



**AGETOR<sup>®</sup>**

AXT Soap Inlet  
User Guide

## Table of contents

Introduction .....	3
1 Installation .....	3
2 Overview of AXT SOAP Inlet.....	4
2.1 A way to send documents to AXT.....	4
2.2 A loosely coupled system integration.....	4
3 How to configure the AXT Soap Inlet .....	5
3.1 The configuration file .....	5
3.2 How the AXT SOAP Inlet will match operations .....	7
3.2.1 Match based on the operation name and namespace in the SOAP request .....	7
3.2.2 Match based on the HTTP header SOAPAction .....	8
3.2.3 Default handling of operations not defined in the configuration .....	10
3.3 Extracting keys to find a document entry in AXT .....	10
3.3.1 New implicit keys in AXT .....	10
3.3.2 Explicit keys in AXT.....	11
3.3.3 Content root used for the request.....	12
3.3.4 Extracting keys, example .....	13
3.3.5 Using namespaces in the XPath expressions.....	14
3.3.6 Extracting prefixed keys in the default handling of operations .....	15
3.3.7 Using a SOAP header to specify a key in AXT .....	17
3.4 Sending the request to AXT.....	20
3.5 Getting the response from AXT .....	21
3.6 Problems with SOAP encoded arrays.....	22
4 Configuration tag reference.....	25
5 SOAP Attachments .....	27
5.1 EbXML Messages.....	27


## Introduction

This user guide introduces the concepts of the AXT Soap Inlet and how to use web services as input to transformations in AXT. A description of the installation process is also provided.

The focus of this guide is explaining how to configure and use the AXT Soap Inlet. Configuration of transformations in AXT is provided in the AXT User Guide. Moreover, it will not introduce the concepts of web services and different technologies used, so please be aware that this guide assumes s knowledge of both SOAP and XSLT to fully understand the examples.

## 1 Installation

This section gives you step-by-step information on how to install AXT Soap Inlet.

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

Before you begin the installation of the AXT Soap Inlet, make sure you have the following prerequisites installed:

- Java Development Kit (JDK) version 1.3 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 Soap Inlet 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 Soap Inlet 2.0.0" and answer the few questions.
- Copy the files located in YOUR\_WEBAPPS/WEB-INF/def into YOUR\_WEBAPPS/WEB-INF folder. Be careful: don't overwrite any files if they are already present! If they are include the content of those files in the existing ones and restart the web-server.
- After restarting your web-server you can post SOAP documents to "http://<YOUR\_WEB\_SERVER/soapinlet/webservice".

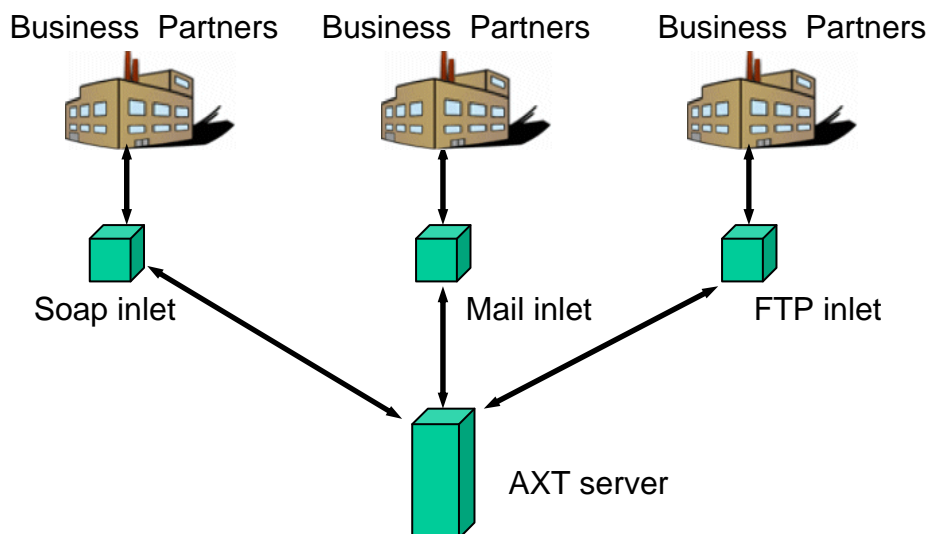
Please note, that new properties might have been added and you should always consult the release.txt for any changes you might need to incorporate.

## 2 Overview of AXT SOAP Inlet

AXT (AGETOR® XML Transformation) is a Business integration management server allowing you to connect different types of documents in an easy and flexible way, thus making your organisation able to respond to the ever-changing demands of the market. By using AXT, your business partners are allowed to send documents and/or data in their desired format, while you are still able to handle all received documents easily using simple transformations within AXT.

### 2.1 A way to send documents to AXT

The AXT Soap Inlet provides the organisation with the opportunity to add the use of web services as yet another format partners can choose to use as the format to send XML-documents to AXT, while still use their existing document transformations along with for instance mail or FTP. The document is handled the exact same way in AXT, regardless of format used to send it.



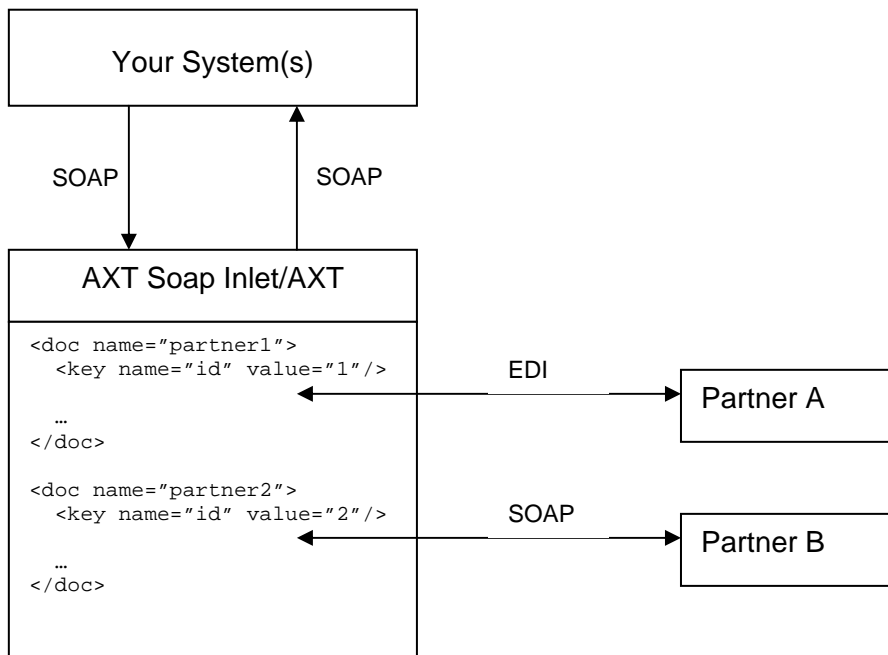
Thus, you can post a SOAP request to the AXT Soap Inlet, extract the configured keys and subsequently handle the extracted XML document in the same way you normally do.

