



AGETOR®

AGETOR/AXT v. 3.0
what's new?

Content

1	AGETOR/AXT v. 3.0 introduction.....	3
1.1	Changes in brief	3
1.2	Installation.....	4
1.2.1	New installation	4
1.2.2	Updating existing versions.....	4
2	Inlets and DDS-service as AXT-proxy	6
2.1	Inlets and DDS delivery properties.....	7
2.1.1	General (“global”) delivery properties.....	7
2.1.2	Folder/Mailbox level specification of DDS delivery properties.....	7
2.2	Using LocalCalls to Create Resend Points.....	8
2.2.1	LocalCall Description Attribute (Resend Point description).....	9
2.2.2	XML Split Filter.....	9
2.3	Invoking AXT using the DDS	11
2.4	SOAP Attachments	11
2.4.1	Soap Inlet Attachment.....	11
2.4.2	SOAP Debug Option.....	11
2.4.3	AXTEBXMLFilter.....	11
2.5	XML To and From Path Parameters	13
3	Log and Trace improvements	14
3.1	Access to failed DDS documents and AXT transformations errors	14
3.1.1	The role of the DDS-services.....	14
3.1.2	Attached function in L&T.....	14
3.1.3	Error and redelivery tabs in Log & Trace	19
3.2	Miscellaneous improvements in L&T.....	20
3.2.1	Configuring choosable values for configurable search properties.....	20
3.2.2	Configuring shortcut links to list sizes for log and trace event lists	20
3.2.3	Checkpoint icons differ depending on attributes on the document.....	21
3.2.4	Configurable trace level operator and trace level for trace searches	21
3.2.5	Improved Trace Status Icon	21
4	User management and login to the system.....	22
4.1	Introduction to the rights system.....	22
4.2	Initial login to the system.....	22
4.3	The AGETOR application bar	22
4.4	User management GUI.....	23
4.4.1	Users and profiles.....	24
4.4.2	Rights and Properties	24
4.4.3	Rights	25
4.4.4	Properties	26
4.5	Configuration database	27
4.6	Management of AGETOR supported databases	27
4.6.1	Resetting users and profiles using system tool	28
4.6.2	Other database management commands supported with ADKDBTool	28
4.7	Database Management GUI.....	28
4.7.1	Database Selection	28

4.7.2	Database Details Management.....	29
4.8	Installation Management GUI.....	30
4.9	Technical interaction and API.....	31
4.9.1	The LoginFilter	31
4.9.2	PageAccessFilter	31
4.9.3	URL-based login and external property overriding	31
5	Configurable persistence of AXT transformation data	32
5.1.1	Data Storage Modes	32
5.1.2	Global configuration	32
5.1.3	Document Entry Configuration.....	32
5.1.4	Filter Level Configuration	33
5.1.5	Performance Issues	34
6	Configuring periodic cleanup of failed/succeeded dataset	35
7	JOBS GUI	36
8	Architectural changes.....	36
8.1	ADK DB-service (vs. log service)	36
8.2	Configuring the distributed system (multiple installations).....	37
8.2.1	How to set up logging server routing.....	37
8.2.2	How to set up configuration server routing	38
8.2.3	Mapping L&T application information to AXT/DDS-servers	38
8.2.4	Example setup with two machines.....	40
9	Miscellaneous.....	41
9.1	New DDS parameters and properties.....	41
9.1.1	Retry pattern.....	41
9.1.2	Controlling checkpoint generation on delivery success and failure from AXT	42
	Checkpoint generation on delivery attempts	42
10	API news (for developers)	43
10.1	Generating AXTRetryableProcessingExcepations from custom AXT filters	43
10.2	Communicating precise error context from custom AXT filters	43
10.3	Adding function support in L&T	44
10.4	Accessing transformer instance and document ID in an XSL context	44

1 AGETOR/AXT v. 3.0 introduction

1.1 Changes in brief

Version 3.0 of AGETOR/ADK provides a number of improvements both with respect to internal architecture, Log & Trace functionality, AXT error presentation and JOB mangement. The basic ADK system now comes with a user management system that allows simple access rights to be individual or grouped. Highlights in short:


- Configurable persistence of AXT filter input
- Detailed error presentation of AXT errors in the L&T GUI
- Presentation and redelivery of DDS-files from within the L&T GUI
- AXT FTP/File/Mail Inlets use DDS to send document to AXT. This ensures retries and a redelivery option on delivery failure
- AXT JOBS GUI allow for browser based search and mainpulation of jobs handled by DDS server

- User and profile management supporting individual page access restriction
- Misc. improvements to the L&T GUI
- Docdeliver can now issue ftp protocol commands in connection with a transfer and support passive mode FTP

1.2 Installation

1.2.1 New installation

Installing from scratch impose no special problems. You should simply install the ADK executable and all related 3.0 packages you need (AXT basic, DDS, FTPInlet, MailInlet, SOAPInlet, Print and EDI packages).

 It is highly recommended to do a new install for AGETOR 3.0.0 rather than upgrade since this simplifies the installation procedure and ensure a sound environment with explicit default for new setting and potentially improved settings for old configuration items..

1.2.2 Updating existing versions

AGETOR/AXT 3.0 can be installed on top of an ADK 2.1.1 version, but not on earlier versions. If you have an ADK 2.1.0 you should first upgrade that to 2.1.1.

Installing on top of 2.1.1 requires some attention since you might have modified configuration and property files. You also have to consider how the upgrade of you Log & Trace database should be conducted. We will go through the options and procedures in this section.

1.2.2.1 Backing up the existing database

You may wish to backup your existing data before executing the update operation (see below) or for other reasons – e.g. periodically. You simply secure the content of the AGETOR_HOME/data/db/derby directory. We recommend to zip the complete directory into a backup file. To subsequent the content you will simply have to 1) delete the directory (important) and 2) unzip the old content again. Replacing the current database will of course loose its data.

1.2.2.2 Installing on top of 2.1.1

- 1) Shut down running AGETOR/AXT components
- 2) Unzip the 3.0.0 zip file on top of your existing installation (your AGETOR_HOME directory). This does not overwrite any of your configurations but simply ensures that the install-tool is updated (classes) and that the ADK/AXT 3.0.0 files are placed in your packages directory (AGETOR_HOME/install/packages)
- 3) Delete agetor and agetor.bat in the directory (AGETOR_HOME/bin).
- 4) Run the setup command from the directory (AGETOR_HOME/bin/install) Ie. setup <AGETOR_HOME> <JAVA_HOME> <project-name-with-no-spaces>
- 5) Invoke the install tool and install the packages

You will be prompted to determine if existing configuration files should be left untouched or overwritten with the files from the packages. ***It is crucial that you think carefully about what modification you have made to the existing files so you do not loose changes in this step.***

The modification to the install tool ensures that you are only prompted for files that are different. It also present the install action (leave or override local file) with a suggested default value. E.g.

some core system configuration files are not likely to be modified by users and may safely be overwritten.

If you decline to overwrite files (because of local changes) you should make sure that the modifications from the package are merged into your files. This is a manual procedure but it is only strictly necessary for the following files (i.e. you may ignore the changes in other files since the system will assume decent defaults anyway).

- docdelivery_services.cfg
 - this file has an important new property. Since you are not likely to have changed the file, we suggest you simply replace the local copy.
- broker.cfg
 - This file has new definitions for a db-service. You must make sure to take the new definitions in. It is likely that you have changed this file if you have added your own services, environments etc.
- Hibernate.cfg.xml, logdb.hbm.xml
 - Must be installed
- Log4j.properties
 - Should be merged since some insignificant errors and warnings are filtered out in this version
- agetor2 webapps path have been renamed to agetor3, to accommodate for this change, copy the content of your agetor2 webapps path into agetor3 (optionally you can afterwards delete the agetor2 webapps path)

Once through the above steps you must update the database if you did not choose to completely install a new fresh one during installation. If you did so, you can disregard the database updating steps below.

1.2.2.3 Updating an existing database

During the install you are prompted if you wish to completely replace the current Log & Trace database with a new empty database. This is the recommended solution since this guarantees a complete and valid data structure.

However, if you have a need for retaining your log and trace data you need to perform an update. This update will construct new tables and columns (added in this version). Since this must be done on live data it may both be time consuming (depending on database size) and come with a risk of data corruption.

The update procedure requires you to manually execute an update command script while the database server is running. The steps are:

- 1) Backup existing data
- 2) Make sure the servicrunner was started (db-services running) but do **NOT** start the broker. This will ensure the database is available but that logging is not transmitted during the update
- 3) To ensure the db-server is running before you do the update you can invoke the `s cmd` tool and execute the `show` command. The db-server should be `running`.

```
C:\projects\adk2.1.1>scmd
ServiceRunner Client ver. 1.0

Type help for overview.

> show
```

Service	Status	Started/stopped
ftpinlet	stopped	Fri Jan 19 13:45:52
log-server	running	Fri Jan 19 13:46:04
db-server	running	Fri Jan 19 13:45:54

4) Execute the update script (run the script `update-lt-db` in an AGETOR prompt)

The output of step 4 should be error free and similar to the output showed below:

```
C:\projects\adk2.1.1>update-lt-db
Updating the Log and Trace database..
create table docFct (parent_id bigint not null, functionId bigint not null unique)
create table eventFct (parent_id bigint not null, functionId bigint not null unique)
create table fct (id bigint not null, name varchar(255), urlbase varchar(255), primary key (id))
create table fctArg (id bigint not null, value varchar(255), name varchar(255) not null, primary key (id, name))
alter table traceevent add column eventNo integer
alter table docFct add constraint FKB0EEE2FFA1803CBC foreign key (parent_id) references document
alter table docFct add constraint FKB0EEE2FF414E199C foreign key (functionId) references fct
alter table eventFct add constraint FK1093625DEB588FB6 foreign key (parent_id) references traceevent
alter table eventFct add constraint FK1093625D414E199C foreign key (functionId) references fct
alter table fctArg add constraint FKB3B7237F450C3BC4 foreign key (id) references fct
C:\projects\adk2.1.1>
```

The update script will upgrade/create Log & Trace database tables without deleting existing data. One final step must be completed to ensure database consistency. The following command lines must be executed to fill the new columns (eventno and available) with valid values.

```
>j dk.bording.inside.db.ADKDBTool --cmd:execSQL --db:logdb --statement:"update traceevent set eventno=1"
```

```
>j dk.bording.inside.db.ADKDBTool --cmd:execSQL --db:logdb --statement:" update fct set available=1"
```

Once the operation has been executed you should restart the system (service runner and tomcat). You should now be able to login with the administrator user (admin/admin).

2 Inlets and DDS-service as AXT-proxy

The File, FTP, Mail and SOAP inlets are now sending retrieved documents for processing in AXT using the local DDS-service. This has a number of advantages – primarily with respect to delivery insurance and homogeneity.

For inlets this means that the inlet does not have to fail immediately if the AXT server is not up and running, instead it can locally store the data on disk together with a job file, and schedule a later redelivery to the AXT server. This is achieved by passing the document to the local DDS-service instead of trying to post directly to the AXT server.

This approach has the following advantages:

- Greater robustness to network fallouts
- Automatic delivery retries
- Configurable retry interval and number of retries, both globally and on folder level
- Easy manual handling through the AXT Redelivery GUI

The main disadvantages are:

- Bigger data transfer overhead (a factor of 2)
- Synchronous replies to sender can no longer be guaranteed

The inlets comes with the use of DDS as proxy as default. It must be explicitly turned of in the inlets properties file.

2.1 Inlets and DDS delivery properties

When using the DDS proxy mode for delivery to AXT (default) in the FTP, Mail and SOAP-inlets, incoming data are sent to AXT through the DDS server. The DDS server support a rich number of delivery options that may be useful to use. E.g. one may wish to specify the number of retries that should be done if delivery fails initially etc. For a complete overview of DDS delivery options please refer to the *DDS User Guide*.

