
AIMMS Multi Agent and Web Services User's Guide - Web Services

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit www.aimms.com

Copyright © 1993–2011 by Paragon Decision Technology B.V. All rights reserved.

Paragon Decision Technology B.V.	Paragon Decision Technology Inc.	Paragon Decision Technology Pte.
Schipholweg 1	500 108th Avenue NE	Ltd.
2034 LS Haarlem	Ste. # 1085	80 Raffles Place
The Netherlands	Bellevue, WA 98004	UOB Plaza 1, Level 36-01
Tel.: +31 23 5511512	USA	Singapore 048624
Fax: +31 23 5511517	Tel.: +1 425 458 4024	Tel.: +65 9640 4182
	Fax: +1 425 458 4025	

Email: info@aimms.com
WWW: www.aimms.com

AIMMS is a registered trademark of Paragon Decision Technology B.V. IBM ILOG CPLEX and sc CPLEX is a registered trademark of IBM Corporation. GUROBI is a registered trademark of Gurobi Optimization, Inc. KNITRO is a registered trademark of Ziena Optimization, Inc. XPRESS-MP is a registered trademark of FICO Fair Isaac Corporation. MOSEK is a registered trademark of Mosek ApS. WINDOWS and EXCEL are registered trademarks of Microsoft Corporation. $\text{T}_{\text{E}}\text{X}$, $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$, and $\text{A}_{\text{M}}\text{S}_{\text{L}}\text{A}_{\text{T}}\text{E}_{\text{X}}$ are trademarks of the American Mathematical Society. LUCIDA is a registered trademark of Bigelow & Holmes Inc. ACROBAT is a registered trademark of Adobe Systems Inc. Other brands and their products are trademarks of their respective holders.

Information in this document is subject to change without notice and does not represent a commitment on the part of Paragon Decision Technology B.V. The software described in this document is furnished under a license agreement and may only be used and copied in accordance with the terms of the agreement. The documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Paragon Decision Technology B.V.

Paragon Decision Technology B.V. makes no representation or warranty with respect to the adequacy of this documentation or the programs which it describes for any particular purpose or with respect to its adequacy to produce any particular result. In no event shall Paragon Decision Technology B.V., its employees, its contractors or the authors of this documentation be liable for special, direct, indirect or consequential damages, losses, costs, charges, claims, demands, or claims for lost profits, fees or expenses of any nature or kind.

In addition to the foregoing, users should recognize that all complex software systems and their documentation contain errors and omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors shall not be responsible under any circumstances for providing information or corrections to errors and omissions discovered at any time in this book or the software it describes, whether or not they are aware of the errors or omissions. The authors, Paragon Decision Technology B.V. and its employees, and its contractors do not recommend the use of the software described in this book for applications in which errors or omissions could threaten life, injury or significant loss.

This documentation was typeset by Paragon Decision Technology B.V. using $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ and the LUCIDA font family.

Chapter 4

Web services

In this chapter you will find a description on how to deploy an AIMMS project as a web service. It contains a short introduction on web service terminology, details on the architecture of AIMMS web services, a description of the **Web-service Properties** dialog box in AIMMS, and a step-by-step guide for setting up a basic AIMMS web service.

This chapter

4.1 Introduction

A *web service* is a non-static service, that can be accessed through the web. A web service can be called from any platform and from within any computer language, as long as the language is capable of sending and receiving *web service requests* and *-responses*. For example, web services exist that provide the latest currency exchange rates, or real-time stock quotes.

Web services

Web service requests are specified as so-called *SOAP* requests. *SOAP* stands for *Simple Object Access Protocol*. It is an XML specification, used for communicating with web services. It allows calls to methods exposed by the web service to be represented in XML.

SOAP requests

A web service should always be defined by means of a *WSDL* document. *WSDL* stands for *Web Service Definition Language*. It is an XML specification that defines details about a specific web service, such as the methods exposed by the web service and the URL of the web service. Generally, if you have access to a *WSDL* document, you know everything you need for successfully sending a *SOAP* request to a web service. Many application development environments provide tools that can automatically create web service requests based on the corresponding *WSDL* document.

WSDL specification

AIMMS enables you to expose an AIMMS project as a web service, which can be called from anywhere on the Internet. Web services in AIMMS are implemented as an extension to the AIMMS multi-agent technology. This has the advantage that you can use the advanced features exposed by the multi-agent technology, such as the ability to dispatch messages to worker agents or to attach an AIMMS GUI application to the agent community to monitor the entire agent

Web services in AIMMS

application, for dealing with web service requests as well. For AIMMS agents there is no difference between web service requests and messages exchanged between ordinary AIMMS agents. This means that you can write multi-agent handler- and filter procedures to handle web service requests. AIMMS automatically generates all necessary files to expose a project as a web service, including a WSDL document, describing the methods exposed by the service, together with the XML schema that is expected for the arguments of each method. Please note that handler- and filter procedures to be called from a web service client, can handle a maximum of 256 arguments.

It is also possible to call an external web service from within AIMMS. This feature is restricted to web services that are designed to implement a role in an AIMMS multi-agent community. You have to specify some information about such a web service in the AIMMS web service properties dialog. Also, such external web services must first make themselves known to AIMMS by sending a Logon request to an AIMMS web service first. If these restrictions are met, however, you have quite some freedom in writing your external web service. You can implement such a web service in any language capable of creating a web service. Please note that AIMMS itself can also be used to implement such a web service. This makes it possible that two AIMMS web services can communicate with each other over the Internet.

Calling external web services

Although AIMMS prescribes its own data format in the WSDL document, a lot of applications will have their own XML data formats. To facilitate calling an AIMMS web service from such an application using their native XML format rather than the format prescribed by AIMMS, you can define so-called *attachment arguments* in your multi-agent messages. These can be regarded as a kind of e-mail attachments: along with the SOAP request, you can send additional files of any kind to AIMMS. A very practical use of this would be to send an `.xml` file in the native XML format of the client, where the agent has an associated `.axm` file that defines the mapping between the XML format and identifiers in your model (see also Section 28.4 of the Language Reference). Using the `ReadXML` statement provided by AIMMS, this allows you to read in XML data in user-defined data formats. You can even send the `.axm` file as an attachment if you want your web service listener to be able to deal with varying data formats.

Attachments...

Figure 4.1 shows how to specify an attachment argument in the multi-agent message types dialog box. An attachment argument is just an additional argument type in this dialog box. However, in your filter- and handler procedure in AIMMS, they show up as ordinary string arguments. These arguments always contain the *name* of a file, not the contents of the attachment itself. Depending on whether the attachment was sent through a web service request, or by a multi-agent project on the same machine, the link may differ: in the latter case, a link to the original file is sent. Otherwise, a link to a temporary file

...in more detail

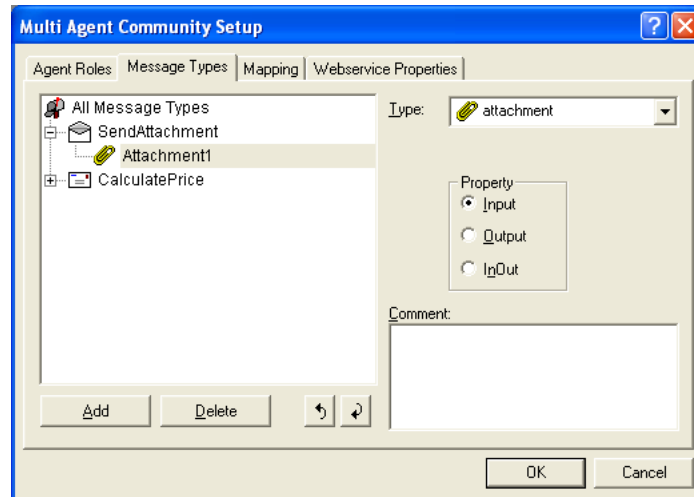


Figure 4.1: Specifying an attachment argument

is sent. Note that you can specify input, output and input/output attachment arguments.

Make sure that you do something with the temporary files that you receive in your handler procedure. After the execution of the handler procedure, all input attachment files are automatically deleted by AIMMS. So, either do something useful with them, or make a copy of the files to an other location, so you can work with them later.

Temporary files

AIMMS web services support two types of attachments:

- *DIME* attachments, and
- *Base64* attachments.

DIME and Base64 attachments

The following two paragraphs explain both types of attachments.

