequence <Customer> Customers;

    struct RecordLine {
        string<30> text;
        float amount;
        float total;
    };
    typedef sequence <RecordLine> RecordLines;

    struct Note {
        string<100> text;
    };
    typedef sequence <Note> Notes;

    struct CustomerReport {
        Customer customer;
        RecordLines recordLines;
    };

    /**
     * CustomerService allows retrieval of CustomerReports,
     * setting Customer information,
     * and getting/setting customer associated notes.
     */
    interface CustomerService #env="dev" #qno=700 {

        /**
         * getCustomers returns a sequence of all Customers.
         * @returns all Customers.
         */
        Customers getCustomers(
        );

        /**
         * getReport gets the CustomerReport associated with a Customer.
         */
    };
};
```

```
        * @param id is the Customer identifier.
        * @param report is the CustomerReport incl. Customer information.
        * @returns whether the report existed.
    */
    boolean getReport(
        in string<10> id,
        out CustomerReport report
    );

    /**
     * getCustomer gets the Customer with the given id.
     * @param id is the Customer identifier.
     * @returns whether the Customer existed.
     */
    boolean getCustomer(
        in string<10> id,
        out Customer customer
    );

    /**
     * setReport sets the Customer information.
     * @param customer is the Customer information incl. an identifier.
     */
    void setCustomer(
        in Customer customer
    );

    /**
     * getAssociatedNotes gets the Notes associated with a Customer.
     * @param id is the Customer identifier.
     * @param notes is the associated Notes.
     * @returns whether the notes existed.
     */
    boolean getAssociatedNotes(
        in string<10> id,
        out Notes notes
    );

    /**
     * setAssociatedNotes sets the notes associated with a Customer.
     * @param id is the Customer identifier.
     * @param notes the Notes that will be associated the Customer.
     */
    void setAssociatedNotes(
        in string<10> id,
        in Notes notes
    );

};
};
```

#### 1.4.1 Service ORB:

When writing a service in Visual Basic, you need to write an implementation of the interface. In the example above, there is an implementation of the CustomerService interface. To make things work



the ServiceORB must be able to invoke the correct methods on your implementation through the interface. Therefore you need to 'bind' your implementation to a ServiceORB and start the ServiceORB on a specified port. When a message from a client is received the skeleton will extract the parameters from the message and call your implementation of the CustomerService interface. Since the generic ServiceORB does not know the types in the specific IDL it is necessary to pass a factory object to the ServiceORB, which will then delegate object instantiation to the specific factory.

start	Starts the ServiceORB on the given port number and blocks the calling thread.
bind	Binds an interface implementation to the ServiceORB for callback.

The following Visual Basic code demonstrates the necessary binding code.

First, create an instance of the implementation of the CustomerService interface:

```
Dim cs As CustomerService
Set cs = New CustomerServiceImpl
```

Second, create the specialized factory for the IDL:

```
Dim vf As VssFactory
Set vf = New VssFactory
```

Third, bind the interface name, the implementation and the specific factory to the ServiceORB:

```
Dim orb As VBORBLib.ServiceORB
Set orb = New VBORBLib.ServiceORB
orb.bind "dk.bording.idl.vss.CustomerService", cs, vf
```

Finally, leave control to the ServiceORB which will block until a message is available:

```
orb.start 8002
```

### 1.4.2 Internal ORB:

When writing a Visual Basic program, you must obtain stub object acting as a proxy for the remote service object you wish to interact with. First you instantiate the InternalORB telling it the hostname and port of the AGETOR Broker. Then you request a stub object by name and environment. The stub object will now marshal parameters in method invocations putting them in a message, sending the message, block the thread until reply is available, demarshal the parameters in the reply and finally return control to your calling program. From your calling program this just looks as a normal method invocation, however it will be executed by a remote service.

connect	Connects to an AGETOR Broker on the specified host and port.
---------	--



getService	Returns a stub object acting as a proxy for method invocations on the remote service object. Invoking methods on this object will make the stub marshal all parameters into a message, send the message, await the reply and demarshal parameters from the reply message
setTimeout	Sets the timeout for remote method invocation.
getTimeout	Gets the current timeout for remote method invocation.

The following Visual Basic code demonstrates how to use invoke remote methods.

First, create an instance of the InternalORB:

```
Dim orb As InternalORB
Set orb = New InternalORB
```

Second, create the specialized factory for the IDL:

```
Dim vf As VssFactory
Set vf = New VssFactory
```

Third, connect the orb to the AGETOR Broker on a machine named hest at Bording Data on port 1234:

```
Dim result as Long
result = orb.connect("hest.bording.dk", 1234)
```

Fourth, get the stub object for the IDL defined interface named dk.bording.idl.crm.CustomerService, which is running on an environment named dev on the machine the orb is connected to:

```
Dim cs As CustomerService
Set cs = orb.getService("dk.bording.idl.crm.CustomerService", "dev")
```

Finally, invoke a method to retrieve the Customer with id "tep" from the remote service:

```
Dim cust As Customer
Dim existed As Boolean
existed = cs.getCustomer("tep", cust)
```

### 1.4.3 Using a AGETOR service from a scripting environment

When you want to use an AGETOR service from a scripting environment like VBScript in an ASP-page or VBA in Microsoft Excel you will have to obtain the stub to the service a little differently because they use late bounding in there invocation of methods. Like wise VBScript has no real type definition leaving everything as a Variant.

The following VBScript code demonstrates how to obtain a reference to a AGETOR service and invoke its methods:

First create an instance of the factory object:

```
Dim vf
Set vf = CreateObject("crm.crmFactory")
```

Then connect the factory object with the ORB giving it both a URL and a port:

```
Dim result
result = vf.ConnectToORB("hest.bording.dk", 1234)
```

Hereafter you can get a script stub from the factory instance:

```
Dim cs
Set cs = vf.getCustomerServiceService("dev")
```

Now you can invoke the methods on the service:

```
Dim cust
Set cust = vf.getCustomerStruct

Dim existed
existed = cs.getCustomer("tep", cust)
```

Finally would have to disconnect the connecting to the AGETOR service when you are done:

```
vf.disconnect
Set vf = Nothing
Set cs = Nothing
```