2.1.1 General (“global”) delivery properties

Delivery defaults pertaining to *all* configured folders/mailboxes or the SOAP documents of the SOAP-inlet may be given in the `delivery-defaults` tag which is placed immediately under the `inlet` root tags. E.g. for the `FTPInlet` we may specify:

```
<delivery-defaults>
  <param name="enrichFromTrace" value="false"/>
  <param name="retryPattern" value="30;60;600"/>
  <param name="description" value="Delivery to local AXT from test directory"/>
  <param name="synchronized" value="false"/>
  <param name="checkpointTextOnError" value="The file {[docprop:ref]} sent by [docprop:sender] with did=[job:_did] for
  <param name="checkpointTextOnSuccess" value="The file {[docprop:ref]} sent by [docprop:sender] with did=[job:_did] f
</delivery-defaults>
```


Such options are used when delivering the job to AXT through the DDS *unless* overridden on folder/mailbox level.

2.1.2 Folder/Mailbox level specification of DDS delivery properties

Under each folder/file/mailbox-folder tag it is possible to specify some DDS delivery properties used when dispatching the job to AXT. If any such properties have been defined in the `delivery-defaults` tag the folder definition will override the global definition from there.

The definition below defines a filefolder from which files are sent to AXT with the key *functions* set to *orderhdl*. In addition a number of DDS properties are given to ensure three redeliveries, synchronized delivery and checkpoint generation when delivered.

```
<filefolder paused="true" name="\${AGETOR_HOME}/data/test/input1" filename="*.t1" listfetchfrequency="5" age="5"
  retrypattern="none"
>
  <delivery>
    <param name="retryPattern" value="10;20;30"/>
    <param name="description" value="Delivery to local AXT from test directory"/>
    <param name="synchronized" value="true"/>
    <param name="checkpointTextOnError" value="FI The file with did=[job:did] and ref=[docprop:ref] could not b
    <param name="checkpointTextOnSuccess" value="FI The file with did=[job:did] was delivered on media [job:med
  </delivery>
  <posturl>
    <param name="function" value="orderhdl"/>
  </posturl>
</filefolder>
```

 Using this method for DDS delivery actually allows for delivery to other media than AXT. E.g the files located in a file-folder could be directly mailed to a mailbox, or stored on a FTP-server. This simply require the specification of an alternative media and whatever media specific properties are needed (similar to delivery from AXT with DocDeliverOutlet filter).

The following two examples illustrates such alternative deliveries:

```
<!--
  FTP folder delivery
-->
<filefolder name="\${AGETOR_HOME}/data/test/input2" filename="*.ftp" listfetchfrequency="5" age="5"
  retrypattern="none"
>
  <delivery>
    <param name="media" value="ftp"/>
    <param name="description" value="file to ftp dir delivery"/>
    <param name="synchronized" value="false"/>
    <param name="serie" value="file-ftp"/>
    <param name="ftpserver" value="acme.main.server"/>
    <param name="mask" value="fileftp%s.txt"/>
    <param name="folder" value="\${AGETOR_HOME}/data/test/output2"/>
  </delivery>
</filefolder>

<!--
  Mail delivery
-->
<filefolder name="\${test.case1.inputdir}" filename="*.mail" listfetchfrequency="5" age="5"
  retrypattern="none"
>
  <delivery>
    <param name="media" value="mail"/>
    <param name="smtpserver" value="\${test.mailserver}"/>
    <param name="from" value="fi@bording.dk"/>
    <param name="to" value="\${test.mailuseraddress}"/>
    <param name="subject" value="Direct input from FI!"/>
    <param name="synchronized" value="false"/>
    <param name="serie" value="file-mail"/>
    <param name="attachment" value="filemail%s.txt"/>
  </delivery>
</filefolder>
```

Figure 1. Delivering from file folders to other media (e.g. another file folder or a mail box).

2.2 Using LocalCalls to Create Resend Points

Using the DDS-service as a proxy is now also available inside AXT from using the LocalCall filter.

This filter normally invokes a new document entry in the same AXT directly. The motivation for using the DDS as a proxy is the possibility to use the extra functionality resident in the DDS server, such as automatic retries and later manual redelivery. This functionality is made available to the Localcall filter with the added attribute *resendable*.

Furthermore the possibility to choose which AXT to invoke has been made available with the attribute *axturl*. If *axturl* is not specified the local AXT is used by default. Thus it is transparent to the user if the local call process is handled directly by AXT or first passed to the DDS server, it is solely controlled by the *resendable* parameter.

2.2.1 LocalCall Description Attribute (Resend Point description)

When invoking a LocalCall (or an XMLSplitFilter) it is possible to set a description attribute on the filter if it is resendable. Setting the description attribute will cause the filter to pass this description to the DDS and if the job fails to be delivered a resend point will be generated with this description. This possibility allows the configuration manager to configure a meaningful text description to the end user, for jobs that might require to be resend in case of failure. An example of how to use the description attribute in connection with local call is shown below:

```
<doc name="testLocalCall">
  <key name="function" value="testlocalcall"/>
  <filter class="dk.bording.axt.flow.LocalCall"
    description="Start of Order processing number {order_no}" resendable="true">
    <param name="function" value="orderproc"/>
  </filter>
</doc>
```

 Note that it is possible to use AXT variables inside the description attribute.

2.2.2 XML Split Filter

The XML Split Filter is now included as a part of the standard AXT filters. It provides the functionality to split a single XML document into several separate new documents to be processed by AXT. These new documents are processed by AXT using the LocalCall described earlier to invoke AXT. This means that all attributes that applies to the LocalCall filter will also apply to the SplitFilter.

Parameter	Type	Description	Default
Select	Optional	An xpath expression used to select elements in the XML document. All selected elements are sent to AXT as separate XML documents.	/
exception_noselect	Optional	if set to "true", cause the filter to throw an exception if the xpath expression doesn't select any elements. If it is set to false (the default value) it doesn't throw an exception if the expression does not select any element.	False
partname	Optional	The part name which is used as the document reference for the newly generated documents together with a per part increasing number.	Part
stopprocessingonfailure	Optional	Determines if the splitfilter should stop incase processing of one of the extracted	True

		documents fails. If set to false the splitfilter will continue processing the next document extracted regardless of failure.	
Createnewtraces	Optional	Determines if the documents should figure as new independent traces in the Log & Trace GUI.	False
__axtkey__<keyname>	Optional	One or more AXT keys may be given using this syntax where the prefix __axtkey__ is removed before the <keyname> part is sent with the document for processing in the AXT server	

Below is an example of a splitfilter invocation is shown:

```

<doc name="splitfilter">
  <key name="function" value="splitfilter"/>



  <filter class="dk.bording.axt.flow.XMLSplitFilter" resendable="true" >
    <param name="select" value="/order/orderline"/>
    <param name="partname" value="part number"/>
    <param name="createnewtraces" value="false"/>
    <param name="stopprocessingonfailure" value="false"/>
    <param name="__axtkey__function" value="testSplitData"/>
  </filter>

</doc>

<doc name="testSplitData" precedence="100">
  <key name="function" value="testSplitData"/>
  ...
  ...
</doc>

```

In this example an xml file is expected as input with a structure of elements that match the xpath expression `/order/orderline`. All elements within this part of the tree are found, split and sent for processing by the testSplitData document entry. If the processing of the split documents fails, the splitfilter will continue with the next in line since the `stopprocessingonfailure` value is set to false. All events will be kept within a single trace and if one of the documents fail, it will automatically be retried a number of times by the DDS service since the `resendable` attribute is set to true and the SplitFilter inherits these options from LocalCall (See LocalCall section above for details). If automatic redelivery fails, it is possible to perform a manual redelivery of the document job and data from the Log and Trace GUI.

-  The output of the AXT SplitFilter is the same as the input, unless something else is configured at filter level.
-  Keys for AXT are unlike the Local Call filter prefixed with `__axtkey__` when specified as parameters to avoid possible name clashing. This is the same convention which is used by the Docdeliver AXT media.


2.3 Invoking AXT using the DDS

The DocdeliverOutlet Filter and the AXT media in Docdeliver have been extended so that the filter now optionally returns the output of the media instead of the result. This is controlled by the output parameter. The DocdeliverOutletFilter is also expanded with the childrenstoredata parameter for the AXT media which controls the storedata attribute for the children of the AXT document entry which is hit by the filter. See LocalCall in Section 5 for more details on storedata functionality.

The DDS now also has the parameter savefaileddata, which is used for controlling what happens with documents that fail. The default behaviour is that the optional savefaileddata parameter is set to true, meaning that if a job fails, the jobfile and the datafile is moved to the docdeliver failed folder, and a redeliver function is available in the L&T GUI. If set to false, the job and data file is simply delete if docdeliver fails to deliver the document, no resend function is generated and in general the data is lost. This is useful if knowledge exists that facilitates a possible redeliver from a different state in the processing pipeline.


2.4 SOAP Attachments


In this release the former AXT Webservice package has been included in the AXT Soap inlet package and thus, the AXT Webservice package does no longer exist.

 The AXT Webservice package should no longer be installed since it is part of the SOAPInlet package

2.4.1 Soap Inlet Attachment

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. Multiple attachments are supported, they will each be sent to AXT using the same keyset for each.

 When there are multiple attachments in a SOAP message these are sequentially posted to AXT. When all attachments have been processed by AXT a reply is sent back to the sender.

 If there are multiple attachments in the soap message and the connection is abruptly interrupted there is a risk that some of the attached documents have been processed by AXT while others remain unprocessed. This is reported back to the sender, which is then responsible for resending the “missing” documents.

2.4.2 SOAP Debug Option

The SOAP inlet now has the possibility to store the http input request it receives by defining setting a debug option in the soap inlet properties file. The received input is printed to stdout, this enables the user to debug the information received and change the soap inlet configuration accordingly.

Furthermore the AXTEBXMLFilter now also contains the possibility to set a debug parameter, which when set will print the http request it sends to stdout.

2.4.3 AXTEBXMLFilter

The AXTEBXMLFilter is an extension of the AXTSOAPFilter which includes the creation of a ebMS header. This filter now supports building a XML like syntax for the ebMS header to enable the possibility of creating multiple receivers. An example of how this is done is illustrated below:

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_syncReply	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.
Ebxml.*	Xml pseudo xpath parameter values. Used for defining multiple type partyids. See XML To and From Path Parameters section.