The SOAP Inlet works with standard SOAP implementations, including Microsoft .NET and Java Axis.

### 2.2 A loosely coupled system integration

Besides being used by your partners to send documents to AXT, the AXT SOAP Inlet can be used to provide a loosely coupled method-call between your own systems and AXT, distributing documents to different partners in different formats using different media.

If you have a situation, where you receive information from many different partners using many different formats, you can setup a method-gateway through the AXT Soap Inlet and AXT, routing your method calls to the right partner. In the configuration of the AXT Soap Inlet you setup what should be used as keys for AXT to choose the right document entry in AXT.



The advantage of this approach is that you don't have to change your system, for instance your ERP-system, every time you get a new partner or an existing partner chooses to change their format. You program the interface to AXT once and then you configure it on the fly.

### 3 How to configure the AXT Soap Inlet

The following explains the basic handling and configuration of the AXT Soap Inlet including how-to connect to specific document entries in AXT.

#### 3.1 The configuration file

Usually the configuration file for the AXT Soap Inlet is located in a folder named "\$AGETOR\_HOME/conf/axt/soap". By default, the file is "\$AGETOR\_HOME/conf/axt/soap/SOAPInlet.xml". In the "soapinlet.properties" file you can specify if you want it located somewhere else or have a different name.

In the "soapinlet.properties" file located in the "\$AGETOR\_HOME/conf/properties/axt" folder there are some new properties specific to the AXT Soap Inlet:

Name	Default	Meaning
soapinlet.configuration.path	\${AGETOR_HOME}/conf/axt/soap	The folder where the configuration is located.
soapinlet.configuration.file	SOAPInlet.xml	The name of the configuration file.
soapinlet.configuration.logfile	System.err	Where the logging information should go. The default is to the webservers errorlog, but you can specify a file-name instead.
soapinlet.configuration.loglevel	minimum	Sets AXT Soap Inlet log level. If set to maximum the performance might decrease.
soapinlet.configuration.prefix	AXT	The prefix that the default key extraction will look for when a default operation is invoked.

The configuration file is written in XML and uses the namespace “www.bording.dk/axt/client/soap/conf”. An example configuration file could look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:si="www.bording.dk/axt/client/soap/conf">

  <defaults reject="false">
    <keyformat prefixed="true" keep="false"/>
  </defaults>

  <si:operation name="getItemPrice" namespace="http://company.com/axt">
    <si:keys>
      <si:key name="id" xpath="/:getPrice/:id" keep="false"/>
      <si:key name="itemId" xpath="/:getPrice/:itemId" keep="true"/>
    </si:keys>
  </si:operation>

  <si:operation name="getItemDescription" namespace="http://company.com/axt">
    <si:keys>
      <si:key name="id" xpath="/:getPrice/:id" keep="false"/>
      <si:key name="itemId" xpath="/:getDescription/:itemId" keep="true"/>
    </si:keys>
    <si:axtinput roottag="AxtRequest"/>
    <si:axtoutput xpath="/AxtResponse/*" add-envelope="YES"/>
  </si:operation>

</si:SOAPInlet>
```

**Figure 1: An example of an AXT Soap Inlet configuration**

Inside the root tag <si:SOAPInlet> you describe different methods accepted by AXT Soap Inlet and how it behaves if it receives a SOAP-request not configured. These operations can either be rejected or to resolve which keys to be used based on a prefix of the tag name.

## 3.2 How the AXT SOAP Inlet will match operations

The AXT SOAP Inlet allows a huge variety of different types of SOAP requests – sending them on to AXT<sup>1</sup>. That means that a partner can use whatever SOAP format they like, and with simple XSLT you can transform it to the format that you want to use internally in AXT.

In the AXT SOAP Inlet configuration you setup the available SOAP requests, together with the keys that shall be sent to AXT. The AXT SOAP Inlet uses three different ways to match the SOAP request with AXT. The next sections will introduce these.

### 3.2.1 Match based on the operation name and namespace in the SOAP request

If you send the following SOAP document, using the configuration from the previous section, the AXT SOAP Inlet will look for the first tag called `<soap:Body>` and use it to determine which operation should be invoked. In this example it will look for an operation with the name “getItemPrice” and the namespace `http://company.com/axt`. If it finds an operation it will extract the keys used to find a document entry in AXT, and possibly transform the document before sending it to AXT.

```
<?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>
    <getItemPrice xmlns=" http://company.com/axt">
      <id>10</id>
      <itemId>1000001</itemId>
    </getItemPrice>
  </soap:Body>
</soap:Envelope>
```

**Figure 2: An example SOAP request**

In this example the AXT SOAP Inlet finds the following operation, because it matches the name and namespace from the SOAP document.

---

<sup>1</sup> The AXT SOAP Inlet doesn't define a WSDL (WebService Description Language). A WSDL can be used to describe how a webservice can be invoked and what the response will be.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf">
  . . .
  <si:operation name="getItemPrice" namespace="http://company.com/axt">
    <si:keys>
      <si:key name="id" xpath="/getPrice/id" keep="false"/>
      <si:key name="itemId" xpath="/getPrice/itemId" keep="true"/>
    </si:keys>
  </si:operation>
  . . .
</si:SOAPInlet>
```

**Figure 3: An example of configuration of an operation**

You can have as many operations with the same name as you like, but they must be separated by a different namespace, otherwise the last configured operation is used. At the same time, you can have as many operations belonging to the same namespace, as you want.

