



**AGETOR®**

AXT Web Service  
Filter

## Table of contents

1	Introduction.....	3
2	Installation .....	3
2.1	Uses for the AXT Web service filter .....	3
2.2	Security in the AXT web service filter .....	4
2.2.1	Use of HTTP basic authorization.....	4
2.2.2	Securing the message with Secure Socket Layer (SSL).....	4
3	The AXTHHTTPFilter .....	5
3.1	Configuration example .....	5
3.1.1	Description of the configuration.....	6
3.2	Error handling .....	7
4	The AXTSOAPFilter .....	8
4.1	Configuration example .....	9
4.2	Error handling .....	10
5	The AXTEBXMLFilter .....	11
6	Handling ebXML Acknowledgements/ErrorLists .....	12
6.1	Asynchronous Output and Reply .....	13
6.2	Synchronous Output and Reply Messages.....	13
6.3	Internal reply XML-format .....	14
7	Persistent connections for faster responses .....	14
8	Statistical information on HTTP connections.....	14

## 1 Introduction

With the AXT Web service filter you can integrate to services over the Internet or locally. This is done using SOAP or any proprietary formats over HTTP.

This can be posting the content from AXT to a web server or getting and using the response from a web service to do further processing in AXT.

## 2 Installation

This section gives you step-by-step information on how to install AXT Web service filter.

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

Before you begin the installation of the AXT Web service filter, make sure you have the following prerequisites installed:

- Java Development Kit (JDK) version 1.3 or newer. If you are planning to use HTTPS/SSL in the communication make sure that you are using JDK version 1.4 or newer.
- AGETOR Development Kit (ADK) version 2.0.8 or newer.
- AXT version 2.0.3 or newer.
- Webserver with servlet runner e.g. Jakarta-Tomcat or Apache with Apache JServ.

Download the newest version of the AXT Web service filter and copy the package into your "AGETOR\_HOME/install/packages" directory. If you are upgrading an existing installation the existing configuration will be retained.

- 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 "AXT Web service filter 2.0.4" and answer the few questions.

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.

### 2.1 Uses for the AXT Web service filter

The AXT Web service filter is divided into two filters each with their own specific use:

Filter	Description
AXTHTTPFilter	This filter is used when communicating with a web-server using either a HTTP POST or GET method. You are able to POST information to a web page like you would when you are posting the content of a form in HTML. It can also be used when getting the content from a web page or a document located on the web server.
AXTSOAPFilter	This filter is used when communicating with web services using either HTTP

POST or GET as the transport protocol. The filter is primarily for use with SOAP web services, but can be used with other web services using XML and HTTP.

## 2.2 Security in the AXT web service filter

Communicating over the Internet using HTTP as the transport protocol, means transferring information in plain text back and forth between clients and servers. This can lead to a situation in which someone is either seeing or even tampering with the information being sent. If the information is sensitive you need to secure the communication between the client and the server.

The AXT Web Service filter supports two methods for securing your communication that can be used either in combination or individually. Each method has its own different advantages.

### 2.2.1 Use of HTTP basic authorization

The HTTP standard defines a simple way to protect a web resource from unauthorized user access. Through the HTTP basic authorization mechanism, the user has to provide a username and password before being granted access to the requested resource. Otherwise they receive an error with the HTTP status code of 401. The username and password are transferred with every request from the client inside a HTTP-header.

```
POST /web/protected/webservice HTTP/1.0
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Content-Length: 1663
Authorization: Basic c2lsdmVyOnNpbHZlcg==

<SOAP-ENV:Envelope>
...
...
</SOAP-ENV:Envelope>
```

Figure 1: An example of the basic authorization HTTP-header

The example shown above has a HTTP-POST request with a username and password in the HTTP basic authorization header. The username and password is encrypted with base64-encoding.

This provides a basic access mechanism, but does not solve all security problems alone. Obviously, the message inside the <SOAP-ENV:Envelope> is still sent in plain text. Also, a base64-encoded string is not difficult to decode into a readable format – and thus others can get access to the username and password.

The conclusion is that the basic authorization method should not be used alone if you need to fully secure the communication between client and server. Instead you should use it together with SSL. This method is discussed in the next paragraph.