An AXT doc entry example configuration that sends an ebXML document is shown below. The following example assumes that values with in {value} branches 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_syncReply" value="signalsAndResponse"/>
  </filter>
</doc>

```

2.5 XML To and From Path Parameters

Instead of defining To and From Parameters using a single standard axt parameter, it is possible to use pseudo xpath expression to defined multiple partyids and attributes on the tags to and from. This is illustrated in the example below:

```
<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/soapinlet/webservice"/>

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

    <!--Special XML path setup of multiple sender/receivers -->
    <param name="ebxml.To.PartyId(0)" value="SE8888888888"/>
    <param name="ebxml.To.PartyId(0)[@type]" value="organizationid"/>
    <param name="ebxml.To.PartyId(1)" value="8888888888888888"/>
    <param name="ebxml.To.PartyId(1)[@type]" value="EAN"/>
    <param name="ebxml.To.PartyId(2)" value="IMPORT"/>
    <param name="ebxml.To.PartyId(2)[@type]" value="AXID"/>
    <param name="ebxml.From.PartyId(0)" value="SE8888888888888888"/>
    <param name="ebxml.From.PartyId(0)[@type]" value="organizationid"/>
    <param name="ebxml.From.PartyId(1)" value="8888888888888888"/>
    <param name="ebxml.From.PartyId(1)[@type]" value="EAN"/>
    <param name="ebxml.From.PartyId(2)" value="IMPORT"/>
    <param name="ebxml.From.PartyId(2)[@type]" value="AXID"/>

    <!-- Set up ebXML header message -->
    <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_syncReply" value="signalsAndResponse"/>
  </filter>
</doc>
```

This will generate an EBXML message header with the to and from field on the following form:

```
<eb:From>
  <eb:PartyId eb:type=" organizationid">SE8888888888</eb:PartyId>
  <eb:PartyId eb:type="EAN">8888888888888888</eb:PartyId>
  <eb:PartyId eb:type="AXTID">IMPORT</eb:PartyId>
</eb:From>
<eb:To>
  <eb:PartyId eb:type="organizationid">SE888888888888</eb:PartyId>
  <eb:PartyId eb:type="EAN">8888888888888888</eb:PartyId>
  <eb:PartyId eb:type="AXTID">IMPORT</eb:PartyId>
</eb:To>
```

3 Log and Trace improvements

3.1 Access to failed DDS documents and AXT transformations errors

With version 3.0, the error reporting from the DDS service and the AXT server have been enhanced to include possible linking information back to the services in the case of errors. This means that when a DDS document delivery fails, DDS will generate a trace event to which a *redelivery* function is attached. The attached function is basically an URL with certain parameters that allow the system to open a *redelivery page* directly from the L&T event listings which fetches information from the DDS server that had the error. I.e. regardless of the location of the DDS server¹ the failed document and its delivery properties may be inspected, corrected and submitted for redelivery.


Similarly, the AXT server will attach function information allowing inspection of the errors that occurred during transformation. If properly configured (see section on *configurable persistence of AXT errors*), the input documents of various steps during the transformation will be available for download from the *AXT error view* which is a web-page that may be opened from L&T.

3.1.1 The role of the DDS-services

Since multiple project installations may exist and cooperate in an AGETOR/AXT environment (and these may be shielded by firewalls) it is not straightforward to access data from different distributed services. The DDS-service plays a special role with AGETOR 3.0 since it exposes documents failed during delivery or data from failed AXT transformations of the installation. This is done through an IDL interface. In this way the data-sets are available throughout the distributed system (i.e. any machine through firewall-tunnelling) if routing is set up.

When the redelivery or AXT error view functions are accessed from the L&T GUI, the web-pages actually uses IDL-requests to fetch data sets from the correct project (DDS/AXT servers) using information registered in the L&T system (name of machine, application type, document id etc.).

3.1.2 Attached function in L&T

In L&T the events that have functions attached will show with a function icon (). This icon indicates that the event has further information or interaction available by clicking the icon.

The standard system provides access to the failed DDS-files and to (configurable) datasets from AXT transformations.

The screenshot below shows a redelivery event attached to a DDS-event (created when DDS gave up document delivery and moved the file in question to the error repository).

¹ Note that in a distributed environment with multiple DDS services reporting to the same L&T-server it may be necessary to configure the internal routing between machines in order to be able to access any DDS-service from any machine. This configuration is described in the section on AXT-server routing.

Trace List	Trace Events		
/ T-14			
✓	07-12-2006 22:43:42:452	Send <u>org-input</u> To JVM-2/dds/server(2)	T-14/bbo-laptop/adk2.1/JVM-1/axt/core(1)/docentry(...)
✓	07-12-2006 22:43:42:502	Received <u>org-input</u> From JVM-1/axt/core(1)/docentry("ddaxt1" S)/main(3)/DocDeliverOutlet(5)	T-14/bbo-laptop/adk2.1/JVM-2/dds/server(2)
✗	07-12-2006 22:43:52:401	Sending <u>org-input</u> : Sending file:AXT error codes:8/200 nested exception(dk.bording.axt.client.StreamPosterException): The IP address of the host localhost: in http://localhost:serilet/dk.bording.axt.server.ContainerServlet could not be determined	T-14/bbo-laptop/adk2.1/JVM-2/dds/core(1)/media(1)/...
✗	07-12-2006 22:43:52:541	Transforming <u>org-input</u> : Transform error.	T-14/bbo-laptop/adk2.1/JVM-1/axt/core(1)/docentry(...)
✗	07-12-2006 22:43:52:551	Transforming <u>org-input</u> : Transform error	T-14/bbo-laptop/adk2.1/JVM-1/axt/core(1)/docentry(...)
✓	07-12-2006 22:43:52:621	Sent <u>-Exception trace-</u> To invoking client(browser?)	T-14/bbo-laptop/adk2.1/JVM-1/axt/core(1)/docentry(...)
✗	07-12-2006 22:43:52:621	Transforming <u>org-input</u> : Transform error	T-14/bbo-laptop/adk2.1/JVM-1/axt/core(1)
✓	07-12-2006 22:43:59:508	Sent <u>org-input</u> To error-repository (C:\projects\adk2.1\data\docdelivery/fail)	T-14/bbo-laptop/adk2.1/JVM-2/dds/core(1)
✗	07-12-2006 22:43:59:509	There are errors in the trace.	

Clicking the function will result in a pop-up with a clickable link. Choosing this link opens the *DDS redelivery page*.

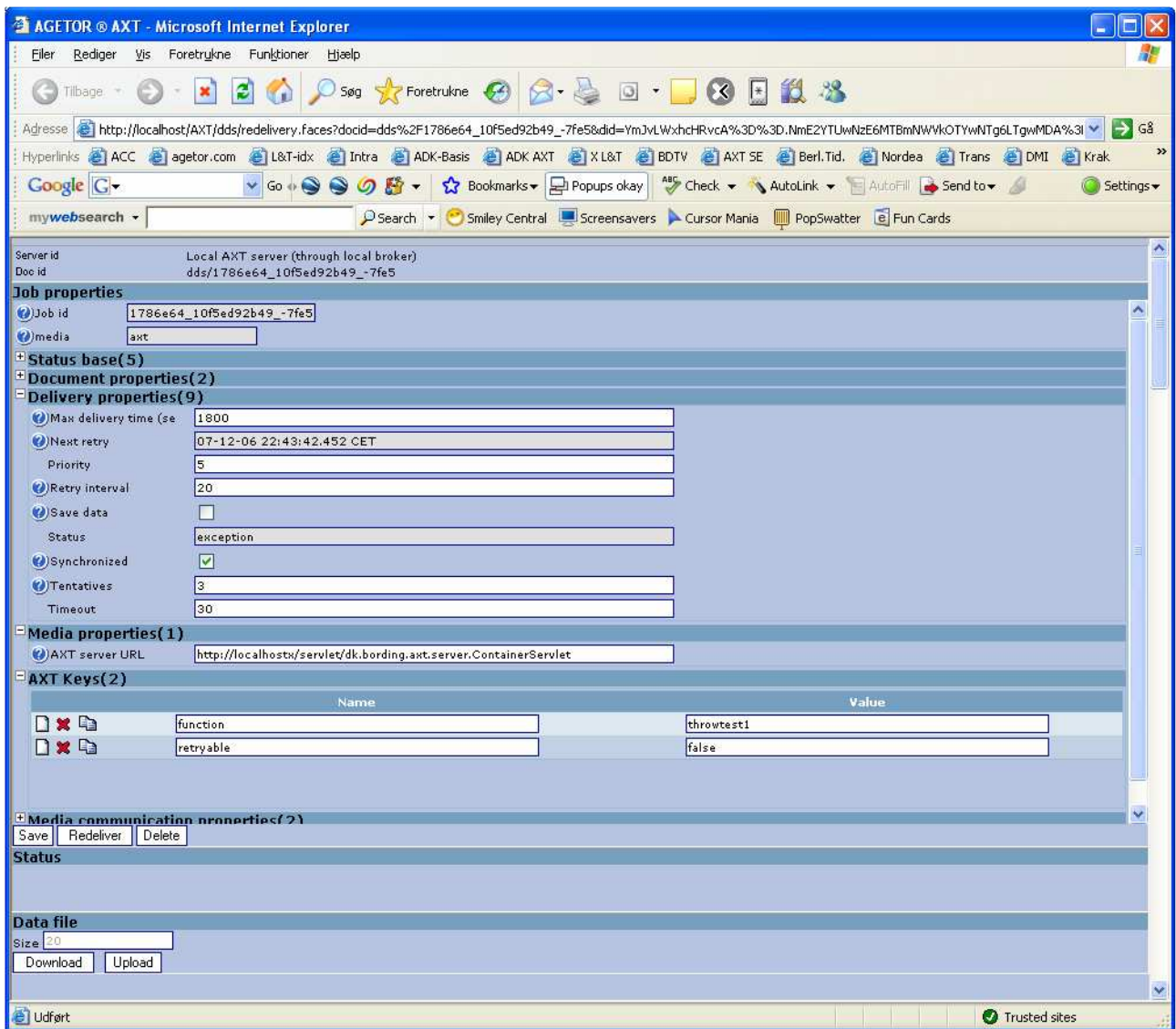
7-12-2006 22:43:52:541	✖	Transforming <u>org-input</u> : Transform error
7-12-2006 22:43:52:551	✖	Transforming <u>org-input</u> : Transform error
7-12-2006 22:43:52:621	➡	Sent <u>-Exception trace-</u> To invoking client
7-12-2006 22:43:52:621	✖	Transforming <u>org-input</u> : Transform error
7-12-2006 22:43:59:508	➡	Sent <u>org-input</u> To error-repository (C:\projects\adk2.1\data\docdelivery/fail)
7-12-2006 22:43:59:509	ⓘ	There are errors in the trace.

[Redelivery Local AXT server
\(through local broker\)](#)

3.1.2.1 The DDS redelivery page

This page shows information on the DDS delivery including all properties from the job file. Some values are editable. E.g. in case it was impossible to send the file to an FTP-server because the FTP-server name was wrong, it is now possible to modify the name and submit the job for redelivery by choosing the redeliver function.

If the delivery was caused by errors in the data content of the file being sent, the file may be downloaded to the local computer where it may be edited and subsequently uploaded. Again a redelivery may be attempted.











The redelivery page structures delivery properties in logical sections that may be expanded/collapsed for better focus and overview.










For AXT jobs a set of keys are present and these may be edited as well.


3.1.2.2 The AXT error view

AXT errors may attach functions as show below.

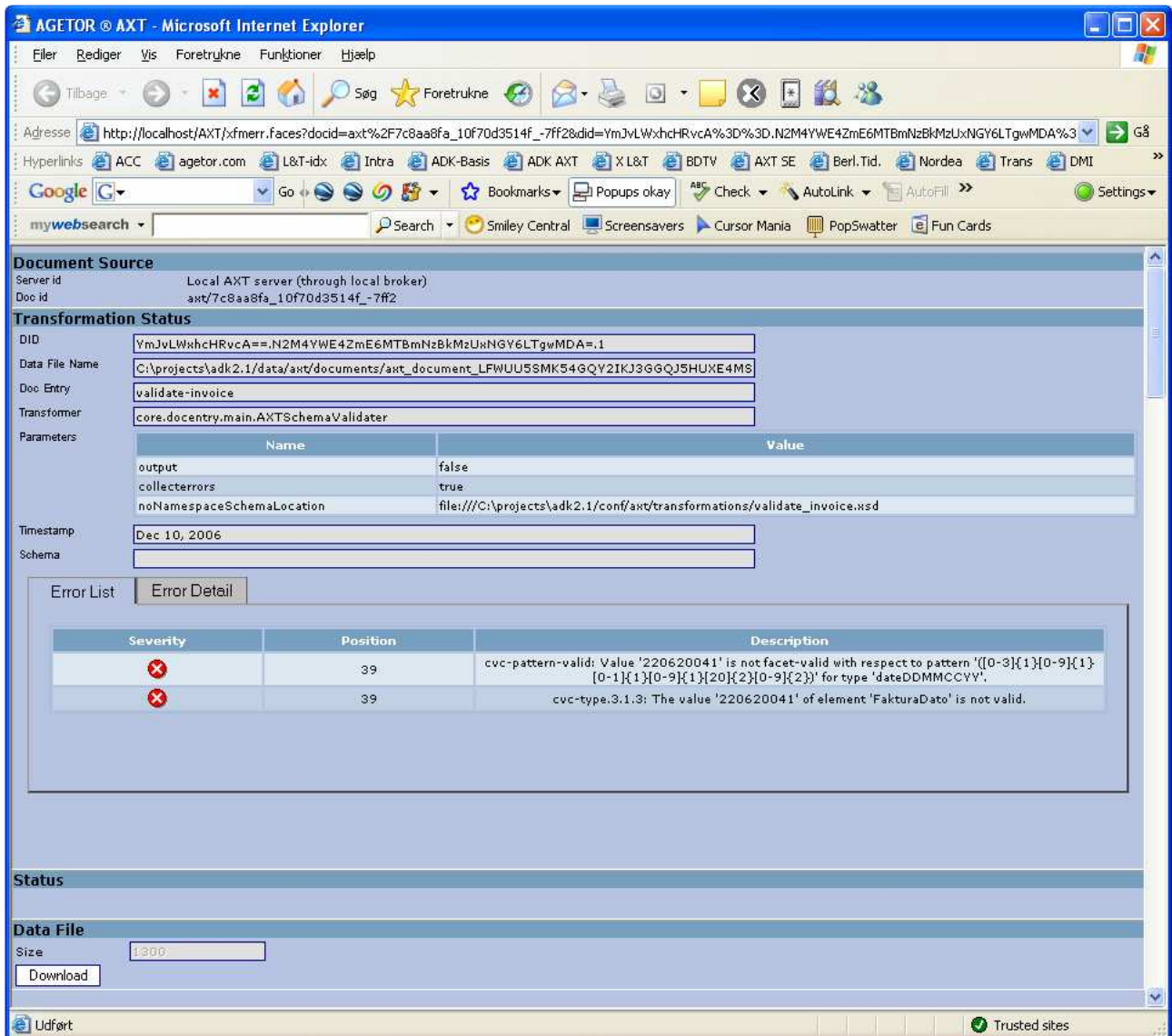
Time	Event	Description	
006 10:37:57:642		Received <u>d1</u> From invoking client(browser?)	T-21/bbo-laptop/adk2.1/JVM
006 10:37:57:672		Sent <u>d1</u> To  <u>main(1)/AXTSchemaValidator(1)</u>	T-21/bbo-laptop/adk2.1/JVM
006 10:37:58:112	 	Received <u>d1</u> From  <u>docentry("validate-invoice" 1)</u>	T-21/bbo-laptop/adk2.1/JVM
006 10:37:58:533		Transforming <u>d1</u> : Transform error	T-21/bbo-laptop/adk2.1/JVM
006 10:37:59:224		Sent <u>-Exception trace-</u> To invoking client(browser?)	T-21/bbo-laptop/adk2.1/JVM

Again the function pop-up allows the user to choose the function – in this case the AXT error view.

Time	Event	Description
11-12-2006 10:37:57:642		Received <u>d1</u> From invoking client(bro
11-12-2006 10:37:57:672		Sent <u>d1</u> To  main(1)/AXTSchema
11-12-2006 10:37:58:112	 	Received <u>d1</u> From  docentry("vali
11-12-2006 10:37:5		<u>Error View Local AXT server</u> <u>(through local broker)</u> nsform error
11-12-2006 10:37:59:224		Sent <u>-Exception trace-</u> To invoking c
11-12-2006 10:37:59:454		Transforming <u>d1</u> : Transform error
11-12-2006 10:37:59:455		<i>There are errors in the trace.</i>

 Note that the function text shown in the popup boxes is configurable and may/may not include information about the source of the error.

The AXT error view is shown below.



Document Source
 Server id: Local AXT server (through local broker)
 Doc id: axt/7c8aa8fa_10f70d3514f_7ff2

Transformation Status

Name	Value
output	false
collecterrors	true
noNamespaceSchemaLocation	file:///C:/projects/adk2.1/conf/axt/transformations/validate_invoice.xsd

Timestamp: Dec 10, 2006

Error List

Severity	Position	Description
✘	39	cvc-pattern-valid: Value '220620041' is not facet-valid with respect to pattern '((0-3){1}{0-9}{1}{0-1}{1}{0-9}{1}{20}{2}{0-9}{2})' for type 'dateDDMMCCYY'.
✘	39	cvc-type.3.1.3: The value '220620041' of element 'FakturaDato' is not valid.

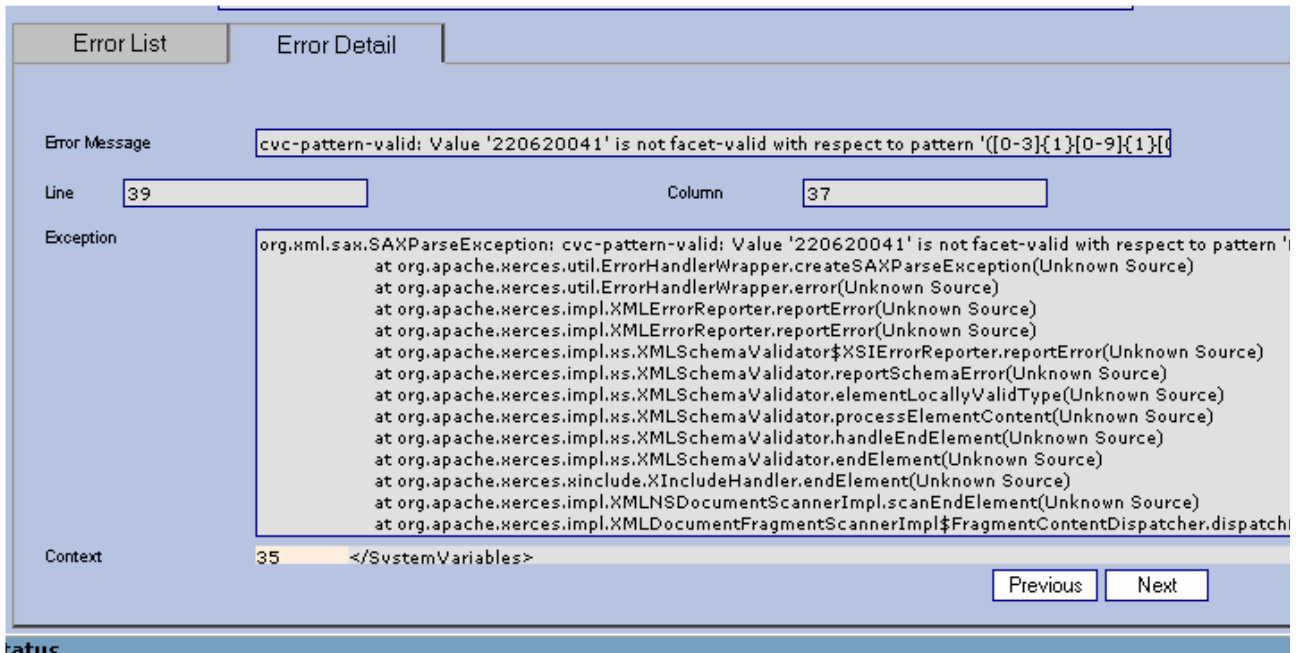
Status

Data File
 Size: 1.300
 Download

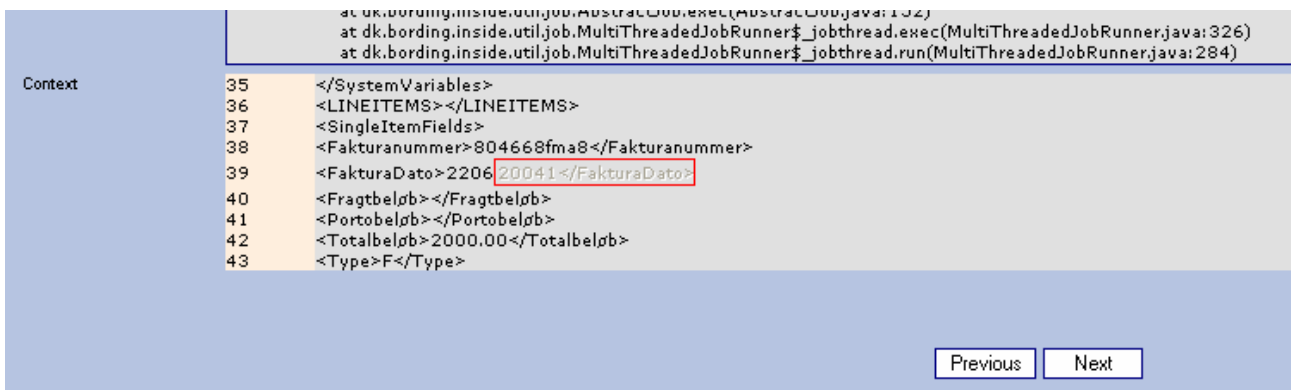
This view has miscellaneous information about the AXT error that occurred. This includes technical information about the document entry, the keys to the filter that failed, the filters transformer name etc.

Just as interesting the error view holds the details on all errors collected by the filter. E.g. the stack traces, the position in the input file *and a snippet from the input pin-pointing the error*. However, the quality and presence of this snippet is completely dependent on the ability of the filter to report the error exactly. Some filters do this well and others not at all. I.e. the XSLProcess and AXTSchemaValidator filters have good support since input processing is what they do. A filter that sets some keys may not report any source position.

The error details below shows how an error message and a stack trace is produced for the schema-validation of an XML-file.



Scrolling further down the screen the context – i.e. the position reported in the error is show with a highlight of the problematic area.



3.1.3 Error and redelivery tabs in Log & Trace

For traces that have attached error or redelivery functions, the trace detail view now provide tabs that gives fast access to viewing error information and redelivering failed jobs. The information and functions available are a subset of the available information in the more detailed views that can be accessed directly from the event list.

See the illustration below for an example.

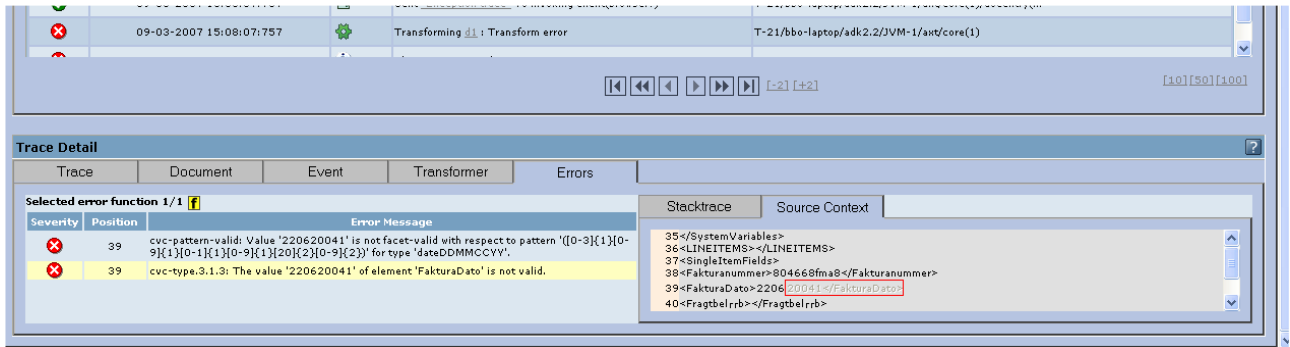


Figure 2. AXT errors visible in error tab

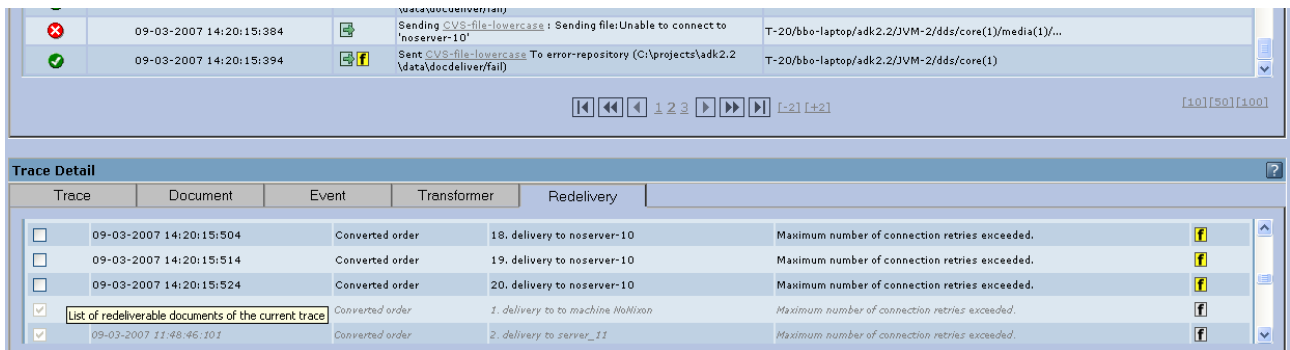


Figure 3. List of redeliverable documents of trace

3.2 Miscellaneous improvements in L&T

3.2.1 Configuring choosable values for configurable search properties

Values for the configurable search properties in the trace search may be predefined to a set of values in the `logtrace.properties` file. This is done by setting a number of value properties corresponding to the configured property:

```
agator.trace.gui.search.property.value.<m>.<n>=
```








where `<m>` is the number of the property and `<n>` is the nth value (ranging from 1 and up). The values are shown in a dropdown list. The order (`<n>`) determines the order of the list. If it must be possible to *not* choose a value, one of the values may be set to nothing (i.e. no right-hand side assignment).

3.2.2 Configuring shortcut links to list sizes for log and trace event lists

For the log and trace lists new shortcut links to specific sizes of the lists may now be set. New installations come with the following values.