The use of namespaces is optional, but it can be a good thing to use them to further describe, what the operation context is or to separate different partners by namespace. Be advised, that if you don't set a namespace to an operation in the configuration, it will only be invoked if the SOAP request does not belong to a namespace. At the same time you cannot use any wildcards to match namespaces or operations.

### 3.2.2 Match based on the HTTP header SOAPAction

When you make a SOAP request using HTTP post, it's possible, based on the SOAP 1.1 specification, to use a HTTP header called "SOAPAction" to indicate the intent of the request. Thus it's possible to specify that an operation should be invoked based on the value of the SOAPAction header instead of the contents of the Body part of the SOAP envelope.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf">
  . . .
  <si:operation soapaction="http://company.com/axt/getItemPrice">
    <si:keys>
      <si:key name="id" xpath="/:getPrice/:id" keep="false"/>
      <si:key name="itemId" xpath="/:getPrice/:itemId" keep="true"/>
    </si:keys>
  </si:operation>
  . . .
</si:SOAPInlet>
```

**Figure 4: An example of using the SOAPAction HTTP header to match an operation**

In the configuration you set the value of the SOAPAction attribute on the operation tag to specify the value of the SOAPAction header. If the AXT SOAP Inlet receives a SOAP request, where the value of the SOAPAction HTTP header matches one of the operations, it will be invoked.

You can mix the use of the SOAPAction attribute with the name and namespace from the previous section, but if the value of the SOAPAction attribute matches the name and namespace is ignored. The SOAPAction attribute should be unique in the configuration, otherwise only the last is called.

Likewise, if the AXT Soap Inlet can't find an operation that matches based on the SOAPAction header, the operation can be used.

```
POST /axis/services/webservice HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length:
SOAPAction: "http://company.com/axt/getItemPrice"

<?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>
    <getItemPrice xmlns=" http://company.com/axt">
      <id>10</id>
      <itemId>1000001</itemId>
    </getItemPrice>
  </soap:Body>
</soap:Envelope>
```

**Figure 5: An SOAP request posted with HTTP-post**

In the above example an example of a SOAP request sent with the HTTP post protocol and with the SOAPAction header set is shown. Upon receiving the SOAP request, the AXT SOAP Inlet will find the operation, extract the keys from the SOAP document and send them to AXT.

### 3.2.3 Default handling of operations not defined in the configuration

You can set a default behavior to handle SOAP requests not defined in the configuration. This gives you the opportunity to handle requests in a very loosely coupled way, without much configuration.

```

<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf">

  <defaults reject="false" contentroot="SOAP">
    <keyformat prefixed="true" keep="false"/>
    <si:keys>
      <si:key name="supplierEAN" value="//supplierId" keep="true"/>
    </si:keys>
    <si:axtoutput add-envelope="NO"/>
  </defaults>

  . . .

</si:SOAPInlet>

```

**Figure 6: The <defaults> operation part of the configuration**

When you set the “reject” attribute on the <defaults> tag to “false”, the AXT SOAP Inlet will allow processing undefined operations in AXT.

An example could be to use all tags starting with “AXT” as keys. The tag “<AXTid>10</AXTid>” would result in the key “id” with the value “10”, if “AXT” is the prefix (in the properties file you set the prefix).

## 3.3 Extracting keys to find a document entry in AXT

When AXT receives a document, it uses keys (name/value pairs) to select the document entry used for the transformation (see the AXT user guide for further information). The AXT Soap Inlet makes it possible to set keys, when a specific operation is invoked. Keys can be a fixed value or based upon some part of the SOAP document giving the possibility to hit different document entries from the same operation, but with different parameters.

### 3.3.1 New implicit keys in AXT

The AXT Soap Inlet will automatically set two new implicit keys that you can use in AXT:

Name	Meaning
_operation	The name of the operation invoked. Is only available when a defined operation is invoked otherwise it will be undefined.
_ns	The namespace of the operation invoked. Is only available when a defined operation is invoked otherwise it will be undefined in AXT.

### 3.3.2 Explicit keys in AXT

Most of the times you want to extract user defined keys, which you can use in AXT to find the right document entry for the request. In the configuration you specify which keys should be extracted or set for each defined operation. The same can be done with the default handling if the AXT Soap Inlet receives an undefined SOAP request.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf">

  <defaults reject="false">
    <keyformat prefixed="true" keep="false"/>
    <si:keys>
      <si:key name="id" xpath="//id" keep="true"/>
      <si:key name="itemId" value="10"/>
    </si:keys>
  </defaults>

  <si:operation name="getItemPrice" namespace="http://company.com/axt">
    <si:keys>
      <si:key name="id" xpath="/getItemPrice/id" keep="false"/>
      <si:key name="itemId" xpath="/getItemPrice/itemId" keep="true"/>
    </si:keys>
  </si:operation>

</si:SOAPInlet>
```

Figure 7: Extracting keys in the configuration

To define the keys to be used, you insert <key> tags inside a <keys> tag. This can be done in both the defaults section and for each operation.

The attributes that you can set on the “key” tag.	
name	The “name” attribute sets the name for the key and will be the same used in AXT to use as a parameter in a call to a filter or to be part of the best match selecting for a document entry.
xpath	The “xpath” attribute is an XPath <sup>2</sup> expression telling the AXT Soap Inlet how to find the value of the key in the SOAP request. That way you can have a variable value based on the values from the SOAP request. The XPath expression should only result in one XML-node being selected. Otherwise the value of the first one will be used.
value	The “value” attribute, which sets the key to a specific value. This value will always be the same; however it will be ignored if you set the “xpath” attribute.
keep	The “keep” attribute can either be “true” or “false” – default is “true”. If the attribute is set to “false”, the selected XML-node will be removed from the XML-document that will be sent to AXT. This attribute only has meaning if you also set the xpath attribute and the expression results in a XML element being selected.