### 2.2.2 Securing the message with Secure Socket Layer (SSL)


The Secure Socket Layer (SSL) is a standard technology for securing the communication between clients and servers with the use of encryption of the information sent across the network. Using

SSL as the underlying transport mechanism for the HTTP-protocol gives you a secure way to send the information back and forth. Using this method you use the “https” instead of “http” in the URL to specify that the SSL protocol be used in the communication.

When a request is sent the complete message is encrypted between client and server. The information cannot be decrypted, so if an outsider intercepts the request he is unable to get hold of the information. At the same time they can't tamper the message or pretend they are someone else. Should somebody along the line try to change the content of the message, it will be caught when the message is decrypted.

To use SSL you need to install a digital certificate on the server and if required on the client. These certificates indicate that the client and server trust each other. You can get a certificate from different sources, for instance <http://www.verisign.com>. Also be aware that encrypting the message using SSL comes with a performance degrade (see chapter below on persistent connections).

Connection to a webservice or webpage page using SSL is done using the HTTPS protocol. An example of an URI is <https://www.verisign.com>.

 This guide does not include how to setup SSL and certificates on the server hosting the webservice or webpage. Instead consult your preferred webserver documentation.

To use a SSL connection you sometimes need to install the certificate from the server on the web server that you are using to host AXT otherwise the AXT Web Service filter can't trust that the server is who it claims to be. Normally when you get a certificate from a trusted organization like Verisign you don't have to install it on the web server. If on the other hand the certificate used is generated you have to install it on the web server hosting AXT. How you install a certificate depends on the web server in question.

In Tomcat 4.0 you can import certificates using the %JAVA\_HOME%\bin\keytool like in the following example:

```
%JAVA_HOME%\bin\keytool -import -keystore .keystore -file client.crt -alias  
trustedserver
```

This installs a certificate located in the file “client.crt” which you have received from the organization that you would like to connect to. On Windows 2000 it will be placed in a file name: C:\Documents and Settings\<<CURRENT\_USER>>\.keystore.

## 3 The AXTHttpFilter

Use the AXTHttpFilter to make requests to a web page from a web server using the HTTP-protocol using either the POST or GET method. Any resource located on the web server can be requested with the AXTHttpFilter, that being a HTML-page, XML-document or an image. At the same time you can use the AXTHttpFilter to POST any content from AXT to a given web server.

### 3.1 Configuration example

The following is an example of a document entry added to AXTConfiguration.xml, which makes a HTTP post request to a web-service with the URI “http://localhost/webservice”. Before the request

is made and after the result is received from the web server, an XSLT-script is invoked to transform the XML sent and the XML received. In the example, the BASIC authorization method is used to make a login. The entry assumes that a username (key-user) and password (key-pwd) was sent as keys to the document entry. These values are used for identification in the HTTPFilter.

```
<axt-conf xmlns="http://www.bording.dk/2001/AXT/config">
  <doc name="webservice entrance">
    <key name="docentry" value="webservice" />

    <filter class="dk.bording.axt.tc.invocation.AXTHTTPFilter">
      <param name="url-target" value="http://localhost/webservice"/>
      <param name="content-type" value="text/xml"/>
      <param name='conn-method' value='POST'/>
      <param name='input-xsl' value='${AGETOR_HOME}/data/axt/input.xsl'/>
      <param name='output-xsl' value='${AGETOR_HOME}/data/axt/output.xsl'/>
      <param name='basic-username' value='{key-user}' />
      <param name='basic-password' value='{key-pwd}' />
    </filter>
  </doc>
</axt-conf>
```

### 3.1.1 Description of the configuration

The filter configured in the above example is the AXTHTTPFilter. This filter takes a number of parameters:

Parameter name	Parameter description
basic-username	The username used to make a basic authorization to the web server. See section 2.2.1 for further information on using this security method.
basic-password	The password used to make a basic authorization to the web server. See section 2.2.1 for further information on using this security method.
conn-method	The HTTP method used for the request. Only "POST" and "GET" methods are valid. Using "GET", the input will not get communicated to the web server and therefore any input-xsl parameter will be ignored. The default is "POST".
content-type	You can set the MIME-type of the content that you are sending to the web server as part of the request. The value of this parameter is depending on the web resource that you are requesting. Normally, using POST as the method, you should set this parameter to "application/x-www-form-urlencoded". If you were making a request with the "GET" method you would normally set it to either "text/html" or "text/xml". The default is "text/xml".
input-xsl	A file reference to an XSLT-script used to transform the XML input from AXT into the content used for the request. In the reference you can insert a reference to the AGETOR_HOME directory with the use of "\${AGETOR_HOME}.
output-xsl	A file reference to an XSLT-script that used to transform the response from the web server before it is outputted to the next filter in AXT. In the reference you can insert a reference to the AGETOR_HOME directory with the use of "\${AGETOR_HOME}.
url-target	The URL for the web page. If you are using "https" in the url see section 2.2.2