DIME attachments appear as a paperclip followed by the word *attachment* in the dropdown list on the **Message Types** tab of the **Multi Agent Community Setup** dialog box. If you specify any of these attachment arguments in a message, the corresponding request in the WSDL document will *not* show these arguments. This is called the *open-ended* way of dealing with attachments. However, if you specify a comment for the argument in the dialog box, the comment will appear in the WSDL document. To actually send a SOAP request containing attachments to your AIMMS web service, the request must be packed into a *DIME* message, in accordance with the W3C standard for SOAP requests with attachments. The AIMMS web service listener understands both "plain" SOAP requests and *DIME* requests. If your message contains output

DIME attachments

(or input/output) attachment arguments, the response that your AIMMS web service sends will also be in DIME format, using the same format as just described.

Base64 encoded attachments appear as a paperclip followed by the word *att.* (*b64 enc*) in the dropdown list on the **Message Types** tab of the **Multi Agent Community Setup** dialog box. This type of attachment arguments *will* show up in the WSDL document. In order to send an attachment as a base64-encoded attachment, the actual attachments *content* must be base64-encoded and put into the actual SOAP request (i.e. no separate DIME attachment is needed). It is allowed to mix both types of attachments in a single request or response message. For the receiving AIMMS project, there is no difference in the handling of both attachment types, so, in your filter- and handler procedure in AIMMS, they both show up as ordinary string arguments.

*Base64
attachments*

Please note that attachments sent along with your SOAP requests are not restricted to files in XML format. You are allowed to send any kind of binary files to your AIMMS web service. You can, for example, send pictures to an AIMMS web service.

*Not restricted to
XML files*

AIMMS also offers the possibility to create *secure* web services, using the *SSL* (*Secure Socket Layer*) standard protocol. Requests to and responses from secure AIMMS web services will be encrypted. Also, both client and server authentication modes are supported.

*Secure AIMMS
web services*

4.2 The AIMMS RPC mechanism

Your AIMMS installation comes with a separate *RPC* installation. *RPC* stands for *Remote Procedure Call*. From AIMMS version 3.8 onwards, the web service functionality is based on this *RPC* mechanism. In order to use the *RPC* mechanism, you must also install the *RPC* installation file. The main consequence of this change to the *RPC* mechanism, is that certain web service related files have become obsolete, have been renamed or have been added.

*AIMMS RPC
mechanism*

The following files have become obsolete in AIMMS 3.8:

Obsolete files

- The `[ServiceName].AimmsWSConfig.xml` file;
- The `AimmsWebServiceHandler.dll`;
- The `MagentTCP.dll`;
- The `MachineWebServicesConfiguration.xml` file;

The RPC mechanism is based on a Windows service, which also handles the AIMMS web service requests for all the AIMMS web services that you deploy on the machine on which you have installed AIMMS. This service is called **AIMMS HTTP/RPC Handler Service** and can be found in the Windows **Services** configuration window. The service is started automatically when installing AIMMS RPC.

AIMMS
HTTP/RPC
Handler Service

Should you encounter problems with some of your AIMMS web services, it may help to restart the AIMMS RPC service. You can do this by clicking on the Windows **Start** button, and choosing **Settings-Control Panel-Administrative Tools-Services**. Select the **AIMMS HTTP/RPC Handler Service** service and choose **Restart** from the right-mouse menu.

Restarting the
listener

The AIMMS RPC service can be configured through its two configuration files

```
%CommonAppData%\Paragon Decision Technology\ServiceHandler\ServiceListener.cfg
%CommonAppData%\Paragon Decision Technology\ServiceHandler\ServiceSecurity.cfg
```

Service
configuration

where the `%CommonAppData%` symbol is explained below. The AIMMS RPC installation will install two default configuration files, `ServiceListener.cfg.default` and `ServiceSecurity.cfg.default`. If any of the above configuration files are not present, the RPC service will use the default configuration files.

The `%CommonAppData%` symbol refers to the area on your computer where Windows applications can store their common application data. Possible locations for this directory are:

Common
application data

- C:\Documents and Settings\All Users\Application Data (Windows XP, Windows Server 2003)
- C:\ProgramData (Windows Vista/Windows 7)

The default contents of the `ServiceListener.cfg` file is

Contents of Ser-
viceListener.cfg

```
<ServiceListenerConfig>
  <ServiceConfigDirectory></ServiceConfigDirectory>
  <ListenerPort>19339</ListenerPort>
  <ServiceMode>PortRange</ServiceMode>
  <FirstAssignedPort>25000</FirstAssignedPort>
  <LastAssignedPort>26000</LastAssignedPort>
  <LogFile></LogFile>
  <LogMask>1</LogMask>
</ServiceListenerConfig>
```

In the paragraphs below each of the configurable options will be explained.

Through the <ListenerPort> configuration option you can set the TCP/IP port to which the RPC service will listen to all incoming web service requests. The default value is port 19339.

The listener port

All files for a single AIMMS web service are automatically placed in a single directory, that you can specify through the web service properties dialog box. However, there is one file for each web service that is stored in the directory located at <ServiceConfigDirectory>, which make all available RPC services known to the RPC service handler. The default location of the service configuration directory is

Web service directories

```
%CommonAppData%\Paragon Decision Technology\ServiceHandler\Services
```

Each service configuration file has the extension .cfg. For web services, the first part of its name is the web service name that you specify in the web service properties dialog box.

Through the <ListenerPort> configuration option you can set the TCP/IP port to which the RPC service will listen to all incoming web service requests. The default value is port 19339.

The listener port

Some RPC services need to open additional TPC/IP ports. The options <ServiceMode> defines how such port will be allocated. Possible values are:

Dynamic port creation

- Dynamic which leaves it up to the operating system to allocate TCP/IP ports
- PortRange which will allocate TCP/IP port from the range <FirstAssignedPort> up to and including <LastAssignedPort>.

The RPC service log various error and events to a log file located at <LogFile>. Its default location is

Log file

```
%CommonAppData%\Paragon Decision Technology\ServiceHandler\Log\ServiceListener.log.
```

In case you have trouble setting up a new web service, it's a good idea to look into this file. You can set the amount of logging by adding the following values for the events and errors you want to have logged, and assign this value to the LogMask configuration option:

- 1 general configuration errors of the RPC service (RPC)
- 2 logging of generic exception occurring in the RPC service (RPCE)
- 4 additional info of the RPC service (RPCI)
- 8 logging of all web service requests (WSL)
- 16 web service related trace of agent technology (WSLA)
- 32 AIMMS related trace of agent technology (AGNT)

For each category, the RPC log file will label log entries with the symbol in parentheses. The categories WSLA and AGNT provide tracing information that

may help Paragon debug errors occurring in the agent technology upon which the AIMMS web service technology is built.

The `ServiceSecurity.cfg` file configures the use of SSL to encrypt all RPC related TCP/IP traffic. This also includes any installed AIMMS web service. The default content of the `ServiceSecurity.cfg` file is:

Contents of ServiceSecurity.cfg

```
<ServiceSecurityConfig>
  <SSLMode>NoSSL</SSLMode>
  <KeyFile></KeyFile>
  <KeyPassword></KeyPassword>
  <CAFile></CAFile>
  <DHFile></DHFile>
</ServiceSecurityConfig>
```

The `<SSLMode>` determines the use of SSL. Possible values are:

SSL mode

- `NoSSL`, no traffic is encrypted using SSL
- `NoAuth`, traffic is encrypted using SSL but neither client nor server is required to authenticate
- `ServerAuth`, traffic is encrypted using SSL and the server is required to authenticate
- `FullAuth`, traffic is encrypted using SSL and both client and server are required to authenticate

When SSL is being used for encryption, the remaining configuration entries of the `ServiceSecurity.cfg` file must be specified as well. They are:

SSL configuration

- `<KeyFile>`, the location of the file containing the private key (in PEM format)
- `<KeyPassword>`, the password use to protect the private key
- `<CAFile>`, this file the X.509 certificates for both this computer (client or server) and of the certificate authority who issued the computer certificate
- `DHFile`, the location of the file containing a Diffie-Hellman parameters (in PEM format) needed to set up the SSL connection.

The next two sections describe how the AIMMS RPC service interacts with the various files that are generated by AIMMS, after filling in the web service properties dialog. This should provide some insight into when it is needed to restart the service and when it is not.

Interaction of the listener and the generated files

Upon startup, the AIMMS RPC service loads the `AimmsWebServiceListener.dll` file for the target web service. It contains important status information for the web service and it starts the queue for the agent community. So, in the event that the actual AIMMS project(s) implementing the web service might terminate in an unusual way (e.g. by a *Fatal Application Error*), it is a good idea

AimmsWebServiceListener.dll

to restart the AIMMS RPC service. When you stop and restart the AIMMS project normally, you don't have to restart it. Also, if the `AimmsWebServiceListener.dll` is changed, after a Software Update of AIMMS, the service needs to be restarted. In this case, AIMMS provides you with instructions when generating the new files for the web service.