<sup>2</sup> XPath is a XML technology used for selecting and defining parts of an XML document. See <http://www.w3.org/TR/xpath> for further information.

### 3.3.3 Content root used for the request

Before you define the XPath expression for the keys, you should be aware of the content root for the operation for which you define the keys. The “contentroot” is an attribute set on the “operation” tag or on the <defaults> tag and can either be “SOAP” or “BODY”. The latter is the default value.

When you set the “contentroot” attribute to “SOAP”, the <soap:Envelope> tag will be the document root from where you should make the XPath expressions. If it’s not set or you set it to “BODY” it will be the first element of the <soap:Body> tag that will act as document root for your XPath expressions.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:si="www.bording.dk/axt/client/soap/conf">

  <defaults reject="false" contentroot="SOAP">
    . . .
  </defaults>

  . . .
  <si:operation name="getItemPrice" namespace="http://company.com/axt"
    contentroot="BODY">
    . . .
  </si:operation>

  . . .
</si:SOAPInlet>
```

**Figure 8: Use of the contentroot attribute in the configuration**

Consider the above configuration. Now you get a SOAP request invoking the “getItemPrice” operation as in the following example:

```
<?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>
    <getItemPrice xmlns="http://company.com/axt">
      <id>10</id>
      <itemId>1000001</itemId>
    </getItemPrice>
  </soap:Body>
</soap:Envelope>
```

**Figure 9: A SOAP request**

Because the “contentroot” attribute on the <operation> tag is set to “BODY”, the resulting XML-document from which keys are extracted, will be what’s inside the <soap:Body> tag. So when you define the XPath expression, the XML-document will be as the following:

```
<?xml version="1.0" encoding="utf-8"?>
<getItemPrice xmlns=" http://company.com/axt">
  <id>10</id>
  <itemId>1000001</itemId>
</getItemPrice>
```

**Figure 10: With the contentroot set to BODY the content of the body is the XML-document**

On most occasions you are not interested in the SOAP specific part of the request, only the content of the body. So stripping the <soap:Envelope> from the actual XML-document, makes it easier to work with.

### 3.3.4 Extracting keys, example

If we use the XML-document from the last section, together with the configuration from 3.3.2 “Explicit keys in AXT”, we want to extract two keys to use in AXT for the “getItemPrice” operation:

```
<si:key name="id" xpath="/getItemPrice/id" keep="false"/>
<si:key name="itemId" xpath="/getItemPrice/itemId" keep="true"/>
```

The first one will result in the value of the “id” tag located as a parameter to the <getItemPrice> tag. After the key has been made, the element will be removed from the resulting XML-document because the attribute “keep” is set to “false”. Before finding the next key, the document will look like the following:

```
<?xml version="1.0" encoding="utf-8"?>
<getItemPrice xmlns=" http://company.com/axt">
  <itemId>1000001</itemId>
</getItemPrice>
```

**Figure 11: The XML-document after the key is extracted**

This will also be the XML-document sent to AXT, because the “itemId” key has the attribute “keep” set to “true”. The key will be made, but the XML-element is kept in the document.

From the XML-document we will get the keys:

```
id = 10
itemId = 1000001
```

So when will you remove tags? Sometimes you use a parameter only to choose the right document entry, whereas the information in the tag is not used for further processing. For instance, a

supplier id is part of the SOAP document and is used to choose the correct AXT transformation. But once the entry has been found, there is no more use for the id and hence you can remove the tag from the resulting document.

### 3.3.5 Using namespaces in the XPath expressions

When you write XPath expressions you commonly want to use namespaces and their prefixes to fully qualify the XML elements that you wish to select. The problem remains what should be the prefix used to express a certain namespace in an XML-document?

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" >
. . .
</soapenv:Envelope>
```

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" >
. . .
</soap:Envelope>
```

In the above example, the two tags are identical since their namespace declarations both point to the same URI and it's only the use of a prefix-name is different. If you want to write an XPath expression selecting the envelope or one of its potential children, you could write something like this:

```
/soap:Envelope
```

This would result in different results depending on which of the two documents we receive in the SOAP request. The first document would fail giving the client a SOAP fault telling that the used prefix (the prefix is "SOAP" in the XPath expression) doesn't resolve to a namespace, because it has no knowledge of the used prefix. In this document, the prefix for the namespace used to qualify the Envelope tag is called "soapenv". If, on the other hand, we got a SOAP document as the second one, it would work fine and the root tag would be selected.

To get around the problem you can declare the different namespaces used in an XPath expression on the <SOAPInlet> tag in the configuration. Note, that it has to be in that tag otherwise they are ignored and at the same time they all act like global namespace declarations, so therefore they can be used to point to anywhere in the XML-document.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:si="www.bording.dk/axt/client/soap/conf"
  xmlns:soap=" http://schemas.xmlsoap.org/soap/envelope/">
. . .
</si:SOAPInlet>
```

Figure 12: Namespace declarations

In the above example you use the three prefixes declared (“xsi”, “si” and “soap”) in your XPath expression. Now, the mentioned XPath expression will be selected correct in both types of documents.

Note that the namespaces you declare in the configuration are only available to the XPath expressions that you write in the configuration - but not for the XSLT-transformations you specify (see below).

