



## *AgentService External Runtime*

*How to mask your application as a sort of external agent*

*The AgentService Team*

*Copyright @ 2008 – l.i.d.o. – DIST University of Genoa*

## **Abstract**

Several *AgentService* users expressed the need to allow their external applications to take actively part in the social activity of an *AgentService* platform. In particular, they want to send and receive messages from classic agents and to exploit the services offered by the platform.

In this guide we want to show how to use *AgentService External Runtime*: a library that allow an third-part programs to act as agents and participate to the life cycle of the platform.

# Index of Contents

AgentService External Runtime.....	1
How to mask your application as a sort of external agent .....	1
The AgentService Team.....	1
Copyright @ 2008 - i . i . d . o . - DIST University of Genoa .....	1
Abstract .....	2
Index of Contents.....	3
1. Introduction .....	4
2. Deployment of the infrastructure .....	4
2.1. Configuring the desktop platform.....	4
2.2. Allowed users .....	5
3. Your application as an agent.....	5
3.1. Referencing the library.....	5
3.2. The AER object model .....	6
3.3. The RemoteRuntime class.....	6
3.4. The RemoteMessageServiceClient class.....	7
3.5. The RemoteWhitePagesServiceClient class.....	7
3.6. The RemoteYellowPagesServiceClient class.....	8
4. Getting started with a simple example.....	8
4.1. How to use AER .....	8

# 1. Introduction

AgentService External Runtime (AER) is a library which can be imported in a .NET project in order to simply mask the application as an AgentService agent which will be able to:

- *Connect to the platform* through the Windows Communication Foundation.
- *Enter credentials* in order to gain full access to the platform services.
- *Choose its name* in order to register as a valid agent.
- *Send and receive messages.*
- *Subscribe and query the Yellow Pages Service.*
- *Query the White Pages Service.*
- *Disconnect from the platform.*

Once the external agent subscribes itself, the platform creates a local message queue in order to collect the messages sent to the external agent and add the agent personal details to the White Pages Service.

AER is based on the WCF server used by the *AgentService Mobile infrastructure*, so from the platform side, the configuration steps are essentially the same.

In particular, the WCF server monitors the activity of the external agent and if it is no longer online, then the platform removes the message queue and unsubscribes the agent from the White Pages Service.

The AER periodically polls the platform in order get all the platform events aimed to the external agent (in particular, for example: new incoming message). If the periodic poll inactivity exceeds a timeout (customizable through a configuration file (see 2.1)), the platform declares the external agent *lost* and deletes its references.

## 2. Deployment of the infrastructure

### 2.1. *Configuring the desktop platform*

Install a standard AgentService platform and then open the file *conf/platform.conf*. You must essentially configure three parameters:

- *system.wcfserver.enable* (true or false) in order to run the WCF Server.
- *system.wcfserver.ip* for the local IP Address.

- `system.wcfserver.tcpport` is the port opened for the WCF Server.

Other parameters are the following:

- `system.wcfserver.debug` (*true or false*), true to see debug information.
- `system.wcfserver.checktime` (in seconds) is the timeout for the deletion of an external agent which does not interact with the platform for the submitted value seconds. An external agent is considered *alive* if periodically polls the server acquiring the list of events for the external agent.
- `system.wcfserver.msgsize`: is the maximum size for a message exchanged between an external agent and the desktop platform.

## 2.2. Allowed users

In order to allow an external agent to connect to the platform, the user must be authenticated.

In the root of the AgentService installation, create a file named `users.xml` and fill it with *usernames* and *passwords*:

```
<Users>
  <User userid="andrea" password="123"/>
  <User userid="alberto" password="345"/>
  <User userid="davide" password="567"/>
</Users>
```

Figure 1: user credentials.

The desktop platform is now configured. Be sure that the computer on which the platform runs is reachable from the external application (check the network configuration, firewalls, etc.).

## 3. Your application as an agent

### 3.1. Referencing the library

In order to write the code necessary to contact the platform services (and its agents) the programmer has to add the reference to three libraries:

- `AgentServiceExternalRuntime.dll`: the dll containing the classes which manage the channel between the application and the platform.
- `AgentService.dll`: the usual AgentService core dll which you can find, for example, in the platform installation directory.
- `AgentServiceInterface.dll`: another AgentService core dll.

### 3.2. The AER object model

In Figure 2 the complete class view of AER is shown. From the programmer point of view, only few classes are significant: *RemoteRuntime*, *RemoteMessageServiceClient*, *RemoteWhitePagesServiceClient*, and *RemoteWhitePagesClient*.

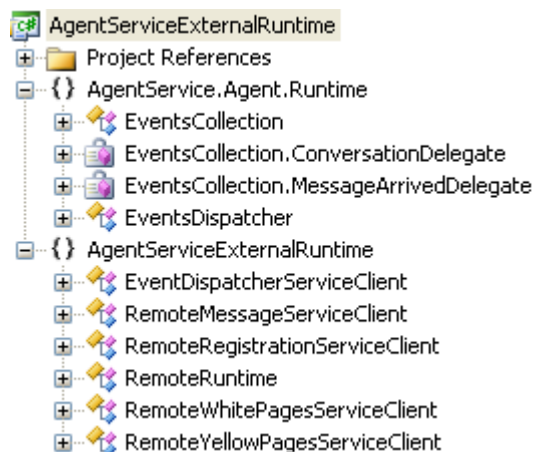


Figure 2: the AER object model.

Every class implements the code needed to contact the remote service owned by the platform. Practically each class contacts a web service through which the platform exposes a particular service.