The `.cfg` file contains web service related information for the AIMMS RPC service, such as the actual location of the `AimmsWebServiceListener.dll` for the web service, and the name of the service. It is read in upon startup of the Windows service. This means, that when you change the service name or the actual location of the AIMMS project(s) which implement(s) your web service, and generate new files, you must close AIMMS and restart the AIMMS RPC service, in order to reflect your changes.

The .cfg file

This paragraph shows the necessary steps to migrate an existing AIMMS 3.7 web service to AIMMS 3.8 or higher. As an example project, the **CD Playing Time Optimizer** example, which can be found in the

```
Examples\CD Playing Time Optimizer\AIMMS Project
```

subfolder of your AIMMS folder, is used. Please open this project with AIMMS 3.7 and follow the steps on the problem description page, to create a working AIMMS 3.7 web service. When you have tested the web service with the provided client in the

```
Examples\CD Playing Time Optimizer
```

directory, please close the AIMMS project and open it with AIMMS 3.8 (or higher). If you still want to use the project with AIMMS 3.7, you should make a backup first. To migrate, please follow the next steps:

Migrating an AIMMS 3.7 web service to AIMMS 3.8 or higher

- Open the **Web Service Properties** dialog box from the menu **Settings - Multi Agent - Community Setup...**, change the **port** from 23809 into 19339, click on **Generate Files**, and close the dialog box once finished.
- Open the attribute form of the `AutoStartAgent` procedure, remove the contents there and enter the following code:

```
if ProjectDeveloperMode then
    MultiAgent::AutoLogonConnectQueue := "CDQueue";
    MultiAgent::AutoLogonAgentRole := 'ServerRole';
    MultiAgent::AutoLogonAgentName := "CDPlayingTimeOptimizer";

    MultiAgent::AutoLogonAgent;
endif;
```

- Save and close the project (it becomes an AIMMS 3.8 or higher project by doing this).
- Restart the AIMMS RPC service.

- Reopen the project and keep it open.
- Open the client program and change the port in the **URL** field from 23809 into 19339. Then click on **Load input data** and **Optimize playlist**. You now have a working AIMMS 3.8 or higher web service.

AIMMS web services can also be run on a Linux machine. However, since no AIMMS IDE is available yet for this platform, you have to do some steps on a Windows machine to prepare your web service, and there are a few additional manual steps that you have to do under Linux to run your web service successfully.

AIMMS web services under Linux

In order to setup a Linux web service, you have to create the web service project itself on a Windows machine. When providing the data for the **Web Service Properties** dialog box, you can use either the value `localhost` for the **Port** field, or you can already enter a specific Linux host. After specifying all the data (all fields are explained further in this manual), you must click on **Generate Files**, to generate all configuration files for your web service. Please note that the best way to run your web service on Linux, is by means of autostart agents, so make sure you specify this in the list with AIMMS agents. Then, as an extra step, you have to *export* your project, including the multi-agent community setup file. You can do this through the menu action **File - Export**. Please select **To a folder** in the dialog which appears. Then export the project files, the generated web service files and the multi-agent community setup file.

Preparing you Windows AIMMS project

The export step in the paragraph above was necessary to convert your AIMMS files to a new format, that is not only understood by Windows, but also by Linux. Now it's time to copy all the files (your AIMMS files, the multi-agent community setup file and the generated web service files) to the Linux machine. On this machine, you only have to edit the `ServerConfiguration.xml` by hand. Please ensure that you have the lines with the elements `<AgentConfigFile>` and `<AgentProjectFile>` pointing to the Linux paths of the files and not to the Windows paths. As the final preparation steps, you first have to copy the `AimmsWebServiceListener.so` file from the AIMMS Lib directory under Linux to the folder with your generated files, and then you have to copy the `.cfg` file which has been generated into the

Migrating the project to Linux

```
C:\Documents and Settings\All Users\Application Data\Paragon Decision Technology\
ServiceHandler\Services
```

directory under Windows, to the

```
/usr/local/Aimms/ServiceHandler/Services
```

directory under Linux and adapt the path that is specified in this file in the `ServiceDLLPath` element, to make sure it points to the just copied `AimmsWebServiceListener.so` file. Then make sure that the `AimmsServiceHandler` is (re-)started. After these steps, you should be able to call your web service.

4.3 AIMMS web services architecture

The most basic AIMMS web service architecture consists of just one AIMMS agent. This agent is tightly linked to the `AimmsWebServiceListener.dll`, which acts as a special kind of AIMMS agent. Like every other agent, the `AimmsWebServiceListener.dll` implements a specific agent role. This role should be set up to send messages to the linked AIMMS agent. The linked AIMMS agent, which is a real AIMMS project, does the actual work of the web service, by means of its handler procedures. The `AimmsWebServiceListener.dll` takes care of the incoming requests from the Internet and sending any response messages back to the sender(s) of the original requests. Figure 4.2 shows the most basic AIMMS web service architecture, with 3 active web clients.

Basic architecture

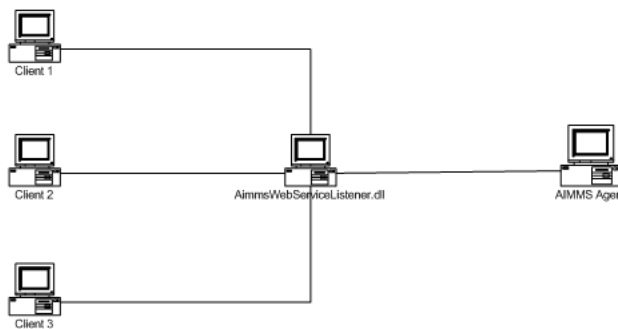


Figure 4.2: The most basic AIMMS web service architecture

AIMMS agents can communicate in one of two modes. They can communicate in *binary* mode, or in *XML* mode. The mode that your agent runs in, must be specified in the `is_xml_agent` argument in the `MultiAgent::Logon` procedure. The AIMMS agent that is linked to the `AimmsWebServiceListener.dll` *must* log in with this argument set to 1. AIMMS already provides a procedure that handles the login for you.

XML vs. binary agents

The most basic architecture, as described above, is only useful in situations in which there is a relatively small number of clients, that only send requests that can be handled relatively quickly by the AIMMS agent. If two clients both send a request to such an AIMMS web service, the client whose request reaches the web service first, is served first. The second client must wait for the other request to be completed, before his request will be handled. This is caused by the fact that only one AIMMS agent is doing the actual work, i.e. there's no parallelism involved in this architecture.

Blocking agent

To overcome the blocking problem in the most basic architecture, the `AimmsWebServiceListener.dll` has been extended with a simple automatic dispatching mechanism. This mechanism allows you to deploy more than one worker agent in your web service. You can specify all worker agents in the web service properties dialog box. Every request that is received in the `AimmsWebServiceListener.dll` will be sent to an available worker agent, i.e. a worker agent that is not currently handling a request. If no available worker agent is available, the request is held in the `AimmsWebServiceListener.dll`, until either a worker agent becomes available, or the request times out. If a worker agent becomes available, the request is automatically dispatched to it. Figure 4.3 shows this architecture, with 3 active web clients and 3 AIMMS worker agents.

A more advanced architecture

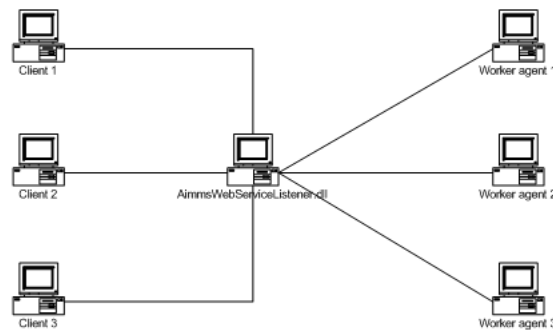


Figure 4.3: The simple automatic dispatching AIMMS web service architecture

If you need a more advanced dispatching mechanism than the one implemented in the `AimmsWebServiceListener.dll`, you can do so by using the tools that the AIMMS multi-agent technology provides for filtering and forwarding incoming messages. As in the most basic architecture, one agent will provide the primary link to the `AimmsWebServiceListener.dll`. However, this agent is not obliged to do all the actual work itself: just let it act as a dispatcher, using the `MultiAgent::FilteredMessageForward` procedure in its filter procedures. If you specify the same `FromAgent` in the procedure call as the `FromAgent` received in the filter procedure, you can transparently let an other AIMMS agent do the actual work. This way, the linked agent only blocks during the relatively short time of dispatching the message to some worker agent. The worker agent can consequently send its response directly back to the `AimmsWebServiceListener.dll` (in the form of output arguments), which, in turn, takes care of sending the response back to the associated client. Figure 4.4 shows this advanced dispatching web service architecture, with 3 active web clients, a dispatching AIMMS agent and 3 worker agents.