### 3.3.6 Extracting prefixed keys in the default handling of operations

When you handle SOAP requests not defined as an operation in the configuration (besides using the explicit key extracting) you have the opportunity to extract keys based on tags with a certain prefix in their name. This allows the SOAP requests themselves specify the keys used to select a document entry in AXT.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf">

  <defaults reject="false" contentroot="BODY">
    <keyformat prefixed="true" keep="true"/>
  </defaults>

</si:SOAPInlet>
```

**Figure 13: Extracted keys from tags with a prefix**

Consider the configuration above. Inside the <defaults> tag the attribute “prefixed” on the <keyformat> tag is set to “true”. This means that the AXT Soap Inlet will search the XML-document for tags with the prefix and - if any found - use them as keys to AXT. If you set the “prefixed” attribute to “false”, the AXT Soap Inlet will not search the XML-document for prefixed keys, and you either have to rely on the implicit keys or set explicit keys in the configuration.

For example, if you get the following SOAP request and the property “soap-inlet.configuration.prefix” is set to “AXT”:

```
<?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>
    <getPrice xmlns="http://company.com/axt">
      <AXTid>10</AXTid>
      <AXTitemId>1000001</AXTitemId>
    </getPrice>
  </soap:Body>
</soap:Envelope>
```

**Figure 14: A SOAP request with prefixed tags**



This results in the following keys and XML-document:

```
id = 10
itemId = 1000001
```

```
<?xml version="1.0" encoding="utf-8"?>
<getPrice xmlns=" http://company.com/axt ">
  <AXTid>10</AXTid>
  <AXTitemId>1000001</AXTitemId>
</getPrice>
```

**Figure 15: The resulting document after key extraction**

In the configuration, the attribute “contentroot” is set to “BODY” so the SOAP part of the request is stripped before the XML-document is sent to AXT. If you set the “keep” attribute on the “keyformat” tag to “false”, all the XML-elements with a prefix is removed from the resulting XML-document as follows:

```
id = 10
itemId = 1000001
```

```
<?xml version="1.0" encoding="utf-8"?>
<getPrice xmlns=" http://company.com/axt ">
</getPrice>
```

**Figure 16: The resulting XML-document after the keys are extracted and removed**

Please be advised, the small performance hit experienced when using prefixed tags, as the AXT Soap Inlet searches through the entire SOAP document for keys. The faster alternative is using XPath expressions to find configured tags.

### 3.3.7 Using a SOAP header to specify a key in AXT

In this example, we use a SOAP header as a key and transform the SOAP request to a more meaningful XML-document before sending it to AXT.

Consider the following SOAP request with an operation called “getItemPrice”, with the namespace “http://company.com/axt”. Besides the Body tag containing the request, a “supplierId” tag is supplied with the value “10” inside the <soap:Header> tag. This value must be used as a key in AXT.

```
<?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:Header>
    <supplierId xmlns="http://company.com/axt">10
    </supplierId>
  </soap:Header>

  <soap:Body>
    <getItemPrice xmlns="http://company.com/axt">
      <itemId>1000001</itemId>
    </getItemPrice>
  </soap:Body>
</soap:Envelope>
```

**Figure 17: A SOAP request with a header**

In the AXT Soap Inlet configuration, we set up the operation giving the name and namespace to identify the SOAP request. To get the value of the <supplierId> tag located in the SOAP header, we have to set the “contentroot” to “SOAP”, otherwise the SOAP part of the SOAP request will be stripped of the XML-document. Setting “contentroot” to “SOAP” means that you work on an XML-document identical to the received SOAP request.

To extract the <supplierId> tag value, the <keys> tag is added a <key> tag as child. Additional keys can be added, but for this example we only use one. In the XPath expression, we select the <supplierId> tag located as a child under the <soap:Envelope> and the <soap:Header> tag. Because we use namespaces in the XPath expression, we add the two namespace declarations to the root of the “Soap Inlet” tag. If this wasn’t done you could get an incorrect result upon receiving a SOAP request depending on the name used as prefix.

We want to transform the XML-document and set the <supplierId> tag as a childtag under the <getItemPrice> tag before AXT is invoked, so we need to set the “keep” attribute on the <key> tag to “true”.

Now you can use an XSLT-transformation to make the <supplierId> tag a child of the <getItemPrice> tag.

A reason to do this could be that you get a “cleaner” XML when all the SOAP specific parts are stripped from the XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf"
  xmlns:ac="http://company.com/axt"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  . . .
  <si:operation name="getItemPrice" namespace="http://company.com/axt"
    contentroot="SOAP">
    <si:keys>
      <si:key name="supplierId"
        xpath="/soap:Envelope/soap:Header/ac:supplierId"
        keep="true"/>
    </si:keys>
    <si:axtinput xsl-file="{AGETOR_HOME}\input.xsl"/>
    <si:axtoutput xsl-file="{AGETOR_HOME}\output.xsl"/>
  </si:operation>
  . . .
</si:SOAPInlet>
```

**Figure 18: An example configuration of an operation to handle a SOAP header**

Above you can see an example of the configuration to handle the SOAP request and in the following there is an example of an XSLT-script transforming the document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:na="http://company.com/axt">
  <xsl:template match="/">
    <getItemPrice xmlns="http://company.com/axt">
      <id>
        <xsl:value-of select="soap:Envelope/soap:Header/na:supplierId"/>
      </id>
      <xsl:copy-of
        select="soap:Envelope/soap:Body/na:getItemPrice/na:itemId"/>
    </getItemPrice>
  </xsl:template>
</xsl:stylesheet>
```

**Figure 19: An example XSLT-script to transform the SOAP request**

Consider the above XSLT-script. The document that AXT would receive would look like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<getItemPrice xmlns="http://company.com/axt"
xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:na="http://company.com/axt"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <id>10</id>
  <itemId>1000001</itemId>
</getItemPrice>
```

Figure 20: The resulting XML-document

### 3.4 Sending the request to AXT

When the AXT Soap Inlet receives a SOAP request and finds the proper operation or entry, it starts extracting keys defined in the configuration. When all keys have been extracted the remaining XML is sent to AXT together with the keys found for the operation. Just before the XML-document is sent, it's possible to change the XML content using the `<axtinput>` tag inserted as a child under the `<operation>` or the `<defaults>` tags.

Before reading further, remember that the root of the XML-document depends on the value of the "contentroot" attribute. See "3.3.3 Content root used for the request".