	for further information on using SSL in your requests.
connection-pool	The default value is "true". If set to true the filter will try to reuse the connection between calls resulting in better performance. This can cause problems in some cases, which is why you can disable the feature.



If any other parameter is set on the filter in the configuration it will be used as a custom HTTP-header in the request. That way you can set any necessary headers that the web server needs for the request.

### 3.2 Error handling

When you use the AXTHttpFilter to make requests to web pages a couple of different errors can occur, which you can catch and handle in an error router in AXT.

Error-type	Description
dk.bording.axt.AXTConfigurationException	When something is wrong with the configuration of either the parameters declarations in the AXTConfiguration or the XSLT-scripts.
dk.bording.axt.AXTEException	This error occurs if something went wrong while sending the request to the web server. Possible errors could be no response from web server; wrong URL or an internal error in the web server.

The following is an example of the use of an error router in AXT to handle when an error occurs in the AXTHttpFilter.

```
<axt-conf xmlns="http://www.bording.dk/2001/AXT/config">
  <doc name="webservice entrance">
    <key name="url" value="webservice" />

    <filter class="dk.bording.axt.tc.invocation. AXTHttpFilter">
      <param name="url-target" value="http://localhost/webservice"/>
      <param name="content-type" value="text/xml"/>
      <param name='conn-method' value='POST'/>
      <param name='input-xsl' value='${AGETOR_HOME}/data/axt/input.xsl'/>
      <param name='output-xsl' value='${AGETOR_HOME}/data/axt/output.xsl'/>
      <param name='basic-username' value='user'/>
      <param name='basic-password' value='passw'/>
    </filter>

    <router type="error" instanceof="dk.bording.axt.AXTEException">
      <filter class="dk.bording.axt.tc.mail.MailHandler">
        <param name="smtpHost" value="mail.company.com">
        <param name="from" value="AXT"/>
        <param name="to" value="adm@mail.company.com"/>
        <param name="content-type" value="text/xml"/>
        <param name="subject" value="Error sending to web server"/>
      </filter>
    </router>

  </doc>
```

```
</axt-conf>
```

## 4 The AXTSOAPFilter

You can use the AXTSOAPFilter to communicate with SOAP<sup>1</sup> web services using HTTP as the underlying protocol. This gives you a standard way to communicate with web services across different platforms and programming languages.


The AXTSOAPFilter supports SOAP web services giving a response. This is not to be confused with the RPC or DOCUMENT style in the SOAP standard, which mostly has to do with encoding of the message and how it is located.

The AXTSOAPFilter accepts the following parameters:

Parameter name	Parameter description
add-soap-envelope	Can be either "true" or "false". The default is "false". When this parameter is set to true it will insert the content from AXT into a SOAP-Envelope. If "false" you will have to make the SOAP-Envelope your self.
basic-username	The username used to make a basic authorization to the web server. See section 2.2.1 for further information on using this for security.
basic-password	The password used to make a basic authorization to the web server. See section 2.2.1 for further information on using this for security.
check-fault	Can be either "true" or "false". The default is "false". When "true" it will scan the SOAP response for SOAP-faults and throw an error if that is the case. Otherwise it will send the response to the next filter in AXT no matter if it went well or not. You can get a small performance improvement if you set this to "false" but then you will not get any error handling.
conn-method	The HTTP method used for the request. Only "POST" and "GET" methods are valid. If "GET" is used as the method the input will not get communicated to the web server and therefore any input-xsl parameter will be ignored. The default is "POST".
content-type	You can set the MIME-type of the content that you are sending to the web server as part of the request. The value of this parameter is depending on the web resource that you are requesting. Web services build with Microsoft .NET will normally have this set to "text/xml" and the request can fail if set otherwise. The default is "application/soap+xml", which is the new standard.
input-xsl	A file reference to an XSLT-script that can be used to transform the XML input from AXT into the content used for the request. In the reference you can insert a reference to the AGETOR_HOME directory with the use of "\${AGETOR_HOME}.
output-xsl	A file reference to an XSLT-script that can be used to transform the response from the web server before it is outputted to the next filter in AXT. In the reference you can insert a reference to the AGETOR_HOME directory with the use of "\${AGETOR_HOME}.
remove-soap-envelope	Can either be "true" or "false". The default is "false". When you set this parameter to "true" it will remove the SOAP:Envelope and SOAP:Body from the