An advanced dispatching architecture

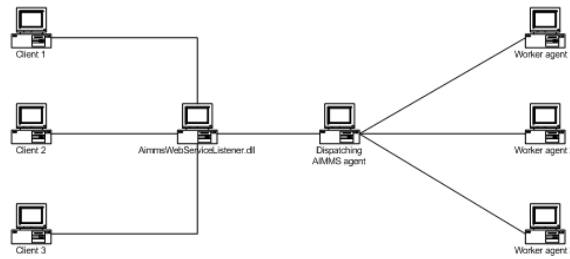


Figure 4.4: An advanced dispatching web service architecture

AIMMS supports two kinds of web services: stateless web services and stateful web services. A stateless web service is a web service that doesn't have to know which client has sent the request. For example, it doesn't matter for a web service that returns the current dollar/euro exchange rate which client has called the service. However, there are situations where it *does* matter. Imagine a very simple web service with a request that counts the number of times a client has called the web service. In that case, it is important to know which client called the web service.

Stateless vs. stateful web services

The automatic dispatching mechanism that AIMMS offers in the `AimmsWebServiceListener.dll`, can also be used in combination with stateful web services. However, for efficiency, a slightly different dispatching scheme than the one described above will be used. For every working agent, the `AimmsWebServiceListener.dll` remembers what was the last served client by this agent. If a request comes in, the dispatching mechanism now checks whether the worker agent that has last served the client which sent the request, is available. If so, it is assigned the request. This offers the advantage that the worker agent doesn't have to store and retrieve the state of this client. However, if the worker agent that last served the client is not available, the request will be sent to another available worker agent. If no worker agents are available, the request is queued in the `AimmsWebServiceListener.dll` until either a worker becomes available, or the request times out.

Stateful web services with automatic dispatching

AIMMS supports yet another automatic dispatching feature: the handling of web service requests as an *exclusive session*. This means, that stateful clients can log on to the web service with an extra argument. This has the effect that a single worker agent is assigned to that client and it is guaranteed that this worker agent exclusively works on requests for this client alone. That eliminates the need for state preserving and recovery, making the handling more efficient. However, the downside of this is that other clients cannot use the exclusively assigned worker agent, even if it is idle sometimes. When logging off the client, the worker agent is automatically stopped by the dispatching

Stateful web services with exclusive session dispatching

mechanism and becomes available for other clients again.

In the web service properties dialog box, you can specify whether a web service should be stateful. When calling such a stateful AIMMS web service, the generated WSDL document specifies a *SOAP header* which contains an Agent-Name element. Also, the WSDL file specifies a Logon message. The first time that you connect to a stateful AIMMS web service, you must send a Logon message, identifying yourself. The Logon message looks like the following in the WSDL file:

SOAP header

```
<element name="LogonRequest">
  <complexType>
    <sequence>
      <element name="AgentName" type="xsd:string"/>
      <element name="AgentRole" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="Uri" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <element name="ExclusiveSession" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    </sequence>
  </complexType>
</element>
```

Only the first argument, the AgentName is mandatory. The other arguments will be discussed below. So, to make your client known to the web service, you have to specify an agent name for it. In all successive calls to the web service, the client must specify the same agent name in the SOAP header. This way, the AIMMS web service always knows which agent sent the request. In the AIMMS project, requests associated with various clients can be identified by their corresponding from_agent argument in the handler procedure. The next two arguments must be specified if you want to setup your client for *two-way communication* (see below), or if you want to use the exclusive session dispatching mechanism. The valid values for the ExclusiveSession argument are Yes and No. In the latter case, you can also omit this argument from the call.

Based on the from_agent argument, you can take appropriate action in your AIMMS project. For example, you can load a case for the session associated with the current from_agent, before actually handling the request. Please note that it is the modeler's responsibility to actually implement a meaningful mechanism to handle session state: AIMMS just provides you with the information that you need for the task. If you intend to use AIMMS cases to store and retrieve session state, a good location to perform the necessary case management actions would be a filter procedure which is called immediately prior to calling the actual handler procedure. Two functions that may come in handy are CaseWriteToSingleFile and CaseReadFromSingleFile (see the User's Guide for details).

Responsibility

The architecture of AIMMS web services allows for two-way communication. This means that asynchronous communication is possible between web clients and an AIMMS web service. It is possible to initiate the sending of a message from an AIMMS agent to a web client. Such a web client must first send a Logon message to the AIMMS web service. Such a message is always generated in the WSDL file. The first argument of the message is the agent *name* that the web client will operate under. The second argument is the agent *role* it will play. The third argument is the URI of the web client. The AIMMS agent should send its requests to this URI. If you have logged on your external web client, you can send requests to it from your AIMMS project, by simply creating a send procedure from the multi-agent implementation dialog box and calling this send procedure with the agent *name* specified for the `to_agent` argument.

Two-way communication

As mentioned before, it is possible to use AIMMS itself as a web service client. This makes it possible for two (or more) AIMMS projects to communicate with each other over the Internet. Both AIMMS projects should use the same multi-agent community setup file (more precise: two copies of the same file, since the projects will typically be located in different physical locations). In this scenario, it's also necessary that the AIMMS agents make themselves known to the other agent, by sending a logon request. To make this possible, AIMMS offers the standard procedure `WebServiceLogon` in the multi-agent module (and the corresponding procedure `WebServiceLogoff`. There are a few extra arguments compared to the logon message in the WSDL:

Using AIMMS as a web service client

- `endPoint`: the URI of the AIMMS web service to which the AIMMS agent wants to log on;
- `serviceName`: the service name of the AIMMS web service to which the AIMMS agent wants to log on;
- `uri`: the URI of the AIMMS web service, through which the AIMMS agent that is currently logging on, can be reached back.
- `exclusiveSession`: specifies whether the calling AIMMS project will have exclusive access to the called AIMMS web service.
- `info`: an output string argument in which status info will be received after the actual call.

WebService-Logon (extra) arguments

The WSDL document always specifies a SOAP header for your web service, whether the web service is stateful or not. The header always contains a `Timeout` element. You can set this timeout to the number of seconds that you are prepared to wait for a response to your web service request. If you don't specify a value (i.e. you specify the value 0), the default multi-agent timeout of 30 seconds will be used. If a timeout occurs, you still receive a response of the web service at the end of the timeout period, telling you that a timeout has occurred.

Timeout

For AIMMS agents which are started automatically by the web service listener (as specified in the web service properties dialog box), there are two ways in which these agents can terminate. The first way is to make use of the exclusive session dispatching scheme, explained earlier. In that case, if you log off your client, its associated agent is terminated automatically. The second way in which an agent can terminate is when it hasn't received any requests for a specified period (default 15 minutes). Then the dispatching mechanism automatically terminates it for you. Clients are automatically logged off when the web service listener hasn't received any calls from it for a specified period (which defaults to 30 minutes).

*Worker agent
lifetime
management*

4.4 The Web Service Properties dialog box

To expose an AIMMS project as a web service, you must fill in the dialog box under **Settings - Multi-Agent - Community Setup/Webservice Properties**. After filling all required fields, clicking on **Generate Files** generates and copies all necessary files to the directory that you specified in the dialog box. The dialog box is shown in Figure 4.5.

*Webservices
dialog box*

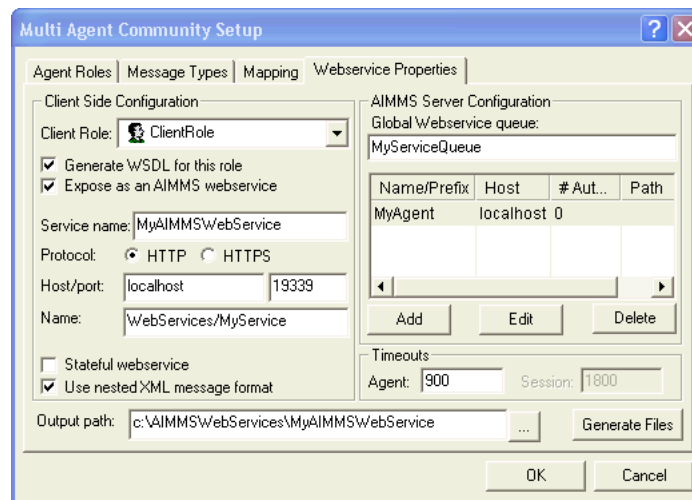


Figure 4.5: The Webservice Properties Dialog box

The dialog box is split up in two main groups, marked **Web service configuration** and **AIMMS Agent Configuration**. The first group contains information about the `AimmsWebServiceListener.dll`, while the second group contains information about the corresponding AIMMS agent project(s), that will be linked to the web service listener.