### 3.3. The RemoteRuntime class

*RemoteRuntime* is the main class of the AER. It exposes the methods to *connect* and *disconnect* from the remote platform or to *access* the single platform services. In details *RemoteRuntime* exposes these methods:

- *Connect()*: it connects the application to the given platform. It returns a Boolean value: *true* if the connection is successfully established, *false* otherwise. The required input parameters are the *ip* and the *port* of the remote platform, the *username* and *password* of an enabled user, the *name of the agent* representing the external application, the *name of the remote platform*. This method performs all the steps necessary to fully include the application in the agent social community (in particular, the creation of a message queue, the notification to the White Pages Service, the creation of a thread for the event polling, etc.). It creates also the proxies which hide the web service invocations.
- *Disconnect()*: it disconnects the agent from the remote platform.

Moreover, *RemoteRuntime* exposes the following properties:

- *Agent* (read only): it returns the AID of the current external agent.

- *ID* (read only): the unique ID which identifies the local branch of the platform. From a practical point of view the ID is negligible.
- *IP* (read only): the IP of the remote platform.
- *IsConnected* (read only): true if the *RemoteRuntime* is connected to the remote platform.
- *LastException* (read only): it returns the last occurred exception.
- *PollingTime*: it sets or gets the wait between two consecutive event downloads. Remember that the *RemoteRuntime* starts a thread which cyclically queries the platform in order to download the events for the external agent. By default the *pollingTime* is 500 milliseconds.
- *Port*: the port where the remote platform is listening.
- *RemoteMsgClient* (read only): the client which manages the message exchange. (see 3.4)
- *RemoteWhitePagesClient* (read only): the proxy which contacts the White Pages Service. (see 3.5)
- *RemoteYellowPagesClient* (read only): the client managing the remote yellow page service. (see 3.6).

### **3.4. The RemoteMessageServiceClient class**

The *RemoteMessageServiceClient* class is exposed by the *RemoteRuntime* in order to allow external agents to send or receive messages.

At now, the conversation mechanism is not implemented, so the external agent is only able to manage single messages.

*RemoteMessageServiceClient* exposes these basic methods:

- *HasMessages()*: it returns *true* if there are new messages on the remote message queue.
- *PeekMessage()*: it returns the first *AgentMessage* in the remote queue, but leaves it in.
- *ReceiveMessage()*: it gets the first available *AgentMessage* and removes it from the queue.
- *RemoveMessage()*: it removes the given message from the remote queue.
- *SendMessage()*: this method sends the given message to the list of recipients.
- *WaitForMessage()*: a blocking method which returns the first incoming message.

### **3.5. The RemoteWhitePagesServiceClient class**

The *RemoteWhitePagesServiceClient* class is able to contact the remote White Pages service in order to perform queries on the list of registered agents. The following methods are available:

- *Search()*: it searches all those agents whose names are compatible with the given regular expression. The method returns an array of *AgentDescription* objects.
- *SearchExact()*: it return the *AgentDescription* of the agent with a given name.

### **3.6. The *RemoteYellowPagesServiceClient* class**

By using the *RemoteYellowPagesServiceClient*, the external agent is able to publish its services in the scope of the whole platform and to find the required agents which are service-providers for a particular service.

The following methods are supported:

- *Deregister()*: it removes the whole agent entry from the Yellow Pages Service.
- *Register()*: it submits the given *DFAgentDescription* to the Yellow Pages Service.
- *SearchAgent()*: it returns an array of *DFAgentDescription* which match the given *ServiceDescription*.

## **4. Getting started with a simple example**

### **4.1. How to use AER**

In the following code snippet we show how easily connect a remote platform in order to give the characteristics of external agent to your application.

Make sure that the remote platform is running, the WCF server is active and there exists a network connection between the platform and the external application.

First, create a link between the application and the platform:

```
RemoteRuntime runtime = new RemoteRuntime();  
  
runtime.Connect("10.0.0.34", "8000", "andrea", "123", "LegacyAgent#1",  
               "ServerPlatform#3");
```

By calling the *Connect* method, the application establishes a connection with the platform and in the meantime introduces the external agent in the remote platform.

Now the application is seen as an agent from the whole *ServerPlatform#3* platform (and potentially from its federated platforms). The external agent

named now *LegacyAgent#1* can, for example, subscribe it to the Yellow Pages in order to advertise its service: *Administration-Console*.

```
DFAgentDescription ad = new DFAgentDescription(runtime.Agent);
ServiceDescription sd = new ServiceDescription("Administration-
    Console", "Windows-Application-Console");
ad.Services.Add(sd);

runtime.RemoteYellowPagesClient.Register(ad);
```

The application can also search for a particular type of agent. For example, *LegacyAgent#1* has to search for a sensor agent which monitoring the temperature:

```
DFAgentDescription[] ads=runtime.RemoteYellowPagesClient.SearchAgent
    (new ServiceDescription("Sensor-Agent", "Temperature-Sensor"));
```

Now *LegacyAgent#1* knows the name of the sensor agent and is able to send a message to it.

```
AgentMessage msg = new AgentMessage();
msg.Body.Add("content", "send me data");
msg.Envelope.To.Add(ads[0].Aid);
msg.Envelope.From = runtime.Agent;

runtime.RemoteMsgClient.SendMessage(msg);
```

The agent can now wait for the response:

```
AgentMessage reply = runtime.RemoteMsgClient.WaitForMessage();
```

Finally, the connection with the remote platform terminates:

```
runtime.Disconnect();
```