<sup>1</sup> SOAP is a standard for web services defined by [www.w3c.org](http://www.w3c.org).

	response from the web service.
SOAPAction	Some web services need this to be set to identify the method that you would like to invoke on a web service. Web services build with Microsoft .NET normally demands this set to something specific and Java Axis is normally only demanding that it is set to "".
url-target	The URL for the web page. If you are using "https" in the url see section 2.2.2 for further information on using SSL in your requests.
soap-attachment	Defines if the document should be send as an attachment or included in the soap body. Default is false, meaning there will be no attachment.
soap-namespace	Soap namespace prefix, optional. Default soap namespace prefix is: "soapenv"

 If any other parameter is set on the filter in the configuration it will be used as a custom HTTP-header in the request. That way you can set any necessary headers that the web server needs for the request.

## 4.1 Configuration example

Below an AXT document entry is configured to POST a SOAP request to a remote internet web service. The web service accepts a SOAP request containing an international city name and returns a SOAP response with the local time of that city.

### AXT document entry:

```
<axt-conf xmlns="http://www.bording.dk/2001/AXT/config">
  <doc name="nano">
    <key name="ip" value="*" />
    <key name="content-type" value="*" />
    <key name="url" value="nano" />
    <key name="protocol" value="*" />
    <filter class="dk.bording.axt.tc.invocation.AXTSOAPFilter">
      <param name="content-type" value="text/xml" />
      <param name="SOAPAction"
value="http://www.Nanonull.com/TimeService/getCityTime" />
      <param name="url-target"
value="http://www.nanonull.com/TimeService/TimeService.asmx" />
      <param name='add-soap-envelope' value='false' />
      <param name='remove-soap-envelope' value='false' />
    </filter>
  </doc>
</axt-conf>
```

### SOAP Input to the document entry:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getCityTime xmlns="http://www.Nanonull.com/TimeService/">
      <city>London</city>
    </getCityTime>
  </soap:Body>
```

```
</soap:Envelope>
```

### SOAP response from the filter (document entry output):

```
<?xml version="1.0" encoding="utf-8" ?>
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <getCityTimeResponse xmlns="http://www.Nanonull.com/TimeService/">
      <getCityTimeResult>11:32 AM</getCityTimeResult>
    </getCityTimeResponse>
  </soap:Body>
</soap:Envelope>
```

If the input was not in the SOAP format but merely ordinary XML like:

### XML Input to the document entry:

```
<getCityTime xmlns="http://www.Nanonull.com/TimeService/">
  <city>London</city>
</getCityTime>
```

You could still use the SOAP web service by changing the add/remove envelope parameters to true:

```
<param name='add-soap-envelope' value='true' />
<param name='remove-soap-envelope' value='true' />
```

and the filter would supply the SOAP envelope wrapping before sending the XML input to the web service as well as remove the SOAP result envelope from the response yielding:

### SOAP response from the filter (document entry output):

```
<?xml version="1.0" encoding="UTF-8" ?>
  <getCityTimeResponse xmlns="http://www.Nanonull.com/TimeService/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <getCityTimeResult>11:55 AM</getCityTimeResult>
</getCityTimeResponse>
```

## 4.2 Error handling

The AXTSOAPFilter has the following error types, which can be handled in an error router in AXT.

Error-type	Description
------------	-------------

dk.bording.axt.AXTConfigurationException	When something is wrong with the configuration of either the parameter declarations in the AXTConfiguration or the XSLT-scripts.
dk.bording.axt.AXTEException	This error occurs if something went wrong while sending the request to the web server. Possible errors could be no response from web server; wrong URL or an internal error in the web server.
dk.bording.axt.AXTSOAPException	This error occurs if the web service responded with a SOAP fault. It is an extension of the dk.bording.axt.AXTEException so you can setup an error router handling both.

The following is an example of the use of an error router in AXT that handles an error in the AXTSOAPFilter.