```
agator.logging.lists.resultsizelinks=10;50;100
agator.trace.lists.resultsizelinks=10;50;100
```



This results in the following links:

Event	Description	Transformer
	Received <code>d1</code> From invoking client(browser?)	T-21/bbo-laptop/adk2.1/JVM-1/ant/core(1)/docentry(...)
	Sent <code>d1</code> To <code>main(1)/AXTSchemaValidator(1)</code>	T-21/bbo-laptop/adk2.1/JVM-1/ant/core(1)/docentry(...)
	Received <code>d1</code> From <code>docentry("validate-invoice" 1)</code>	T-21/bbo-laptop/adk2.1/JVM-1/ant/core(1)/docentry(...)
	Transforming <code>d1</code> : Transform error	T-21/bbo-laptop/adk2.1/JVM-1/ant/core(1)/docentry(...)
	Sent <code>-Exception trace-</code> To invoking client(browser?)	T-21/bbo-laptop/adk2.1/JVM-1/ant/core(1)/docentry(...)
	Transforming <code>d1</code> : Transform error	T-21/bbo-laptop/adk2.1/JVM-1/ant/core(1)
	There are errors in the trace.	

[10] [50] [100]

Expanding the size of the lists may be convenient to get a complete view when many events are present in e.g. traces.

3.2.3 Checkpoint icons differ depending on attributes on the document

The checkpoint icons shown now depend on whether interesting attributes are present for a checkpoint. If not, an “empty” document icon is shown () , otherwise the usual document icon is shown () . It is possible to substitute these icons with others.

A document is defined as “interesting” when it has attached properties *beyond* the properties defined by the system property `agetor.trace.gui.results.checkpoint.doc.cond`.

The default setting of the property is

```
agetor.trace.gui.results.checkpoint.doc.cond=ref
```

This means that only checkpoints with documents that have more properties than the ref property will be shown with the usual document icon. I.e. if certain attributes should not attract special attention, these may be listed in this property using semi-colon separation:

```
agetor.trace.gui.results.checkpoint.doc.cond=ref;myboringproperty1;myboringproperty2;...
```



3.2.4 Configurable trace level operator and trace level for trace searches

The properties in the example below allow adjustment of the default levels and operators used for trace and log searches:

```
agetor.trace.gui.search.defaultloglevelop=>=
agetor.trace.gui.search.defaultloglevel=Error
agetor.logging.gui.search.defaultlevelop=>=
agetor.logging.gui.search.defaultlevel=Warning
```




3.2.5 Improved Trace Status Icon

Previously a trace had two primary states namely:

-  Indicates that errors have occurred in the trace.
-  Indicates that there are no errors in the trace.

With the introduction of redelivery, it is possible for a trace to contain errors in the events and yet still be completed successfully. This happens if a trace has errors which are caused by a delivery which is unable to complete and thus generates one or more manual redelivery points. A trace in

such a state will be in error and thus the status icon shown will still be red. If manual handling of this trace (ie. resending the pending documents) is successful, so there are no longer any undelivered documents in the trace, it changes state. The new state reflects that there has been error in the trace earlier, but they have been taken care of manually. The icon for this state is green with a tiny red mark in the bottom right corner. Below is a summary of the new states:

-  Indicates that the trace has errors.
-  Indicates that there was no error in the trace.
-  Indicates that there were errors in the trace, but they have been fixed by manual redelivery.

4 User management and login to the system

4.1 Introduction to the rights system

With ADK 3.0 the AGETOR/AXT system supports the use of differentiated user rights and properties. This means that a user must log in to the system using a name and password.

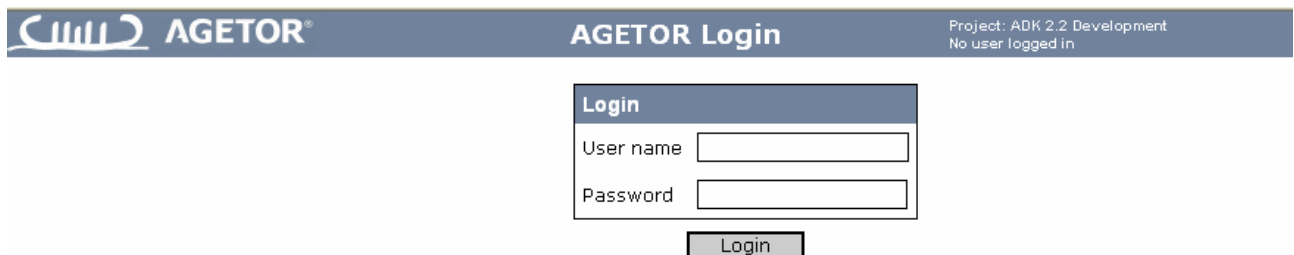
Individual users may rely on common *profiles* but still override certain properties and rights.

A profile defines a set of rights and a set of properties. At present the rights are limited to page-access (i.e. restricting the access to certain pages in the system) and the property system is not really used yet except for a few properties limiting the traces visible for users to traces with a certain document or trace properties (see the Log & Trace system documents for details on the group concept).

4.2 Initial login to the system

When the AGETOR system is initially invoked from a browser the user is required to provide credentials in the form of a login name and password. These are validated against information in the configuration database.

By default the user/password is `admin/admin` (which may be modified after login if required).



Upon successful verification of the credentials the page requested is shown.

4.3 The AGETOR application bar

Once logged in, an information and shortcut bar is displayed at the top of (most) AGETOR and AXT pages. The bar displays information about the current project and the user logged in.

This bar is show below.

It may be seen from the above bar that we are in the application “AGETOR Control Center”, that the project name is “ADK 3.0 Development” and that the user logged in is “Administrator”.

The bar provides convenient shortcuts to certain pages of the system from the *menu* link. Holding the mouse over the menu item will give a pop-up menu with links to the Administration, Log & Trace etc.

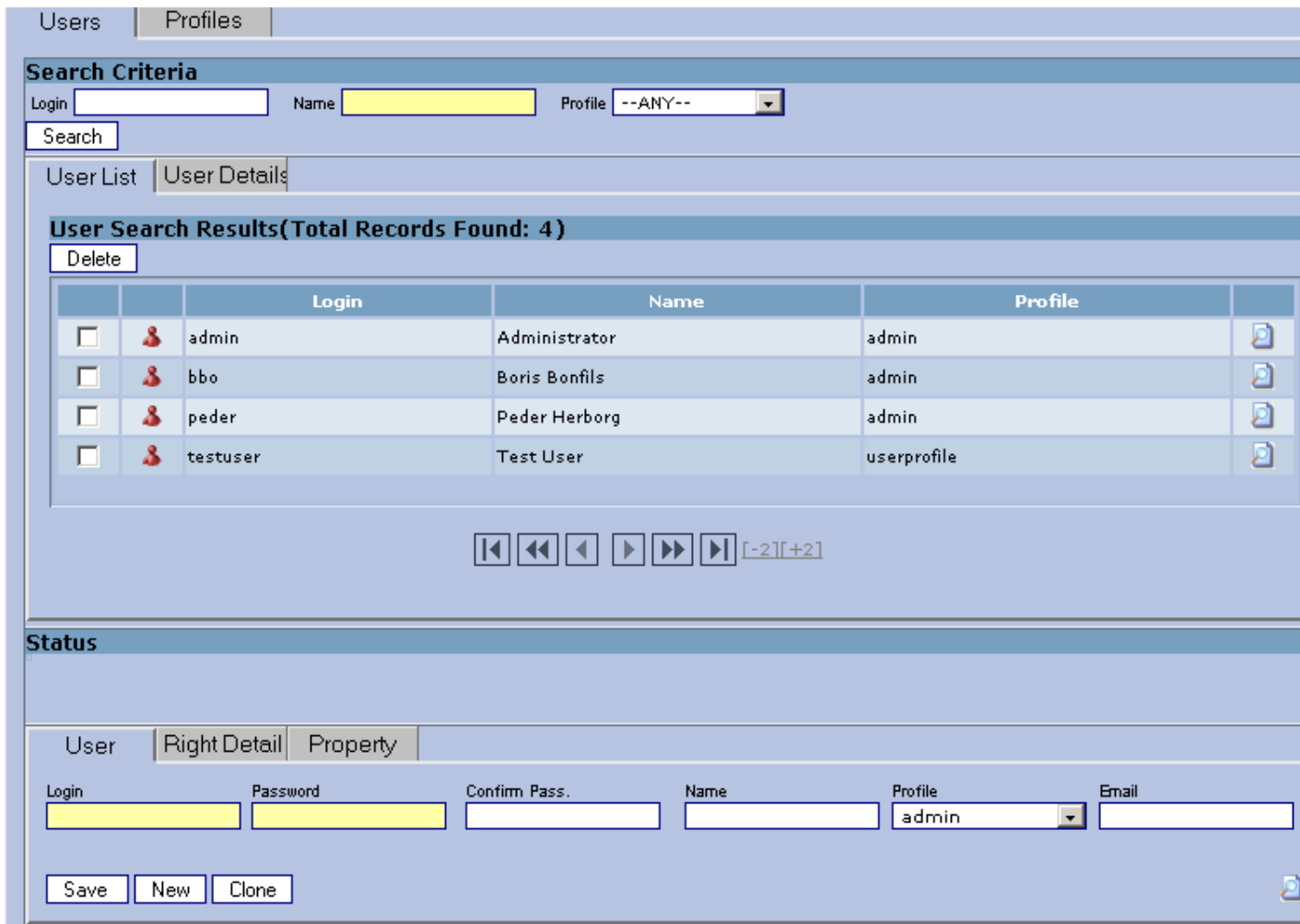
4.4 User management GUI

The user management GUI can be reached from a browser entering the url:

<http://host:port/ADK/rightsman.faces>

Or by using the AGETOR shortcut link *administration* described above.

The user management gui is used for creating and altering users and user settings in the AGETOR system. Below is a screenshot of the administration GUI.



Screenshot 1: User Management GUI

4.4.1 Users and profiles

The user management GUI layout consists of 2 separate tabs, one for users and one for profiles. Once a tab is selected the presented view is separated into 3 parts.

- The top part of the screen contains the search criteria, allowing the user to find and edit a subset of the entire set of users/profiles. Pressing search will by default find all available data as no specific search criteria has been entered.
- The middle part shows the found subset of the search. Here data can be deleted by checking the checkbox of each found user/profile to be deleted and pressing the Delete button. It is also possible to view the tree details of a user/profile by clicking the details icon (🔍) which will bring up the tree view described in 4.4.2
- The bottom part shows the details of the selected item in the middle view. For users this means the login name, user name, password etc. Here a user can switch password, change profile, user data etc. It is also possible to clone a selected user or profile and to create a completely new instance.

4.4.2 Rights and Properties

Rights and properties for a user or a profile are tree structures. These are built from template or default tree structures, and for users this involves merging of multiple trees into a result tree.

Both rights and property trees have a template tree which contains all possible nodes together with a type and description of each node. All rights and property trees use the template tree for construction of data. Furthermore users usually have both a profile and a set of locale rights and properties. This is due to fact that many users might have almost similar rights and properties and thus would gain from using the same profile. However there are a few places in the tree where the rights and properties differ for each user. These places are described in local tree and the rest are handled by the assigned profile.

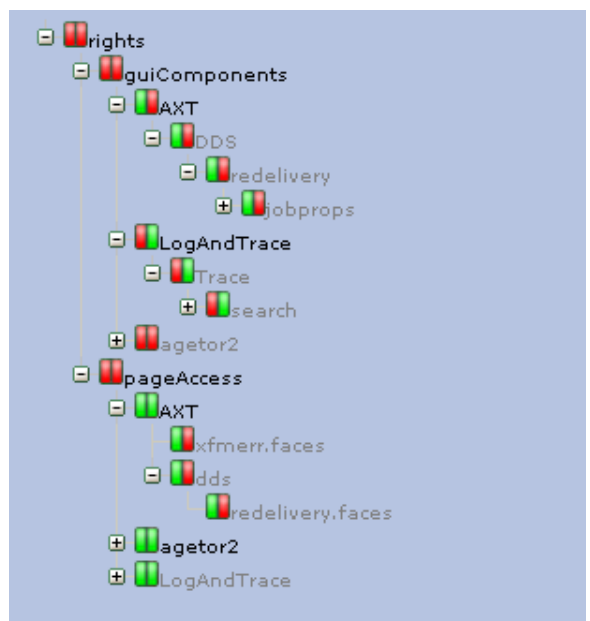
🔔 Locally assigned rights or properties for a user always overwrite the ones for the given profile, but in order to change the profile one must edit the profile not the user.

4.4.2.1 General Tree Navigation

To describe the general navigation we use the rights tree illustrated to the right to describe the different actions.

A tree is by default displayed with all nodes. This means the All Options checkbox is checked by default. However there might be some nodes in the tree which are not explicitly defined for this user/profile. When the all options is turned on these nodes are shown With a dimmed grey color like for instance the LogAndTrace node in the bottom of the tree to the right. Clicking the All Options checkbox off will cause all dimmed nodes to disappear from the tree.

A tree can be expanded and collapsed at each level by clicking the + and – icons for the sub tree to be expanded or collapsed.







Clicking a node will bring up the node details for that node in the node detail view which always is displayed below the tree. The editing of details in a tree is described below for rights trees and property trees.

Holding the mouse over a node for a little while will bring up a description for this node the purpose of the right/property etc.

4.4.3 Rights

When selecting a node in the tree, the details for this node is shown in the details view. For rights trees the details for a node are shown below. At the very top the path in the tree is shown with the node names separated by a dot (.) ie. (PageAccess.AXT) refers to the node child of the node PageAccess name AXT.

User		Right Detail	Property		
pageAccess.AXT					
Type	Derived from	On	Off	Undef	Disable
 read	userprofile profile	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
 write	userprofile profile	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
 Subtree read	userprofile profile	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
 Subtree write	Test User user	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Save					





A rights node has 4 separate settings namely: read, write, sub-tree read and sub-tree write. Each of these can be set separately, the read and the write setting determines the read and write access for this particular node in the tree. Where as the sub-tree read/write does not say anything about the access to the current node, but restricts/grants access to its children until overwritten on sublevel.

There 3 usual states for a right/setting and it can only be one of these hence the radio button layout:

- When on the light is green and the user/profile has access.
- When off the light is red and the user/profile does not have access.
- When undefined the state is determined by the parent node or somewhere higher in the tree.


The Derived from field tells the user if this right/setting is set explicitly by the user or inherited from another tree for instance a profile.


When viewing a rights tree, in front of each node a double red/green light box icon is shown determining the current right state of this node, ie:

-  read access denied/write access denied
-  read access denied/write access granted
-  read access granted/write access granted
-  read access granted/write access denied

4.4.3.1 Page access rights

In the current release rights are only used to restrict users from accessing specific web pages. The Page URL is described as nodes in the tree until the actual page is reached. This means that the following [URL:http://localhost/AXT/DDS](http://localhost/AXT/DDS) will result this lookup in the tree: pageAccess.AXT.DDS To prevent access to this URL and all sub URLs simple set the read and sub-tree read rights for this node to off.

 In order for changes to take effect the user(s) affected by this must logoff and login again.

 Only the read access to a page is checked.

4.4.4 Properties

The properties are structured as a tree to achieve a natural grouping of properties that relates to one another. There is no inheritance and property values can only be seen by choosing a node in the property tree. The property details view is shown below. A property contains a type and a value. The property needs to be explicitly defined by using the Def/Undef radio button.



4.4.4.1 Trace properties

The following properties exist to restrict the traces that can be searched and modified by a user:

Property name	Description
trace.group	This property states the group to which the user belongs. If the group is stated (and no viewgroups property is stated) then the user will only be able to search for traces that have the same trace group value. (The trace group value is typically set during AXT processing)
trace.gui.viewgroups	If defined, this property states the trace groups that the user may search for. I.e. if present it overrules the users group property. The value is a semi-colon separated list of groups. E.g.: group 1; group 2; ...
trace.gui.search.property._<N>.type	The type of an addition generic trace restricting property (number N). Valid values are “trace” and “document” depending on what kind of property you wish to restrict on. Please refer to the Log & Trace guide for information on trace and document properties.
trace.gui.search.property._<N>.name	The name of an addition generic trace restricting property (number N). Please refer to the Log & Trace guide for information on trace and document properties.
trace.gui.search.property._<N>.operator	The operator of an addition generic trace restricting property (number N). Valid operators are =, in, <, >, <=, >=, like

	<p>The value (see below) must conform to the operator – if using the like operator you need to use percentage wildcards (%) in the value field. When using the in-operator you should provide a comma-separated list of values. Strings should be single-quoted.</p> <p>Please refer to the Log & Trace guide for information on trace and document properties.</p>
<p>trace.gui.search.property._<N>.value</p>	<p>The value of an addition generic trace restricting property (number N). Please refer to the Log & Trace guide for information on trace and document properties.</p>

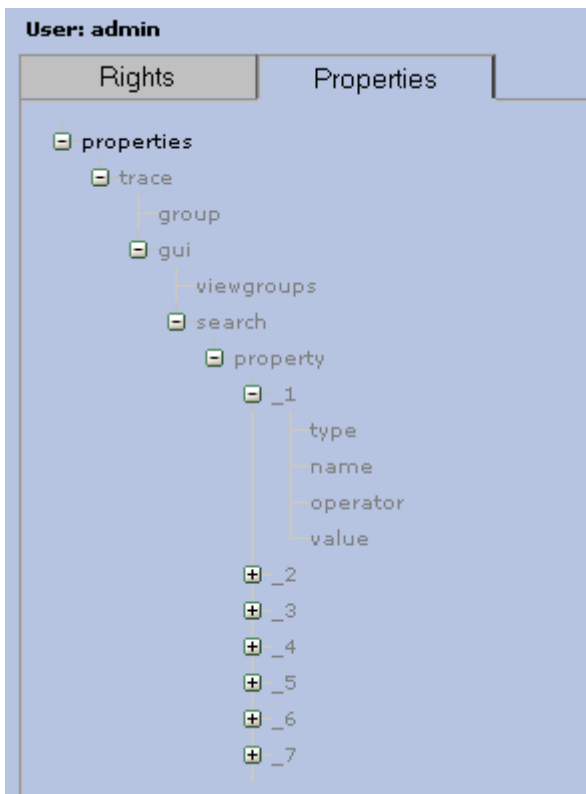


Figure 4. trace properties for a user in the property administration

4.5 Configuration database

The user information is stored in the derby database `cfgdb` and made available with the AGETOR `cfg-service` which is started with the Service Manager (aka Service Runner).

4.6 Management of AGETOR supported databases

At present the logging and configuration databases are the only databases shipped with AGETOR. These have the internal names `logdb` and `cfgdb`.

4.6.1 Resetting users and profiles using system tool

A command exist for recreating the information in the configuration database database (users, profiles, rights and properties). This is done by invoking a script from the command-line:

```
> db-reset-users
```

Issuing this command will 1) recreate all tables of the configuration database and 2) insert the default user, profile and the administrator (with credentials admin/admin).

4.6.2 Other database management commands supported with ADKDBTool

The above script wraps the execution of the java class dk.bording.inside.db.ADKDBTool. This class provides a few database management commands described below. Please note that the effects may be irreversible so it is highly recommended to backup the database before using these commands.

4.6.2.1 Recreating an AGETOR supported database

To recreate these databases (delete all tables and recreating them) you may issue the following command. *Please note that all data are lost in the process:*

```
> j dk.bording.inside.db.ADKDBTool --cmd:export --db:<db-name>
```

Where db-name is one of the supported databases.

The configuration database (cfgdb) will need to have some default users created in order to have anybody login subsequently. The default users and profiles may be created with the command:

```
> j dk.bording.inside.db.ADKDBTool --cmd:createAdmin
```

Issuing this command without the previous export command will result in errors.

4.6.2.2 Updating an existing database schema

It is possible to update the tables of a supported database such that tables defined in the Hibernate schema descriptions which are not part of the currently installed database will be created and tables that have been modified to have new columns are updated accordingly.

The command below updates the Log & Trace database:



```
> j dk.bording.inside.db.ADKDBTool --cmd:update --db:logdb
```

4.7 Database Management GUI

An alternative to managing the AGETOR supported databases logdb and cfgdb described in section 4.6 is to use database GUI found on the administration pages together with user and profile management.

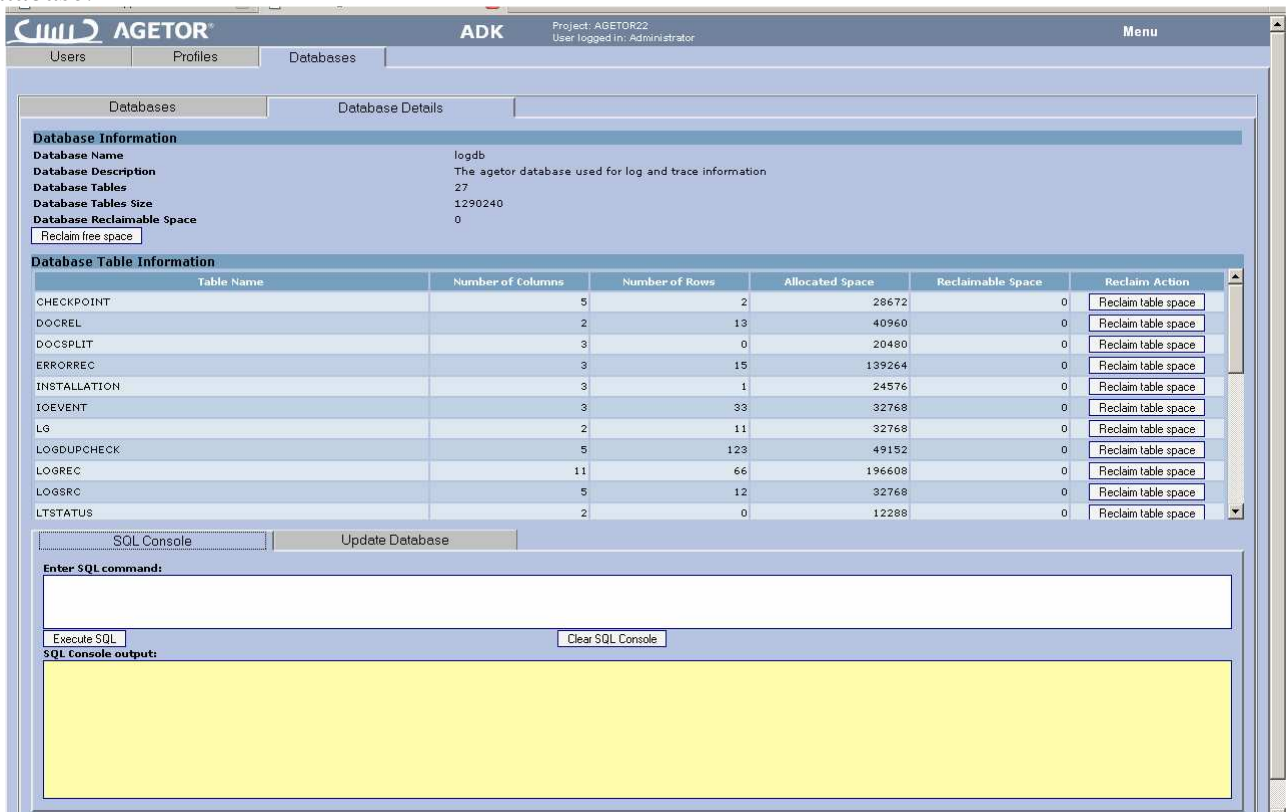
4.7.1 Database Selection

Before managing a databases detail, a database must be selected by clicking on the details icon. See screenshot below where the two AGETOR supported databases are listed.

Database Name	Database Description	Details
cfgdb	The agettor configuration database used for user profiles	
logdb	The agettor database used for log and trace information	

4.7.2 Database Details Management

When a database has been selected, the details for the selected database are displayed in the Database Details tab. This information is mainly number of tables, columns and rows in the database.



Database Information

Database Name: logdb
 Database Description: The agettor database used for log and trace information
 Database Tables: 27
 Database Tables Size: 1290240
 Database Reclaimable Space: 0

Database Table Information

Table Name	Number of Columns	Number of Rows	Allocated Space	Reclaimable Space	Reclaim Action
CHECKPOINT	5	2	28672	0	<input type="button" value="Reclaim table space"/>
DOCREL	2	13	40960	0	<input type="button" value="Reclaim table space"/>
DOCSPLIT	3	0	20480	0	<input type="button" value="Reclaim table space"/>
ERRORREC	3	15	139264	0	<input type="button" value="Reclaim table space"/>
INSTALLATION	3	1	24576	0	<input type="button" value="Reclaim table space"/>
IOEVENT	3	33	32768	0	<input type="button" value="Reclaim table space"/>
LG	2	11	32768	0	<input type="button" value="Reclaim table space"/>
LOGDUPCHECK	5	123	49152	0	<input type="button" value="Reclaim table space"/>
LOGREC	11	66	196608	0	<input type="button" value="Reclaim table space"/>
LOGSRC	5	12	32768	0	<input type="button" value="Reclaim table space"/>
LTSTATUS	2	0	12288	0	<input type="button" value="Reclaim table space"/>

SQL Console |

Enter SQL command:

SQL console output:

4.7.2.1 Derby database specific details

The database shipped AGETOR is derby. AGETOR ships with a few extra features for managing the derby database specifically. This includes calculation of allocated space for the database and manual data reclaim of unused space. This is due to the fact that derby does not automatically frees allocated data space that is unused due to deletion. This has to be done manually by either clicking the “Reclaim free space” button for the selected database, or by explicitly reclaiming space for a given table.

4.7.2.2 SQL Console Tab

The SQL Console tab provides the possibility to execute SQL statements manually and displaying a possible output in the SQL Console Output text area. This gives the user the possibility to create specific search statements which cannot be formulated using the provided GUI. Also updates and database specific commands can be executed here.

4.7.2.3 Update Database Tab

The Update Database tab is used to update/upgrade from a previous version of AGETOR database, incase of new product updates which requires changes in the database structure. There are 3 main functions in this tab:

1. Update Database Tables
2. Restore Database Default
3. Run Update Script

1. The Update Database Tables updates the tables in the database with the latest columns, indexes, etc. that comes when upgrading an existing installation to a new version.

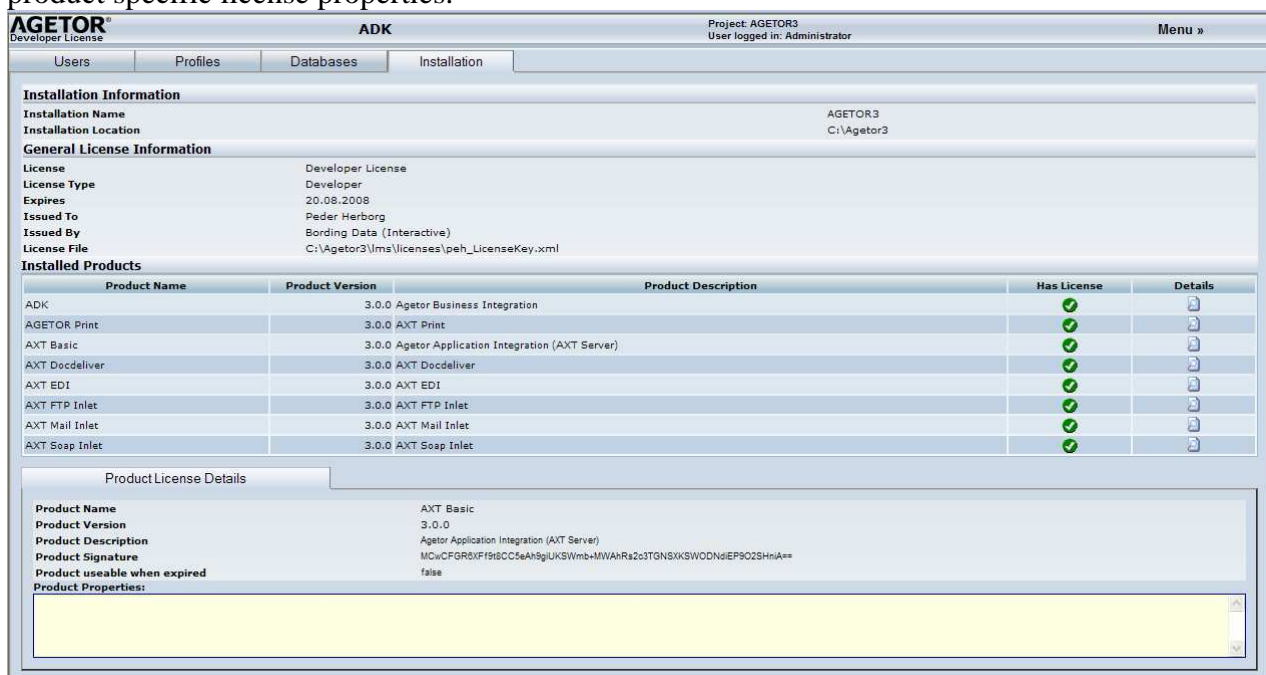
2. The Restore Database Default restores the database to its default state.

This means that all data in the database will be deleted

3. When installing an update it is possible that there have been changes in the database structure that requires an update script to be run, before the installation can continue to function properly. This will be clearly stated during the installation process, if this is necessary.

4.8 Installation Management GUI

With the introduction of the new license system in AGETOR 3.0 a need to view the installed products and licenses arose. Therefore the Administration web pages has a tab called Installation, which shows the details for a given installation, license details for the installed products, and product specific license properties.



The screenshot shows the AGETOR Administration web interface. At the top, there is a navigation bar with tabs for Users, Profiles, Databases, and Installation. The Installation tab is active. Below the navigation bar, there is a section for 'Installation Information' showing details for 'AGETOR3' installed at 'C:\Agetor3'. A 'General License Information' section follows, detailing the license type (Developer License), expiration date (20.08.2008), and issuer (Peder Herborg). Below this is a table of 'Installed Products' with columns for Product Name, Product Version, Product Description, Has License, and Details. The table lists several products, all of which have a 'Has License' status of 'true' (indicated by a green checkmark). At the bottom, there is a 'Product License Details' section for the selected 'AXT Basic' product, showing its version (3.0.0), description (Agetor Application Integration (AXT Server)), and a long alphanumeric signature.

Product Name	Product Version	Product Description	Has License	Details
ADK	3.0.0	Agetor Business Integration	✓	
AGETOR Print	3.0.0	AXT Print	✓	
AXT Basic	3.0.0	Agetor Application Integration (AXT Server)	✓	
AXT Docdeliver	3.0.0	AXT Docdeliver	✓	
AXT EDI	3.0.0	AXT EDI	✓	
AXT FTP Inlet	3.0.0	AXT FTP Inlet	✓	
AXT Mail Inlet	3.0.0	AXT Mail Inlet	✓	
AXT Soap Inlet	3.0.0	AXT Soap Inlet	✓	

Here the general installation and license information is shown, together with a list of installed products, and whether they have a license or not.

4.9 Technical interaction and API

4.9.1 The LoginFilter

The login filter web filter redirecting all request to the web server through this filter. This configured in the web.xml for the web server. The filter checks if the user is logged in to the system before allowing access to the requested page. If the user not logged in to the system, the filter will redirect the user to the login web page.

4.9.1.1 Disabling the LoginFilter

The LoginFilter is enabled by default, but can be disabled either by manually editing out the LoginFilter configuration in the web.xml for the web server or by disabling it in the AGETOR_HOME/conf/properties/adminprivate.properties property file.

```
# Bypass the login filter, allowing access to all webpages regardless of login
agetor.login.filter.disabled=true
```

4.9.1.2 Excluding URL's from LoginFilter

Some times it is not desirable that a servlet requires a login to be used, for instance when there is no user involved in the communication process. Therefore the LoginFilter allows pages to be excluded from the Login restriction. This is illustrated below, where the relative URL's are consecutive numbered exclude pages in the AGETOR_HOME/conf/properties/adminprivate.properties property file:

```
# The Login Filter excludes the following pages from redirection to Login Page
agetor.login.filter.excludepage.1=/servlet/
agetor.login.filter.excludepage.2=/soapinlet/
```

4.9.2 PageAccessFilter

The PageAccessFilter is a web server filter like the LoginFilter. It restricts users from accessing specific pages in the web server if the user does not have access rights. In order for the pageAccessFilter have any effect a user must be logged in.

4.9.3 URL-based login and external property overriding

4.9.3.1 Automatic URL login

It is possible to avoid the manual entering of username and password by supplying the username and password as URL parameters accessing a page. This will cause a redirection the Login Filter which will retrieve the username and password and attempt to login. If the login is successful the user will be redirected to the originally requested page automatically.

Example:



```
http://localhost/agetor2/login/login.jsp?username=admin&password=admin
```

4.9.3.2 URL Property settings

When automatically logging in a user with URL parameters it is possible to override properties set for this user by manually supplying these properties which needs to be overwritten as URL parameters on the request URL.

Example

```
http://localhost/agetor2/login/login.jsp?username=admin&password=admin&_trace.group=group1
```

-  Each property is prefixed with an underscore(_) to minimize the risk of name clashing.
-  Note that the property is identified with its complete path and that the value must be URL encoded if necessary.

5 Configurable persistence of AXT transformation data


AXT offers the possibility to configure persistence of data during the transformation process. Subsequently documents failed during a transformation step may be inspected. Furthermore one can configure AXT to store the documents and each step during the transformation process to follow the process and possibly track down errors which occurred earlier in the transformation pipeline, but where not reported.

5.1.1 Data Storage Modes

There are four different settings which can be used from the AXT configuration file to determine when to store data.

Mode	Description
ALWAYS	Saves all datasets regardless of error status
NEVER	Does under any circumstance save any data
ONERROR	Saves all datasets in the transformation, if any of them are in error otherwise it does not store anything.
ONLYERRORDATA	Stores the datasets in error, but not the ones which where successful

These modes/settings are configurable on three different levels: globally, on document entry level and on filter level, each overruling the former.

-  The actual setting is the four modes are case insensitive

5.1.2 Global configuration

The default AXT setting is to never store transformation data.

This setting can be found in the `AGETOR_HOME/conf/properties/axt.properties` file:

```
# The default behaviour for AXT is not to store the documents
# There are 4 possible settings: never,always,onerror,onlyerrordata
axt.configuration.storedata=never
axt.configuration.storedata.location=${AGETOR_HOME}/data/axt/documents/
```

Here the location where to store the document data and metadata for each transformation step is also configurable.

5.1.3 Document Entry Configuration

On document entry configuration the global setting `axt.configuration.storedata` can be overruled locally with the attribute `storedata` (illustrated in the example below).