*Web service
listener vs.
agent project*

In the dialog box, select the agent role that you want to expose as a web service from the *Agent Role* dropdown list. This list only contains the agent roles that you've selected in the Agent Roles tab. Keep in mind that the agent role you select from the dropdown list must correspond to the agent role that the `AimmsWebServiceListener.dll` will play in the multi-agent community, not to the role that the AIMMS agent project will play!

Agent Role

You must check the check box, labeled *Generate WSDL for this role*, if you want the currently selected role either to be exposed as an AIMMS web service, or to act as an external client of an AIMMS web service, that can handle incoming requests by itself. In the latter case, the external client is a web service on its own and doesn't have to be implemented in AIMMS. Any other language that is able to create web services will do. Obviously, you must adhere to the WSDL file that AIMMS will generate for this role when you implement your external client.

Generate WSDL for this role

You must check the check box labeled *Expose as an AIMMS web service* if you want the currently selected role to act as an AIMMS web service. Please remember to check this box for the right agent role: you should not use the role that actually handles the request, but the role that is linked with the `AimmsWebServiceListener.dll`.

Expose as an AIMMS web service

The *Service name* field allows you to specify a name for the web service. If you do not specify a name explicitly, a default name is filled in automatically.

Service name

The four fields labeled *Protocol*, *Host*, *Port* and *Name* let you specify the actual *URI* for the AIMMS web service that you're creating. The resulting *URI* will be stored in the WSDL file after you click on the *Generate Files* button. The *URI* is constructed from these four fields as follows:

*Protocol - Host
Port - Name*

```
<Protocol>://<Host>:<Port>/<Name>
```

So, for example, the values specified in the fields in Figure 4.5 will result in the following *URI*:

```
http://localhost:19339/Webservices/MyService
```

The *Protocol* radio buttons let you either choose a *HTTP* service or a secure *HTTPS* service. The *Host* field lets you specify the host at which the AIMMS web service will be deployed. The *Port* field is optional. If you specify nothing for it, the default port of the host on which the web service will be deployed will be used. Always make sure that the port that you specify here, matches the port in the `ServiceListener.cfg` file in the

```
%CommonAppData%\Paragon Decision Technology\ServiceHandler
```

directory. The default port used for AIMMS web services is taken from this file (its value is 19339 after the AIMMS installation) and is filled in by default in the Port field. The Name field lets you specify the rest of the URI. It is allowed to use forward slashes in this field, as you can see in the example above.

The checkbox labeled *Stateful web service*, lets you specify whether you want your web service to be stateful or stateless. As described above, this influences the format of the SOAP header defined in the generated WSDL document.

Stateful web service

If you check the check box labeled *Use nested XML message format*, the XML format that your AIMMS web service uses for communicating is more compact than when you don't check this check box. The XML format used is reflected in the WSDL file, in messages that have higher dimensional arguments. Using the compact format has the obvious advantage that the SOAP messages are shorter. However, there's a downside: when programming a client to call your web service, it proves much easier to do so when not using the compact format.

Use nested XML message format

The *Queue* field lets you specify the name of the queue through which the AIMMS multi-agent projects can connect to the listener DLL. Please note that the `AimmsWebServiceListener.dll` creates this queue when the AIMMS RPC service is (re)started.

Queue

In the *AIMMS agents* field, you must specify the agent name(s) (not the agent *role(s)!*) that the AIMMS project(s) specifies when it logs on to the agent community. You can enter more than one agent, in case you want to use the automatic dispatching mechanism that AIMMS offers. With the *Add*, *Edit* and *Delete* buttons, you can specify a list of agents. If you click on *Add* or *Edit*, or double click on a line in the list, a dialog pops up, which lets you specify the agent details (see Figure 4.6). Please note that you should make sure that all agents that you specify in this dialog box, are indeed capable of handling *all* requests that are allowed to be sent from a client to the AIMMS web service. I.e.: make sure that the message mapping for the *roles* that each of these agents fulfills, states that all messages that can be sent from a client role can indeed be received by the all the possible roles for these agents.

AIMMS Agents

AIMMS agents that act as worker agents for your web service can either be manually started or automatically. Typically, when designing a new web service, you use manually started agents, since you can actually see them on your screen. In a deployment situation however, it's much more convenient to use automatically started agents. They are started (and terminated) when appropriate, and logged on automatically by the web service listener. In this dialog you can specify 1 manually started agent by entering a value of 0 in the *Nr. of autostart agents* field, or one or more automatically started agents by entering a value greater than 0 in the *Nr. of autostart agents* field. The next four

Manually vs. automatically started agents

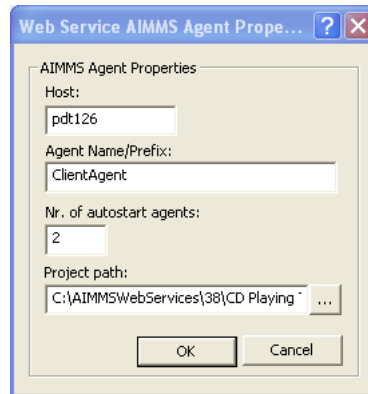


Figure 4.6: The Webservice AIMMS Agent Properties Dialog box

paragraphs describe the fields found in this dialog in detail.

In the edit box labeled *Host*, you must specify the name of the host where the (worker) agent(s) will run. Note that it's possible to have a web service with worker agents running on different machines, to enable real parallelism in your AIMMS web services. Please note that you must only specify the machine name without its domain. E.g. *MyMachine*, instead of *MyMachine.paragon.nl*, if your full machine name as displayed by Windows is *MyMachine.paragon.nl*. The value *localhost* is allowed.

Host

The edit box labeled *Agent Name/Prefix* has two possible meanings, depending on the value you enter in the *Nr. of autostart agents* field. If a value of 0 is entered there, the value for the *Agent Name/Prefix* denotes the name of the (manually started) agent. If a value greater than 0 is entered, the value for the *Agent Name/Prefix* denotes the prefix that will be used for the agent name under which the automatically started agents will be logged on. The prefix will be extended with a unique number, starting with 1.

*Agent
Name/Prefix*

Setting this field to 0, means that you're specifying one single manually started agent in the dialog. The *Agent Name/Prefix* field then denotes the actual name of this agent. Setting this field to a value of 1 or greater means that you're specifying one or more automatically started agents. In that case, the *Agent Name/Prefix* field denotes the prefix for the name(s) of these agent(s). Specifying automatically started agents, means that the *AimmsWebServiceListener.dll*, upon receiving its first request, should automatically start up the linked AIMMS multi-agent project, if that hasn't been started already. Please note that the AIMMS project will be started up in server mode, so you will not see it in your Windows task bar (in the Windows task manager, however, an *AimmsServiceHandler.exe* entry appears for each agent project). If anything goes wrong during

*Nr. of autostart
agents*

the automatic startup, a response is sent to the client that sent the request, indicating the failure. In case you set more than one (worker) agent to be automatically started, each worker agent is only started at the moment at which it is actually needed. For example, suppose that you have a project with two (identical) worker agents. Upon the first request, the first worker agent will be started. If this request has been fully handled when the second request arrives, this second request will also be sent to the first worker agent! Only if the first worker agent is still busy handling the first request upon arrival of the second, the second worker agent will be started to serve the second request.

By entering a positive value in the *Nr. of autostart agents* field, a new field is displayed. The field is labeled *Project path*. In this field, you must specify the absolute path and filename of the linked AIMMS multi-agent project. When you specify an agent on a different machine (through the *Host* field), you must enter the absolute path relative to this other machine. For example, if you will deploy the agent on a machine named *OtherMachine*, in the path *C:\Webservices\TheService*, you should fill in *C:\Webservices\TheService* in this field, and not *\\OtherMachine\C\Webservices\TheService*.

Project path

Note that when you specify more than 1 automatically started agent, multiple instances of the *exact same* AIMMS *project* will be started, from the location that you specify. If you want different worker AIMMS projects to act as automatically started agents, you should enter them individually (so, fill in this dialog for each agent, setting the value of *Nr. of autostart agents* to 1 each time).

In the *Output path* field, you must specify the path, into which you want to install your web service. All files that are generated when clicking on *Generate files* will be stored in that directory (except for the *.cfg* file, as described before). Note that the actual AIMMS project that implements your web service doesn't have to be located in the same directory.