```

<axt-conf xmlns="http://www.bording.dk/2001/AXT/config">
  <doc name="webservice entrance">
    <key name="url" value="webservice" />

    <filter class="dk.bording.axt.tc.invocation.AXTSOAPFilter">
      <param name="url-target" value="http://localhost/webservice"/>
      <param name="content-type" value="text/xml"/>
      <param name='add-soap-envelope' value='true'/>
      <param name='remove-soap-envelope' value='true'/>
    </filter>

    <router type="error" instanceof="dk.bording.axt.AXTEException">
      <filter class="dk.bording.axt.tc.mail.MailHandler">
        <param name="smtpHost" value="mail.company.com">
        <param name="from" value="AXT"/>
        <param name="to" value="adm@mail.company.com"/>
        <param name="content-type" value="text/xml"/>
        <param name="subject" value="Error sending to web server"/>
      </filter>
    </router>

  </doc>
</axt-conf>

```

## 5 The AXTEBXMLFilter

The AXTEBXMLFilter is an extension of the AXTSOAPFilter which includes the creation of a ebMS header.

The AXTEBXMLFilter accepts AXTSOAPFilter parameters with the following additions:

Parameter name	Parameter description
ebms_from	Description of the sender
ebms_to	Description of the receiver
ebms_cpaid	Identifier that governs the exchange of messages between parties.

ebms_conversationid	Conversation id between to two parties.
ebms_service	Service acts on the message.
ebms_action	Action that should be performed.
ebms_messageid	Messageid.
ebms_description	Description of the attached document.
ebms_descriptionLang	The Language code used for the description. Optional, default is en-US
ebms_syncReplyMode	Flag telling if the message should be sent synchronously or asynchronously. Default setting is none which means that the Webservice Filter doesn't receive an acknowledgement on the same connection, and later ebXML acknowledgements must be received in the AXT SOAP inlet.  For synchronous transfer it can be set to "signalsAndResponse" and it will expect a reply on the same connection.
ebms_partyIdType	Optional. Sets the type on the partyId tag.
ebms_serviceType	Optional. Sets the type on the service tag.

Example of an AXT doc entry configuration that sends an ebXML document. The following example assumes that values with in {value} already are defined as keys in AXT:

```
<doc name="sendsoap">
  <key name="function" value="sendsoap"/>
  <filter class="dk.bording.axt.tc.invocation.AXTEBXMLFilter">
    <param name="content-type" value="text/xml"/>
    <param name="SOAPAction" value=""/>
    <param name="url-target"
value="http://localhost:11111/soapinlet/webservice"/>

    <!-- Send AXT output as soap attachment -->
    <param name="soap-attachment" value="true"/>