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:si="www.bording.dk/axt/client/soap/conf">
  . . .
  <si:operation name="getItemPrice" namespace="http://company.com/axt"
contentroot="BODY">
    <si:keys>
      <si:key name="id" xpath=":/getItemPrice/id" keep="false"/>
      <si:key name="itemId" xpath=":/getItemPrice/itemId" keep="true"/>
    </si:keys>
    <si:axtinput xpath="/*" roottag="AXTRequest"
xsl-file="{AGETOR_HOME}\input.xsl"/>
  </si:operation>
  . . .
</si:SOAPInlet>
```

Figure 21: Use of the axtinput to control the input to AXT

The `<axtinput>` tag has three attributes:

The attributes that you can set on the <code>&lt;axtinput&gt;</code> tag.	
xsl-file	The "xsl-file" attribute can take a filename and gives you the possibility to run an XSLT-transformation before the two below 2 attributes come into effect.

	You can use references to AGETOR_HOME.
xpath	The “xpath” attribute holds an XPath expression, where you can select which XML elements should make up the XML-document sent to AXT. Note that the XPath expression should only result in one selected element. If the XPath expression results in more than one element and you haven’t set the “roottag” attribute (see below), then only the first selected element will be used. If the “roottag” attribute is set, all the selected elements will be added as children to the new root element.
roottag	The “roottag” attribute gives you an easy way to add the content of either the XML-document or the result of the XPath expression to a document with a root element with the given name.

### 3.5 Getting the response from AXT

Receiving the response from AXT you have the opportunity to control how and what is actually sent back to the client. This is only relevant if the transformation in AXT has been successful, otherwise the AXT Soap Inlet will automatically send back an error to the client telling what went wrong.

```

<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf">

  . . .

  <si:operation name="getItemPrice" namespace="http://company.com/axt"
    contentroot="BODY">
    <si:keys>
      <si:key name="id" xpath=":/getPrice/id" keep="false"/>
      <si:key name="itemId" xpath=":/getPrice/itemId" keep="true"/>
    </si:keys>
    <si:axtinput xpath="/*" roottag="AXTRequest"
      xsl-file="{AGETOR_HOME}\input.xsl"/>
    <si:axtoutput xpath="/AXTResponse/*" add-envelope="true"
      xsl-file="{AGETOR_HOME}\output.xsl"/>
  </si:operation>

  . . .

</si:SOAPInlet>

```

**Figure 22: Controlling the response from AXT**

If you want to control the content of the SOAP response, you add an <axtoutput> tag to either the <defaults> or the <operation> tag. The <axtoutput> tag has three attributes.

The attributes that you can set on the “axtoutput” tag.	
xsl-file	This attribute takes a filename and gives you the possibility to run an XSLT-transformation, before the below mentioned attributes come into effect. In the filename you can use references to AGETOR_HOME.

xpath	This attribute holds an XPath expression, where you can select which XML elements should be the XML-document sent back to the client. Note that the XPath expression should only result in one element to be selected. If the XPath expression results in more than one element and the “add-envelope” attribute is set to “false” (see below), only the first element selected will be sent back to the client.
add-envelope	The “add-envelope” attribute can either be set to “true” or “false” telling the AXT Soap Inlet if it should append the content to a SOAP envelope or not. If the attribute is not defined the default behavior is to add the SOAP envelope, i.e. true.

### 3.6 Problems with SOAP encoded arrays

To work with SOAP messages, an important item is to know the different ways the SOAP specification explains how SOAP encoded arrays is written. You should be aware that the different elements of the array can be nested as children located under the method request tag or with a reference to a tag located somewhere else in the document.

In AXIS/Java you can write a method taking an array as parameter like the following:

```
public void updateArray(MyStruct[] input) {
    . . .
}
```

**Figure 23: A Java method returning an array**

Or the same in Visual Basic .NET:

```
<WebMethod()> Public Sub updateArray(ByVal input as MyStruct())
    . . .
End Sub
```

**Figure 24: A Visual Basic .NET method that returns an array**

Both the SOAP request and response would be the same and potentially they both contain the array encoding issue.

The following is an example of a SOAP request in which the items in the array are located as children under the <input> tag. This format is easy to work with because the SOAP part can easily be removed, allowing you to work with the contents as a valid XML-document.

```

<?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>

    <updateArray xmlns="http://company.com/axt">
      <input>
        <MyStruct>
          . . .
        </MyStruct>
        <MyStruct>
          . . .
        </MyStruct>
      </input>
    </updateArray>

  </soap:Body>
</soap:Envelope>

```

**Figure 25: An SOAP request with the array items nested as children**

According to the SOAP specification arrays can also be represented with a reference to elements located somewhere else in the XML-document. In the following example the request is the same as above, but all items in the array are located as children of the <soap:Body> tag. Under the <input> tag there are only references to the elements in the array.

```

<?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>

    <updateArray xmlns="http://company.com/axt">
      <input>
        <item href="#id1"/>
        <item href="#id2"/>
      </input>
    </updateArray>

    <MyStruct id="#id1">
      . . .
    </MyStruct>
    <MyStruct id="#id2">
      . . .
    </MyStruct>

  </soap:Body>
</soap:Envelope>

```

**Figure 26: A SOAP request with the references to the items in the array**

If we remove the SOAP specific part from the SOAP request above, we no longer have a valid XML-document, because there is more than one root-element. This is not well-formed XML.

Therefore, in the case of arrays, you should always set the “contentroot” attribute of the <operation> tag to “SOAP”. If you want to remove the SOAP specific part of the XML-document before sending it to AXT, you have to use the <axtinput> tag to control what the AXT Soap Inlet should do with it, using an XSLT-script to resolve the items, so they are nicely nested as children leaving only one root-element. This can be costly in performance, but will give you a nice XML-document to work with in AXT.

An alternative is to use the “roottag” and “xpath” attribute to select the contents of the “BODY” tag and add them to a new root-element. This removes the SOAP Envelope but leaves the nested XML array to work with in AXT.

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xmlns:si="www.bording.dk/axt/client/soap/conf"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

  . . .

  <si:operation name="updateArray" namespace="http://company.com/axt"
    contentroot="SOAP">
    <si:keys>
      . . .
    </si:keys>
    <si:axtinput xpath="/soap:Envelope/soap:Body/*" roottag="AXTRequest"/>
  </si:operation>
  . . .
</si:SOAPInlet>
```

**Figure 27: An configuration of an operation to handle an array in the request**

```
<?xml version="1.0" encoding="utf-8"?>
<AXTRequest>
  <updateArray xmlns="http://company.com/axt">
    <input>
      <item href="#id1"/>
      <item href="#id2"/>
    </input>
  </updateArray>

  <MyStruct id="#id1">
    . . .
  </MyStruct>
  <MyStruct id="#id2">
    . . .
  </MyStruct>
</AXTRequest>
```

**Figure 28: The resulting XML-document**

The result would look like the above.