Output path

In the *Timeouts* part of the dialog, the field *Agent* specifies the number of seconds that an active worker agent remains alive after its last handled request. After this timeout, the associated agent process is automatically killed. The *Session* field specifies the number of seconds that a logged on session agent in a stateful web service remains logged on, after this session agent has sent its last request. After this timeout, the session agent is automatically logged off.

Timeouts

If you have filled in all needed fields in the web service properties dialog box, the **Generate Files** button will be enabled. By clicking on this button, 3 files will be put into the directory that you specified in the *Output path* field:

Generate Files

- The WSDL document;
- A copy of *AimmsWebServiceListener.dll*;
- The *ServerConfiguration.xml* file (see below).

In addition, a file called <ServiceName>.cfg is stored in the

```
%CommonAppData%\Paragon Decision Technology\ServiceHandler\Services
```

directory. For details, see below.

The generated file `ServerConfiguration.xml` contains all the information that the `AimmsWebServiceListener.dll` needs for starting the queue, setting up a connection with the AIMMS multi-agent community and communicating with it. Its contents are mainly based on what you specified in the web service properties dialog box. If you have changed anything in the web service properties dialog, this change will be reflected in the `ServerConfiguration.xml` file. You need to restart the AIMMS RPC service, in order to apply the changes to the web service.

*ServerConfig-
uration.xml*

The file <ServiceName>.cfg, contains the URI and the physical path of the AIMMS web service. For each web service that you create, one such file is generated. The purpose of the file is to provide the AIMMS RPC service with information about what web services are active on the current machine. So, if you want a web service to be deployed on another machine than the one on which you designed it, you should move the corresponding .cfg file to the

.cfg file

```
%CommonAppData%\Paragon Decision Technology\ServiceHandler\Services
```

directory of the target machine and restart the AIMMS RPC service. Better yet, you can just open the web service properties dialog box of the AIMMS web service project on the target machine and click `Generate Files`.

If you want to create a secure external web service, to which AIMMS can send requests, you must provide extra information in the web service properties dialog. First, you have to specify that the current role will be implemented by an external web service, by checking `Generate WSDL for this role` checkbox and unchecking the `Expose as an AIMMS web service` checkbox. If you do so, you'll notice that the dialog changes as shown in Figure 4.7, in order to show three new fields.

*Secure external
clients...*

If you don't fill in the new fields, AIMMS will assume that the external web service is a non-secure web service. If the external web service is a secure web service however, the other fields must be entered, depending on the web service's authentication mode. If the external web service offers server authentication, only the `Key/Cert.` and the `Password` fields have to be specified. If the external web service also requires client authentication, AIMMS needs to know which CA (Certificate Authority) certificate file to send along with the requests, so in that case you also have to specify the `CA Cert.` field. Both the `Key/Cert.` and the `CA Cert.` fields accept a file with the extension `.pem`, while the `CA Cert.` field also accepts files with the extension `.crt`.

...continued

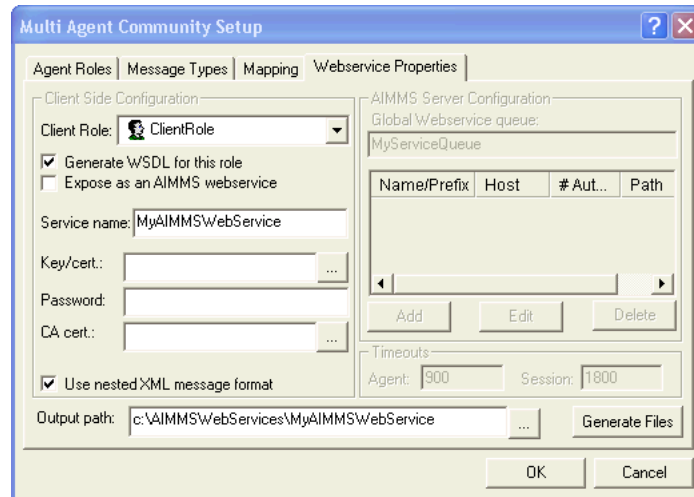


Figure 4.7: The Webservice Properties Dialog box for external web services

Note that the Use nested XML message format check box is also present for external web services (whether secure or not). By checking this check box you can specify that your external web service understands the nested XML format specification. If unchecked, the non-nested format must be understood by the external web service.

Nested XML message format

4.5 Step-by-step guide for setting up a basic web service

The objective of this section is to show you, step-by-step, how to create, install and call a basic AIMMS web service. The web service that is created in this guide will contain two messages: one to return the square root of the input argument and one that uses attachment arguments.

Objective

As the very first step for creating an AIMMS web service, you have to create a new AIMMS multi-agent project. You can do this by performing the following steps:

New AIMMS agent project

- ▶ Create a new AIMMS project;
- ▶ Select **Settings/Multi Agent/Install Agent Module** from the menu; and
- ▶ Select **Settings/Multi Agent/Select community Setup File...** from the menu and specify a name for your community setup file.

Now it's time to define two roles for your project. To do this, select **Settings/Multi Agent/Community Setup...** from the menu. The **Multi Agent Community Setup** dialog box appears, with the **Agent Roles** tab active. Add two roles here: `ExampleServiceHandler` and `ExampleServiceAIMMSProject`. The first role is the role that the `AimmsWebServiceListener.dll` will play, the second is the role that the AIMMS agent project will play.

Define roles

Next, we'll define the two messages that will implement the functionality of our web service-to-be. Activate the **Message Types** tab and add two messages, as shown in figure 4.8. Obviously, the I/O type of the `InputFile` argument must be set to *input*, whereas the I/O type of the `OutputFile` argument must be set to *output*. Any comments that you type for the message types and arguments will automatically appear in the generated WSDL file later.

Define message types

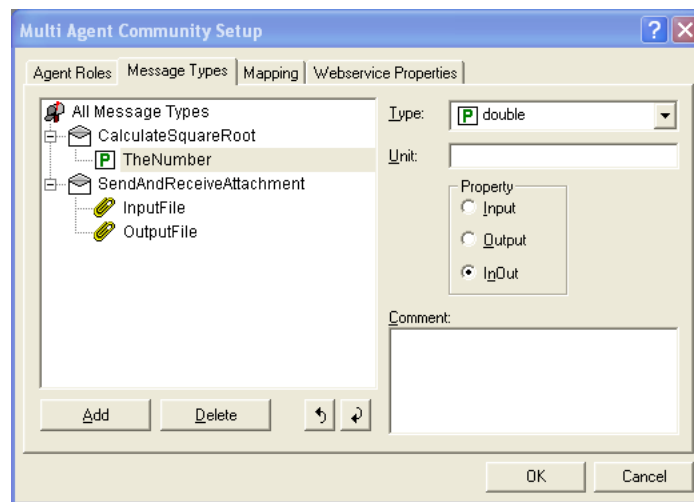


Figure 4.8: Message types for the example AIMMS web service

Now the two messages have been defined, we need to set the message mapping. To do this, activate the *Mapping* tab and, for both messages, select `ExampleServiceHandler` as the From Agent Role and `ExampleServiceAIMMSProject` as the To Agent Role. Then click OK and save your work.

Define message mapping

Select **Settings/Multi Agent/Agent Setup...** from the menu. The **Agent Roles** tab will be active. Please select the `ExampleServiceAIMMSProject` role here, and activate the **Receiving** tab. This tab will show both messages, since the `ExampleServiceAIMMSProject` role was configured before to receive both messages. Now, select both messages, one at a time, and check the **Message Handler**

Create handler procedures

checkbox for both of them. Answer **Yes** to the questions that AIMMS will ask you regarding the handler procedures.

Now it's time to specify the web service-related properties. Please click on **OK** first to close the dialog box and save your work. Activate the Settings/Multi Agent/Community Setup dialog box and activate the **Webservice Properties** tab. Select the `ExampleServiceHandler` role from the dropdown list. Please fill in the dialog box exactly as shown in figure 4.9. For the **Host** field (which is filled in with `localhost` in the figure), in the pop-up dialog for the AIMMS agents, you have to fill in your own machine name on the network, your IP-number, or simply `localhost`. If you forget to do so, you'll receive a warning upon clicking the **Generate Files** button. Make sure you only specify your machine name without its domain, so, for example, if your full machine name is `My-Machine.paragon.nl`, only specify `MyMachine`. Obviously, you can select your own output path in the *Output path* field. To show the AIMMS agents pop-up dialog, click on the *Add* button below the *AIMMS Agents* list.