    <!-- Set up ebXML header message -->
    <param name="ebms_from" value="{ebms_from}"/>
    <param name="ebms_to" value="{ebms_to}"/>
    <param name="ebms_cpaid" value="{ebms_cpaid}"/>
    <param name="ebms_conversationid" value="{ebms_conversationid}"/>
    <param name="ebms_service" value="{ebms_service}"/>
    <param name="ebms_action" value="{ebms_action}"/>
    <param name="ebms_messageid" value="{ebms_messageid}"/>
    <param name="ebms_description" value="{ebms_description}"/>
    <param name="ebms_syncReplyMode" value="signalsAndResponse"/>
  </filter>
</doc>
```

## 6 Handling ebXML Acknowledgements/ErrorLists

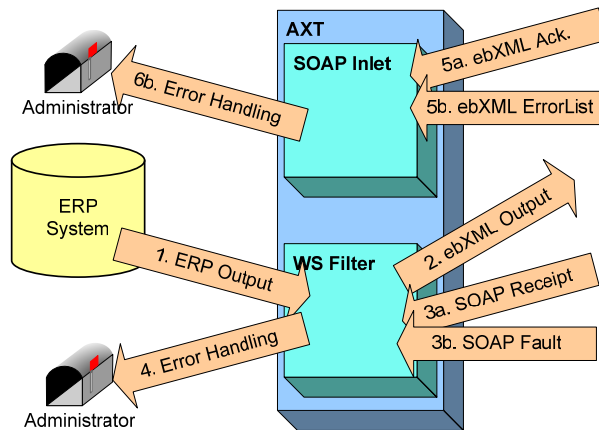
When an ebXML message is sent to a remote receiver it is either accepted by sending an acknowledgement message back, or if errors occurred while processing the message it the remote receiver will process the message and return a ebXML errorList.

Reply messages from the receiver can be sent either synchronously or asynchronously.

## 6.1 Asynchronous Output and Reply

The illustration below shows this case. The Webservice sends the ebXML message to a receiver. If something goes wrong during transmission it will receive a SOAP fault, if the transfer was successful it will receive a soap receipt. The actual ebXML reply message will follow later on a new connection and cannot be handled by the AXT Web Service filter. Instead the acknowledgements and error lists can be received using the AXT SOAP inlet.

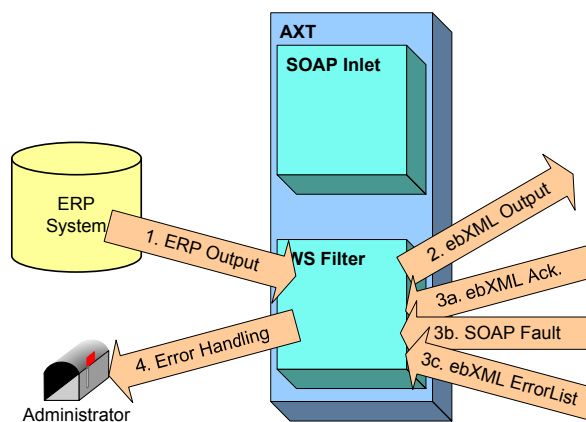
### Async. Output



## 6.2 Synchronous Output and Reply Messages

When sending ebXML messages synchronously using the AXT Web Service filter, the reply from the remote receiver, is received by the AXT Web Service filter on the same open Http connection that was used to send the original ebXML message. This is illustrated in the figure (Sync. Output) below. The reply is then transformed into an internal format to easy processing of information.

### Sync. Output



## 6.3 Internal reply XML-format

The Acknowledgement or Errorlist received from the remote receiver is transformed into an internal format. The design of this format is as follows:

In case of an acknowledgement message the following xml is generated:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<status type="Acknowledgment" referenceId="abc">
  <acknowledgment/>
</status>
```

In case of an errorList message the following xml is generated:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<status type="ErrorList" referenceId="abc">
  <errorlist>
    <error
      errorCode="ValueNotRecognized"
      location="/soap:Envelope/soap:Body/eb:Manifest..."
      severity="Error"
      lang="en-us">Field is absent, field must exist
    </error>
    <error
      errorCode="ErrorCode"
      location="Location"
      severity="Error"
      lang="en-us">Error description
    </error>
  </errorlist>
</status>
```

In case of a receipt message the following xml is generated:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<status type="Receipt" referenceId="abc">
  <receipt/>
</status>
```



The AXTEBXMLStatusFilter can be used to automatically generate these xml in this format. Look in the AXT SOAP Inlet Documentation for more information on how to do this.