## 4 Configuration tag reference


This section is a reference of all the tags and attributes in the configuration file. You can also find the XML schema specification of the configuration in \$AGE-TOR\_HOME/doc/axt/soap/SOAPInlet.xsd.

Tag	Tag description	
	Attribute	Attribute description
SoapInlet	The root element of the configuration. Can have one <defaults> and as many <operation> tags as you like.	
	None	
defaults	Used to handle SOAP requests not defined as an operation in the configuration. It can have "keyformat", "keys", "axtinput" and "axtoutput". Only "keyformat" has to be present - the others are optional.	
	Reject	"true" if undefined operations are to be rejected otherwise "false" to accept the default handling of operations not defined in the configuration.
	contentroot	"SOAP" if you want to work with SOAP request. If "BODY" or not defined it will be the first element of the Body tag inside the SOAP request, that will be the working XML-document.
keyformat	As a child of the "defaults" tag it can specify if the AXT Soap Inlet should search for tags with a certain prefix in the name and use them as keys in AXT. No children.	
	prefixed	"true" if you want to let the AXT Soap Inlet search for tags with a prefix in the name.
	Keep	"true" if the tag that holds the prefix in the name should be left in the XML-document. If you set it to "false" the tags will be removed from the XML-document together with all their children. The default is "true"
operation	Defines a SOAP operation to be matched either via "name", "namespace" or the "soapaction" attribute. It can have zero or more elements of keys, axtinput and axtoutput.	
	Name	The name of the operation. Together with the namespace attribute it can be used to identify the operation when the AXT Soap Inlet gets a SOAP request.
	namespace	The namespace used for the operation. See above.
	soapaction	A string that can be used to identify the operation from the SOAPAction HTTP header in a SOAP request.
	contentroot	"SOAP" if you want to work with the entire SOAP request. If "BODY" or not defined it will be the first element of the Body tag inside the SOAP request that will be the working XML-document.
	keys	Holds a list of children <key> tags that defines the <keys> used for an <operation> or the <defaults> handling.
	None	
key	Holds the information to extract keys for use in AXT. It has no children.	

	Name	The name of the key. It will be the same name that you use in AXT to select a document entry or as a parameter to a filter.
	Value	A specific value that you want this key to have. It is ignored if the xpath attribute is set.
	Xpath	A string containing an XPath expression that can be used to extract the value for the key. This way you can make it dynamic.
	Keep	“true” if the tag that holds the value for the key should be left in the XML-document. If you set it to “false” the tag will be removed from the XML-document together with all the children. The default is “true”
axtinput		Can be a child under both <defaults> and the <operation> tags and gives the opportunity to control the content of the XML-document that is sent to AXT. It has no children.
	Roottag	If defined the string will be used to create a root element. This is the last action before the XML-document is sent to AXT.
	Xpath	A string holding an XPath expression to select parts of the XML-document to send to AXT. If the XPath expression results in more elements being selected, you have to specify a root tag, otherwise only the first element will be used.
	xsl-file	A string containing a filename to an XSLT-transformation that can be invoked to transform the SOAP request to something different before selecting the elements with the xpath.
axtoutput		Can be a child under both <defaults> and <operation> tags and is used to control the SOAP response to the client.
	xsl-file	A string containing a filename to an XSLT-transformation used to transform the output from AXT to something else.
	Xpath	A string that holds an XPath expression used to select which part of the AXT output should be sent back to the client. If the expression selects more than one node you have to set “add-envelope” to “true”, otherwise only the first element will be used.
	add-envelope	“true” if you want the AXT Soap Inlet to add a SOAP envelope surrounding the output. If “false” the output is simply used as response to the client. The default is “true”.

## 5 SOAP Attachments

If the SOAPInlet receives a soap message with attachments included, the attachment included will be sent on to AXT. Keys extraction and xsl transformation is not possible for attachments and only single file attachments are supported.

 Multiple attachments are not supported, due to the way the AXT transformation works.

## 6 EbXML Messages


The SOAP inlet can be used to receive messages using the ebMS transport protocol. In such a case the Soap message Header and Body contains information about the attached document using the ebXML messaging specification ( [http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS\\_v2\\_0.pdf](http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf) ).

### 6.1 The ebXML Message Flow

Messages sent using the ebXML messaging protocol expects an answer to all sent messages. Therefore the SOAP inlet as a receiver of ebXML must send a reply to the sender. This reply is sent as feedback to the sender to acknowledge that the message was received successfully, or as an error-list telling the sender what went wrong. The reply format is also specified by the above mentioned ebXML messaging specification document. Furthermore a reply can be sent either synchronously or asynchronously the difference between these two transfer modes is outlined below:

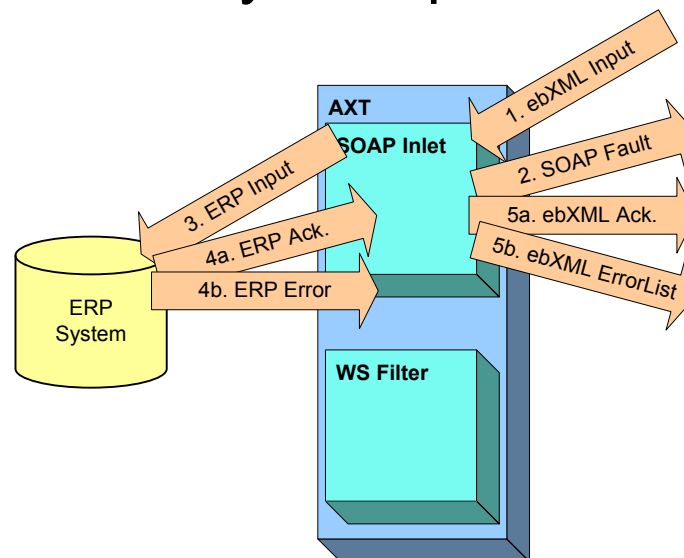
#### 6.1.1 Synchronous ebXML MessageTransfer

When a reply is sent synchronously, the connection originally established by the sender is kept open by the AXT SOAP inlet until the ebXML message has been processed successfully as an AXT stream. A reply is then generated by AXT and the result is the sent as a reply from the AXT SOAP inlet back the sender. Ie. everything happens on the same http connection.