Specify web service properties

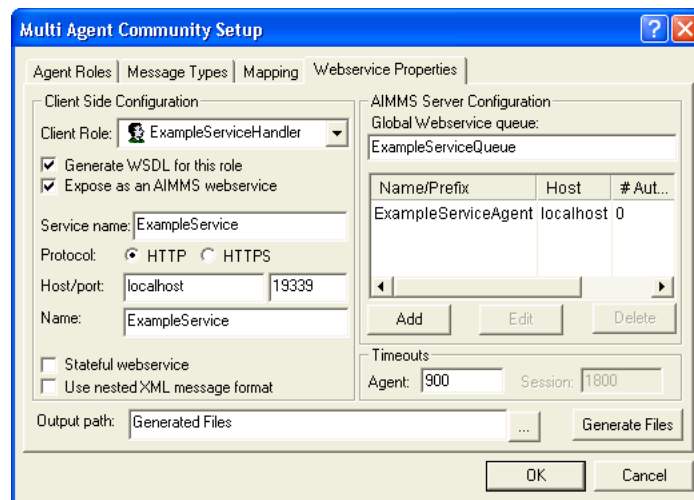


Figure 4.9: Properties for the example AIMMS web service

At this time you're ready to generate the necessary files for your web service. Click on the **Generate Files** button. If any error message appears, please follow the instructions in the error message. If all goes well, AIMMS shows a message "Files generated successfully". You can verify in the *Generated Files* subdirectory of the directory where you located your project, that the files were actually generated. Obviously, if you specified your own output path, you should see the files there. Click on **OK** and save your work.

Generate files

As you can see, AIMMS has created two handler procedures for you in the Multi Agent Handlers section in the model explorer. Please implement the Handler_CalculateSquareRoot procedure as

```
TheNumber := Sqrt(TheNumber);
```

and the Handler_SendAndReceiveAttachment procedure as

```
FileCopy( inputFile, "c:\\CopiedPic.jpg");
OutputFile := "c:\\CopiedPic.jpg";
```

Please note that a line like `OutputFile := InputFile;` will not work, since all input attachments are automatically deleted upon leaving the handler procedure, hence deleting the output attachment as well in this case.

As described in the Web Service documentation preceding this step-by-step guide, AIMMS Web Service projects need to be logged on to their agent community. AIMMS provides a standard procedure called `AutoLogonAgent`, in the Auto Logon Support section, which is located in the Multi-Agent module. This procedure works both when the project will be used as an automatically started agent, or as a manually started agent. Since the Web Service that you're developing right now will be used as a manually started agent, you have to make sure that this procedure is called automatically upon startup of the project. Please do this, by calling it in a new procedure, called `AutoLogon`. You also have to specify the values that will ensure a correct logon by the `AutoLogonAgent` procedure. Please implement `AutoLogon` as follows:

```
if ProjectDeveloperMode then
  MultiAgent::AutoLogonConnectQueue := "ExampleServiceQueue";
  MultiAgent::AutoLogonAgentRole := 'ExampleServiceAIMMSProject';
  MultiAgent::AutoLogonAgentName := "ExampleServiceAgent";

  MultiAgent::AutoLogonAgent;
endif;
```

By checking whether the project is run in developer mode (in the `if` statement), you enforce that this code will only be executed when the project is started manually. After doing this, save your work and close the AIMMS project. Please note that when you're only using the project as an automatically started agent, you don't have to provide this implementation of `AutoLogon`. Instead, the `AimmsWebServiceListener.dll` fills in this information for you automatically when starting up the agent. However, it is good practice to provide this implementation of `AutoLogon`, since with it, you have both the manual and the automatic case covered. When having written this procedure, you must set it as the startup-procedure of your AIMMS project (menu: **Settings - Project Options...**, option **Project - Startup & authorization - Startup procedure**).

Implement the handler procedures

The AutoLogon-Agent procedure

At this time, the implementation part of your AIMMS web service is complete. The only thing you have to do now, is restarting the AIMMS RPC service in order to deploy the web service. After this, you have a working and installed AIMMS web service.

No installation needed

Your web service is ready to be called. To do so, you have various options. You can use an external tool, write a Visual Basic Script, or use the `wsd1.exe` tools that come with .NET. Another possibility is using gSOAP to generate a C or C++ client to call your web service. The AIMMS web service functionality is actually implemented using gSOAP. For more information regarding gSOAP, see <http://www.cs.fsu.edu/~engel/soap.html>.

Calling the web service...

You can use an external tool (e.g. Altova's *XMLSpy*, or the open source program *SoapUI*), to generate a SOAP request for you and send the request to the web service. However, most tools don't support the sending and receiving of attachments.

...with an external tool

If you've installed the Microsoft SOAP toolkit, you can use the small Visual Basic Script `SendAndReceiveAttachment.vbs` that comes with AIMMS, to call the `SendAndReceiveAttachment` method of the web service. This small script passes a picture to the web service, and receives it back in the directory from which it is executed as `ReceivedAttachment.jpg`. To obtain this script, please unpack the `.aimmspack` file in the `Examples\Web Service Caller` directory of your AIMMS installation. The script is then located in the `Web Service Caller Files` subdirectory of your unpack target directory.

...with Visual Basic Script

Another possibility is to use the `wsd1.exe` tool or one of its successors, which come with Visual Studio .NET. This tool automatically generates a file in your favorite .NET programming language for you, which encapsulates all details of calling your web service. In the case of our basic example web service, it provides you with just two single functions for calling the web service requests. The next sections show you step-by-step how you can generate and implement a working client for your AIMMS web service using the .NET environment. Note that we've included these sections to illustrate how straightforward it is to generate a client. The other environments mentioned offer a similar ease of use for converting a WSDL file into a working client.

...from .NET languages

In this section, you can find the steps that you need to perform, in order to generate and implement a working client for your AIMMS web service. It makes use of the `wsewsd12.exe` tool, which comes with WSE 2.0, which stands for *Web Service Enhancement*. You can download and install WSE 2.0 from the Microsoft web site

Generate a working client

<http://msdn2.microsoft.com/en-gb/webservices/Aa740663.aspx>

Please note that the current version of WSE is 3.0. We have decided to create this example using the older version, because WSE 2.0 can also be used with older versions of Microsoft Visual Studio than version 2005. WSE 3.0 cannot be used for this example, since the client code that it generates is different from that of WSE 2.0. Also, WSE 3.0 doesn't support the attachment type that AIMMS web services expect. So, beside WSE 2.0, you also need .NET and Microsoft Visual Studio .NET 2003 or higher installed on your system in order to successfully follow the steps in this section. For your convenience, the complete Visual Basic project is included in the Examples directory of AIMMS. However, if you're serious about learning how to write an AIMMS web service client, we strongly advise you to try to create the project yourself. To do so, and if all the prerequisites are properly installed, please follow the following steps carefully:

- ▶ Copy the `wsewsdl2.exe` file to the output path that you specified in the **Web Service Properties** dialog box. Normally, this file can be found in `C:\Program Files\Microsoft WSE\v2.0\Tools\Wsd1`
- ▶ Open a DOS command prompt in this path
- ▶ Type:

```
wsewsdl2.exe ExampleService.wsdl ExampleService.vb VB
```

This step automatically generates the Visual Basic source file `ExampleService.vb` for you.

- ▶ Start Microsoft Visual Studio .NET 2003 or higher
- ▶ Create a new Visual Basic project, of type *Windows Application*. Name your project and open the solution explorer.
- ▶ Right-click on the project name and select **Add reference...** Please add references to the .NET components `Microsoft.Web.Services.2`, `System.Web` and `System.Web.Services`.
- ▶ Move the generated file `ExampleService.vb` to your Visual Basic project directory and add it to the project by right-clicking on the project in the solution explorer and selecting **Add/Existing item...**
- ▶ Try to build your project. This will give errors regarding the `SendAndReceiveAttachment` request. These are caused by the fact that this request only accepts an attachment argument, which results in an entry in the WSDL file of a message without any arguments. `wsewsdl2.exe` doesn't handle these messages well. In order to correct the problem, simply paste the following code in the file `ExampleService.vb`, just below the definition of class `CalculateSquareRootResponse`:

```
<System.Xml.Serialization.XmlTypeAttribute([Namespace]:= _
"http://www.aimms.com/generated_namespaces/ExampleService/xsd"), _
```

```

System.Xml.Serialization.XmlRootAttribute([Namespace]:= _
    "http://www.aimms.com/generated_namespaces/ExampleService/xsd", _
    IsNullable:=False)> _
Public Class SendAndReceiveAttachmentRequest
End Class

<System.Xml.Serialization.XmlTypeAttribute([Namespace]:= _
    "http://www.aimms.com/generated_namespaces/ExampleService/xsd"), _
System.Xml.Serialization.XmlRootAttribute([Namespace]:= _
    "http://www.aimms.com/generated_namespaces/ExampleService/xsd", _
    IsNullable:=False)> _
Public Class SendAndReceiveAttachmentResponse
End Class

```

As you can see, this code is merely a copy of the code for the other request and response, only without any class members (i.e. without arguments).