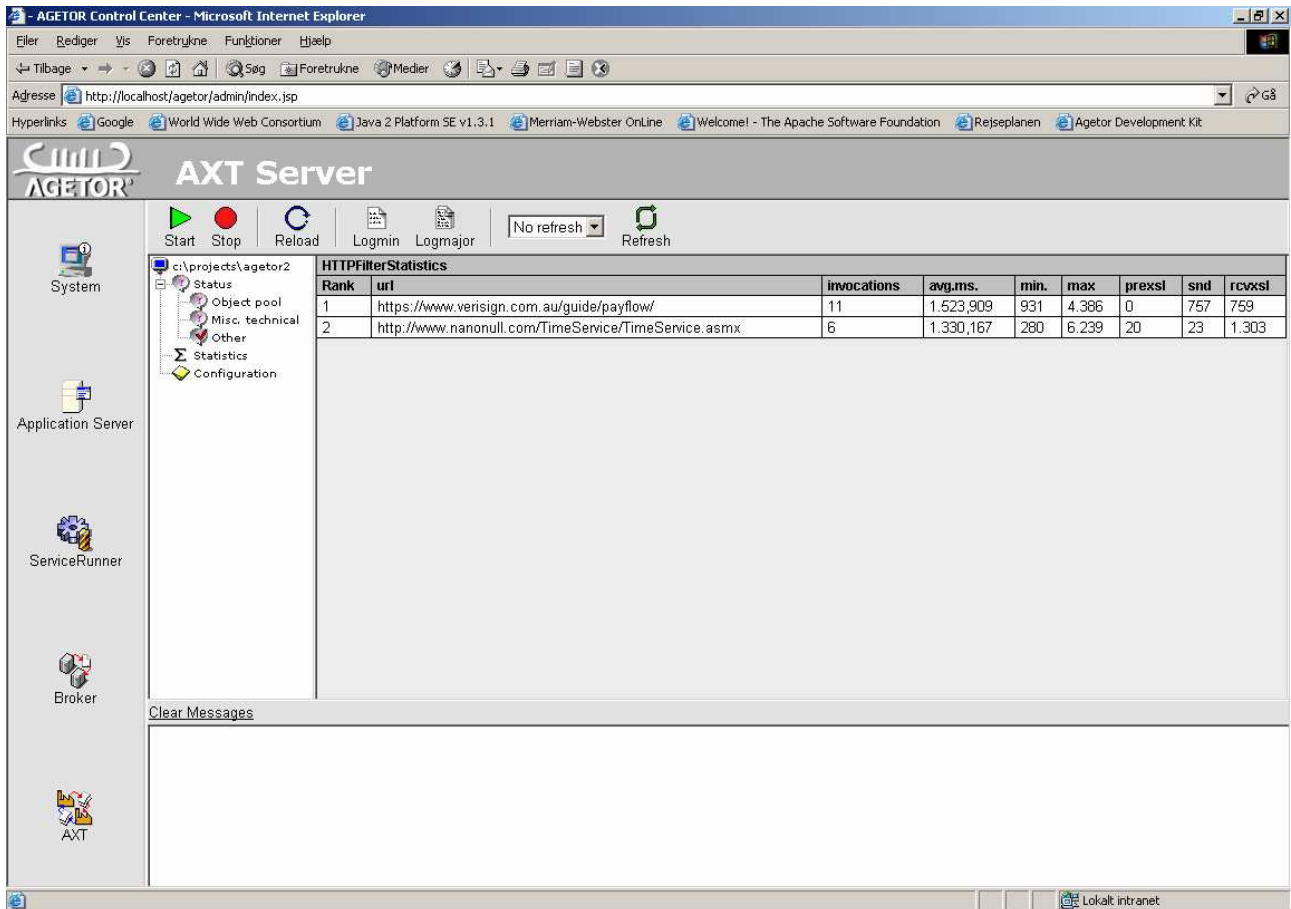
## 7 Persistent connections for faster responses

The AXTHHTTPFilter and the extension AXTSOAPFilter reuses previously established HTTP/HTTPS sessions to remote web servers. This *persistent* use of connections ensures fast data exchange since connection establishment and SSL handshaking is not performed on each invocation. However the degree to which connections may be reused is not a client issue alone. If the remote web server is not configured to keep connections alive the described gains are lost. Please consult with the service provider to have connections kept alive. Also, be aware that to get the full benefit of persistent connections you should use JDK 1.4.

## 8 Statistical information on HTTP connections

With AGETOR 2, AXT supports collection of information across filter instances and dynamic exposure of the generated information in the AGETOR control center. The AXTHHTTPFilter utilizes this new feature to generate statistical information on HTTP requests made. For each unique URL invoked, the filter records the number of invocations, request/response time etc. Once the Web Service package have been installed this information can be browsed in the control center.

However the filter must have been invoked at least once for information to appear. Below an example of the displayed information is shown:



The screenshot shows the AGETOR Control Center interface in Microsoft Internet Explorer. The browser address bar shows `http://localhost/agetor/admin/index.jsp`. The page title is "AGETOR AXT Server". The interface includes a navigation sidebar on the left with icons for System, Application Server, ServiceRunner, Broker, and AXT. The main content area displays "HTTPFilterStatistics" with a table of data. Below the table is a "Clear Messages" button.

Rank	url	invocations	avg.ms.	min.	max	prexsl	snd	rcwsl
1	<a href="https://www.verisign.com.au/guide/payflow/">https://www.verisign.com.au/guide/payflow/</a>	11	1.523,909	931	4.386	0	757	759
2	<a href="http://www.nanonull.com/TimeService/TimeService.asmx">http://www.nanonull.com/TimeService/TimeService.asmx</a>	6	1.330,167	280	6.239	20	23	1.303