 Whether a message reply is sent asynchronously or not, is decided solely by the received ebXML messages. If the ebXML messageheader contains the **SyncReplyMode** tag it expects a synchronous acknowledgement, otherwise the acknowledgement is sent asynchronously.

Below is a diagram illustrating the flow in synchronous mode:

# Sync. Input



Figur 1: Synchronous ebXML Input

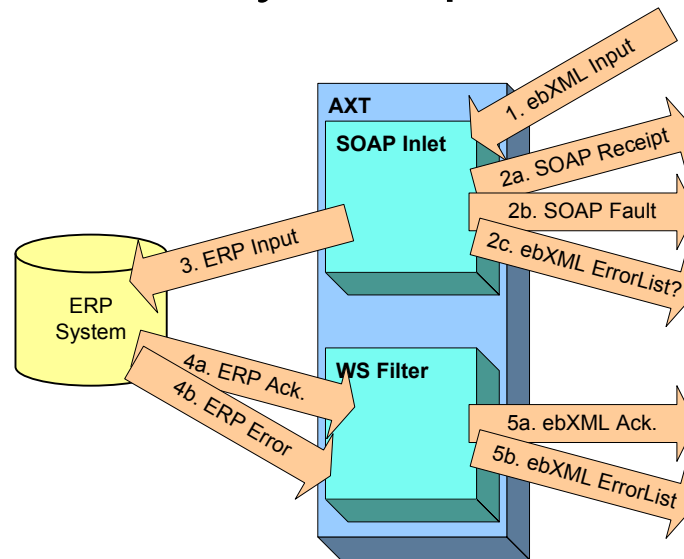
## 6.1.2 Asynchronous ebXML MessageTransfer

Since validation and possible further delivery of an ebXML message can be a time consuming process it is not always convenient to keep an open http connection. Therefore the ebXML messaging protocol allows ebXML messages to be transferred asynchronously. This means that once an ebXML message is received in the AXT SOAP inlet a http response is sent immediately to the sender, indicating that the transfer went well, and the connection is closed. The sender then knows that the message has been sent and is waiting for an acknowledgement to be sent by the receiver of the message on a new connection.

In asynchronous ebXML messages, the acknowledgement/errorlist reply is sent by the AXT Web-service filter.

A diagram illustrating this behaviour is shown below:

# Async. Input



Figur 2: Asynchronous ebXML Input

## 6.2 Internal Status/Result Format


In order to create a proper reply the AXT SOAP Inlet use an internal format to generate the ebXML acknowledgment/errorlist. To ease the process of generating this internal XML format used by the SOAP inlet the AXTEBXMLStatusFilter can be used. This filter can be used to generate a reply which is then sent back to the AXT SOAP inlet.

The AXTEBXMLStatusFilter accepts the following parameters:

Parameter name	Parameter description
status-type	Can be either "acknowledgement" or "error". Defines if the generated XML is an acknowledgement message or an error list. If it is an error the remaining parameters are required, otherwise the can be ignored.
errorcode	Error code for the error
location	XPath expression in the ebXML file where the error first occurred. le: <b>soap:Envelope/soap:Body/eb:Manifest</b>
language	Language code, used for the description. le: <b>en-us</b>
description	Error description

Example of the filter usage:

```
<filter class="dk.bording.axt.tc.invocation.AXTEBXMLStatusFilter">
  <param name="status-type" value="error" />
  <param name="errorcode" value="ValueNotRecognized" />
  <param name="location" value="/soap:Envelope/soap:Body/eb:Manifest/eb:Reference[0]/@xlink:href" />
  <param name="language" value="en-us" />
  <param name="description" value="VV-005:[ISC.0082.9034] Field is absent, field must exist" />
</filter>
```

 For more information on how the internal format is designed and how to generate it yourself look in the AXT Web Service Documentation.

### 6.3 Extracting the AXT keys from the ebXML header

Soap Header and Body information can be extracted to keys in AXT and used to matching transformations or resending the document as an ebXML document with appertaining header information. An example of key extraction from an ebXML document is seen below:

```
<?xml version="1.0" encoding="UTF-8"?>
<si:SOAPInlet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:si="www.bording.dk/axt/client/soap/conf"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:eb="http://www.ebxml.org/namespaces/messageHeader">

  <defaults reject="true" contentroot="SOAP">
    <keyformat prefixed="false" keep="true"/>
  </defaults>

  <si:operation name="Manifest" namespace="http://www.ebxml.org/namespaces/messageHeader"
    contentroot="SOAP">
    <si:keys>
      <!-- Retrieve keys from ebXML message -->
      <si:key name="ebms_from"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:From/eb:PartyId" keep="true"/>
      <si:key name="ebms_to"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:To/eb:PartyId" keep="true"/>
      <si:key name="ebms_cpaid"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:CPAId" keep="true"/>
      <si:key name="ebms_conversationid"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:ConversationId" keep="true"/>
      <si:key name="ebms_service"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:Service" keep="true"/>
      <si:key name="ebms_action"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:Action" keep="true"/>
      <si:key name="ebms_messageid"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:MessageData/eb:MessageId" keep="true"/>
      <si:key name="ebms_timestamp"
xpath="/soapenv:Envelope/soapenv:Header/eb:MessageHeader/eb:MessageData/eb:TimeStamp" keep="true"/>

      <si:key name="ebms_description"
xpath="/soapenv:Envelope/soapenv:Body/eb:MessageHeader/eb:MessageData/eb:TimeStamp" keep="true"/>
    </si:keys>
  </si:operation>
</si:SOAPInlet>
```