- ▶ Double-click on the Form1.vb file in the solution explorer.
- ▶ Add two text boxes to the form: one named Input and one named Output.
- ▶ Add one button to the form, label it SQRT and name it SQRTButton.
- ▶ Double-click on the button to view the code of its Click event. In the sub that is shown, add the following code:

```

Dim WebService As New ExampleService
Dim SQRTRequest As New CalculateSquareRootRequest
Dim SQRTResponse As New CalculateSquareRootResponse

SQRTRequest.TheNumber = Val(Input.Text)
SQRTResponse = WebService.CalculateSquareRoot(SQRTRequest)
Output.Text = Str(SQRTResponse.TheNumber)

```

This code declares a web service object, and a request and a response object. Then, it simply assigns the value that the user enters in the **Input** text box to the **TheNumber** argument of the request, sends the request to the AIMMS web service and stores the result in the **Output** text box. This is all that it takes to call a simple AIMMS web service.

- ▶ Make sure that your AIMMS project is running in order to process the web service requests that you'll send it using your client
- ▶ Build your project and run it. Fill in a value in the **Input** text box and click on the **SQRT** button. If all goes well, the **Output** text box will show the square root of the value you entered, calculated by the AIMMS web service.

An extremely handy tip for debugging any AIMMS web service is to go to **Settings/Project Options...** in your AIMMS project and open the **Aimms/External functions** node. Set the option **External message filter** to the value **All**. If you show the message window in AIMMS (CTRL+M), it will show a line when a multi-agent message is either received, filtered, handled or sent. If anything goes wrong with your AIMMS web services, you can quickly see at what stage it happens.

Debugging tip

So far, you've only called the `CalculateSquareRoot` message of your AIMMS web service. If you want to learn how to send and receive attachment arguments, please follow the following steps:

Extending the client project

- ▶ Add two objects of type *Picture box* to your form (adjust the size of the form first, if needed). Call one of them `InputPicture` and the other `OutputPicture`.
- ▶ Set the **BorderStyle** property of both to `FixedSingle` and the **SizeMode** property to `StretchImage`.
- ▶ Add a button labeled **Copy Image** and call it `CopyImageButton`.
- ▶ At the top of the file `Form1.vb` (if you're using Visual Studio .NET 2003), or `Form1.Designer.vb` (if you're using Visual Studio 2005), please add the following declaration, just below the line which reads `Inherits System.Windows.Forms.Form`:

```
Private InputAttachment As String
```

This is needed for remembering the file name of the input attachment across the various subs.

- ▶ In file `ExampleService.vb`, directly under the line which reads `Inherits SoapClient`, add the following declaration:

```
Public InputAttachment As String
```

This is needed for setting the file name of the input attachment in the class, based on the picture that the user selects.

- ▶ Double-click on the **InputPicture** picture box, in order to add a click event to it. In the click event, enter the following code:

```
Dim Dlg As New OpenFileDialog
Dlg.Filter = "Image Files (*.jpg)|*.jpg"

Dim Result As DialogResult = Dlg.ShowDialog()
If Result <> Windows.Forms.DialogResult.OK Then
    exit Sub
End If

InputPicture.Image = Image.FromFile(Dlg.FileName)
InputAttachment = Dlg.FileName
```

This code shows a dialog to the user in which he can select a JPG image file. Also, the image is displayed in the form and remembered.

- In the file `ExampleService.vb`, add a `FilterMessage` sub to override the default `FilterMessage` sub. To do so, add the following code below `End Sub` line of the `Sub New()` sub:

```
Protected Overrides Sub FilterMessage(ByVal envelope As SoapEnvelope)

    If envelope.Context.Addressing.Action.ToString =
        "ExampleService#SendAndReceiveAttachment" Then
        'Send an attachment along with the request.
        Dim attachment As New Microsoft.Web.Services2.Dime.DimeAttachment _
            ("image/jpeg", Microsoft.Web.Services2.Dime.TypeFormat.MediaType, _
            InputAttachment)
        envelope.Context.Attachments.Add(attachment)
    End If

    'If you also want to pass a Timeout value in the header of the
    'SOAP message, please uncomment the lines below.
    'envelope.createHeader()
    'envelope.header.innerXML = "<m:ConnectionInfo xmlns:m=" + chr(34) _
    ' + "http://www.aimms.com/generated_namespaces/ExampleService/xsd" _
    ' + chr(34) + "><Timeout>100</Timeout></m:ConnectionInfo>"

    MyBase.FilterMessage(envelope)
End Sub
```

The `FilterMessage` sub will be called automatically, just before the actual message will be sent to the web service. This gives you the opportunity to add the actual attachment to the message. Please note the lines commented out regarding the `Timeout` value: there is a bug in WSE 2.0 with sending an extra header along with a SOAP request. The code commented out code shows you the workaround. This bug has been fixed in WSE 3.0, but, as stated before, WSE 3.0 doesn't support the attachment type which AIMMS expects.

- In `ExampleService.vb`, locate the line which reads

```
<SoapMethod("ExampleService\#SendAndReceiveAttachment")>
```

This line, and the `SendAndReceiveAttachment` sub below it, must be *replaced* by the following code:

```
<SoapMethod("ExampleService#SendAndReceiveAttachment")> _
Public Function SendAndReceiveAttachment( _
    ByVal request As SendAndReceiveAttachmentRequest, _
    ByRef at As System.IO.Stream) As SendAndReceiveAttachmentResponse

    Dim env As SoapEnvelope
    env = MyBase.SendRequestResponse("SendAndReceiveAttachment", request)

    Dim NumberOfAttachments As Integer
    NumberOfAttachments = env.Context.Attachments.Count
```

```

Dim Attachment As Dime.DimeAttachment
Attachment = env.Context.Attachments.Item(0)
at = Attachment.Stream 'For returning

Return CType(env.GetBodyObject(GetType(SendAndReceiveAttachmentResponse), _
    SoapServiceAttribute.TargetNamespace), _
    SendAndReceiveAttachmentResponse)
End Function

```

This code replaces the original code for sending the `SendAndReceiveAttachment` request. It handles the receiving of the attachment in the response message. As you can see, you can determine the number of attachments in the response automatically. This code 'knows' that there will only be 1 output attachment.

- Double-click on the **Copy Image** button, to create its click event. Enter the following code for this event:

```

Dim Webservice As New ExampleService
Dim AttachmentRequest As New SendAndReceiveAttachmentRequest
Dim AttachmentResponse As New SendAndReceiveAttachmentResponse

Webservice.InputAttachment = InputAttachment
Dim TmpStream As System.IO.Stream

AttachmentResponse = _
    Webservice.SendAndReceiveAttachment(AttachmentRequest, TmpStream)
OutputPicture.Image = Image.FromStream(TmpStream)

```

This code is very much like the code you wrote earlier for the **CalculateSquareRoot** request: it declares a request, a response and a web service object, calls the AIMMS web service and handles the response.

- Finally, build your project, run it, double-click on the input picture box, pick a file and click on **Copy Image**. If all goes well, the image file is copied by your AIMMS web service.

AIMMS comes with another example: the **CD Playing Time Optimizer** web service. This web service can be used to calculate the best distribution of songs to burn on CD's. The model in the web service tries to minimize the waste of recording time (for details, see the model itself). This web service and the corresponding AIMMS project can be found in an `.aimmspack` file in the `CD Playing Time Optimizer` directory in the `Examples` directory. The next section will show you how straightforward it is to install the web service on your own machine.

*CD Playing
optimizer
example...*

To install the **CD Playing Time Optimizer** web service, please perform the following steps:

...and how to install it

- ▶ Double-click the .aimmspack file in the **Examples\CD Playing Time Optimizer** folder
- ▶ Select a location on your hard drive to install the AIMMS project into.
- ▶ Open the AIMMS project in the location that you have specified.
- ▶ Go to **Settings/Multi Agent.../Community Setup.../Web Service Properties**, double-click on the only line in the **AIMMS Agents** box and change the host in the right part of the dialog with your own machine name (as indicated there).
- ▶ Click on **OK**.
- ▶ Click on **Generate Files**.
- ▶ Click on **OK**.
- ▶ Restart the AIMMS RPC service.

This is already enough to deploy the web service on your machine. Perform the following steps to test the web service:

- ▶ Keep the AIMMS project open and start **CD Playing Time Optimizer.exe**
- ▶ Click on **Load Input Data**, to load some sample data
- ▶ Click on **Optimize Playlist**, to actually call the web service. You should see a response within a few seconds.