```
<doc name="hello" storedata="always">
```

```
<key name="function" value="hello"/>
<filter class="dk.bording.axt.tc.xml.XSLProcess">
  <param name="xsl-script" value="\${AGETOR_HOME}/conf/axt/examples/basic/hello.xml"/>
</filter>
</doc>
```

Setting the `storedata` attribute to "always" will cause the document and metadata to be stored before each filter invocation in the document entry. In the example above, even though the global setting is set to never store data, this document entry will store input document and metadata before invoking the `XSLProcess` filter.

In the example below the input document and metadata are only stored if the `XSLProcess` filter fails.

```
<doc name="hello" storedata="onerror">
  <key name="function" value="hello"/>
  <filter class="dk.bording.axt.tc.xml.XSLProcess">
    <param name="xsl-script" value="\${AGETOR_HOME}/conf/axt/examples/basic/hello.xml"/>
  </filter>
</doc>
```

5.1.4 Filter Level Configuration

The third possibility is to configure the data storage functionality on filter level. This overrules both the setting on document entry level, and on global level. In the example below there are three settings in effect. However the only active one for this example is the `storedata = "always"` on the filter level since it overrules all previous settings.

```
<doc name="hello" storedata="onerror">
  <key name="function" value="hello"/>
  <filter class="dk.bording.axt.tc.xml.XSLProcess" storedata="always" >
    <param name="xsl-script" value="\${AGETOR_HOME}/conf/axt/examples/basic/hello.xml"/>
  </filter>
</doc>
```

In the next example we have configured the document entry only to store data on errors. This means that filters will be executed in a serial fashion and if one fails only the input data for this filter is stored. Input data for previous filters which succeeded are not stored. Furthermore in the example below the document entry setting is overwritten by a filter which says that even though this filter fails, AXT should not store the input data here.

```
<doc name="hello" storedata="onlyerrordata">
  <key name="function" value="hello"/>
  <filter class="dk.bording.axt.tc.xml.XSLProcess">
    <param name="xsl-script" value="\${AGETOR_HOME}/conf/axt/examples/basic/hello.xml"/>
  </filter>
  <filter class="dk.bording.axt.tc.xml.XSLProcess" storedata="never" >
    <param name="xsl-script" value="\${AGETOR_HOME}/conf/axt/examples/basic/hello2.xml"/>
  </filter>
</doc>
```

5.1.4.1 Local Call Filter and XMLSplitFilter

The local call filter is a special filter case which requires special handling of the `storedata` mode. Like any other filter the input to the local call filter can be stored with the `storedata` attribute, but unlike most other filters the local call filter invokes a set of new filters. The `storedata` value to these filters can be set using the `childrenstoredata` attribute, and the result is that the document entry's `storedata` data attribute (invoked by the local call) is overruled by the value set by the

childrenstoredata filter. If the childrenstoredata is not defined the storedata value from the documententry of the localcall is inherited instead.


5.1.5 Performance Issues

Adding storage of data files between each filter transformation for AXT Transformation naturally comes with a considerable performance penalty. How big the performance penalty is depends on the various different aspects which we describe in detail below:

- Storedata mode (never,always,onerror,onlyerrordata)
- Size of the data
- Complexity of the transformations compared to the document size
- Disk speed and CPU speed.


The performance can greatly depend on the storedata mode for the transformation. In the schema below the expected performance impact is listed (based on statistics) with an average of various sizes and number of transformations. The index is the time used, thus an index of 1.5 means that the transformation takes approximately 50% longer to complete, compared to the indexed transformation.

Mode	Performance degradation	Performance
NEVER	Doesn't save data and thus is used as an index	1
ALWAYS	Saves all datasets during the transformation, when the transformation is complete moves the data to AXT documents repository.	2
ONERROR	Saves all datasets during the transformation and then discards them as there was no error.	1.5
ONLYERRORDATA	Saves all datasets during the transformation and then discards them as there was no error.	1.5

 All transformations ran without errors.

It is important to realise that no matter which storedata mode that is chosen besides “never”, a considerable performance impact may be expected since AXT will need to store the data input for each filter. Only when the entire transformation has run to an end it can be determined whether to permanently store the data or not. For small data this data can be buffered in memory, thus speeding up the process. For large document data AXT must store the data temporarily on disk until it has been determined what to do with it. Thus for small documents the performance impact will be considerably lower than for large data, as all temporary data can be kept in memory. For large document data both the AXT cache and the System disk cache will be stressed as large amounts of data is written to disk and then later deleted.

The size of the memory buffers can be tuned in the `axt.properties` configuration file (see the AXT User Guide for details).

 For large data files with the storedata turned on, expect a performance slowdown of a factor of 2 unless the filter processing of the data is time consuming (dominating). For

smaller data files expect the processing to be a little slower than normal, but not more than a maximum of 10-15%.

 Tune the AXT buffer size for better performance in concrete scenarios

5.1.5.1 Timeout Issues

Due to the added processing time when storing data, timeout problems may arise when using the store data option. This is also the case with the settings: `onerror` and `onlyerrordata`. This is because AXT often has to store all data, because it cannot before the transformation has ended whether or not it was successful.

Currently this has been noticed when using the `DocdeliverOutlet` filter. This is due to the fact that AXT now uses a bit more time on the processing due to the storage of data process. These timeout problems can occur in two different scenarios which look like this:

1. `dk.bording.inside.orb.RemoteTimeoutException: Input timed out`
This problem can usually be overcome by tuning the `brokerTimeout` parameter on the `docdeliverOutletFilter`. The timeout used for the communication with the DDS.
2. `dk.bording.axt.AXTProcessException: Error ERR_SERVICECONTACT`
This problem can usually be overcome by tuning the `RESPONSE` attribute in the `broker.cfg`. Which is the number of seconds the broker will wait for a service to respond to a client before timing out and sending a `NO_SERVICE_CONTACT` error to the client.
This error is usually due to the fact that the `DocdeliverOutletFilter` is running in synchronized mode and thus must keep an open connection to await completion of the job.

6 Configuring periodic cleanup of failed/succeeded dataset

The persisted data and job-files stored with DDS and AXT may be deleted when they reach a certain age. This is recommended to avoid build-up of large space-consuming files. Especially if AXT is configured to persist much data (always option), setting the delete threshold low is recommended.

The configuration of periodic delete is specified in the DDS configuration file with the tags shown below.

```
<deleteFailedJobs schedule="0 10 * * * ?" age="2 weeks"/>
<deleteSucceededJobs schedule="0 20 * * * ?" age="2 weeks"/>
<deleteAXTErrors schedule="0 30 * * * ?" age="2 weeks"/>
```

The tags are simply place inside the `docdeliver` root-tag of the configuration.

The three tags specify the frequency of deleting, the age of datasets that qualifies them for deletion and of course the type of files in question (failed DDS jobs, succeeded jobs or AXT datasets).


The frequency is defined using a cron-expression. In the example these expressions define that the deletion check is performed every hour at 10, 20 and 30 minutes past the hour. Only datasets that are more than 2 weeks old are deleted in the example.

The `age` attribute accepts expressions like:

1 day, 10 days, 7 minutes 30 seconds, 2 month, ...

Etc.

Please refer to the Quartz homepage description on cron-expressions for details on these:
<http://quartz.sourceforge.net/javadoc/org/quartz/CronTrigger.html>

 *Not defining the periodic cleanup will result in usual behavior. I.e. no deletion of DDS-files (failed or succeeded). However the AXT datasets are removed after 2 weeks by default.*

7 JOBS GUI

A new GUI for managing DDS jobs has been implemented. This GUI allows for searching, ordering, grouping jobs from multiple DDS servers in the system.

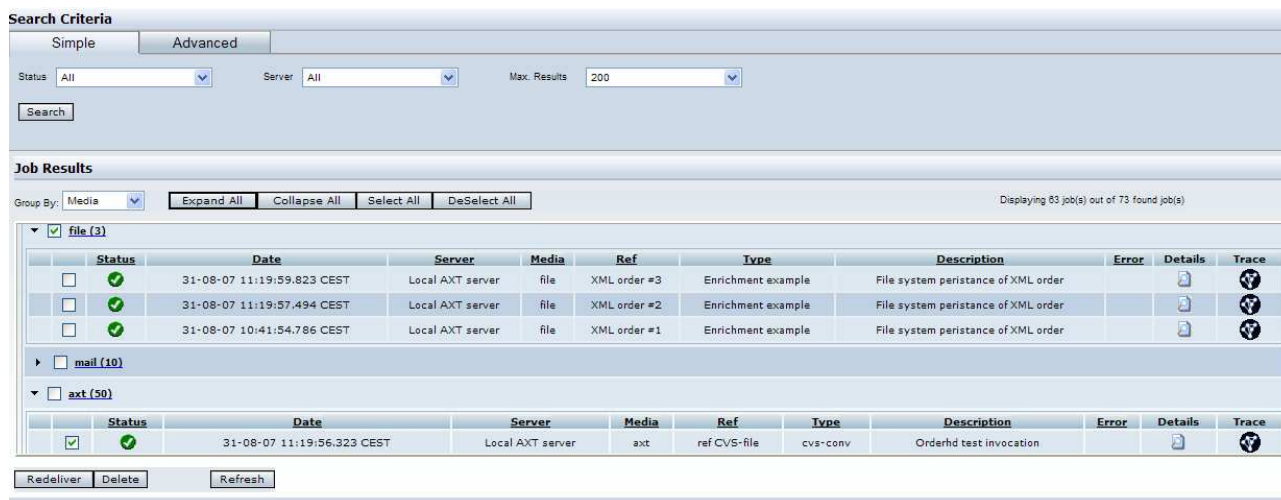
Job status and information is shown in result sets from which selected jobs may be deleted, redelivered and refreshed with data from the L&T system (job enrichment).

Job details may also be inspected and modified and corresponding data files may be downloaded locally for inspection and modification and subsequent upload and redelivery. The details of the job are shown in a separate window (see section 3.1.2.1 for a screen dump of this interface).

Jobs have link to their corresponding traces in the trace system.


The jobs may be enriched with information (document and trace properties) from the L&T system but this require configuration.

The JOBS GUI allow for flexible configuration of new columns of such data as well as configuration of search fields for specific information items.



The screenshot shows the JOBS GUI interface. At the top, there are tabs for 'Simple' and 'Advanced' search criteria. Below these are dropdown menus for 'Status' (set to 'All'), 'Server' (set to 'All'), and 'Max. Results' (set to '200'). A 'Search' button is located below these fields. The main area is titled 'Job Results' and shows a 'Group By' dropdown set to 'Media'. There are buttons for 'Expand All', 'Collapse All', 'Select All', and 'DeSelect All'. A status bar indicates 'Displaying 83 job(s) out of 73 found job(s)'. The results are grouped into three categories: 'file (3)', 'mail (10)', and 'axt (50)'. The 'file' group is expanded, showing a table with columns: Status, Date, Server, Media, Ref, Type, Description, Error, Details, and Trace. The table contains three rows of job data. Below the table, there are buttons for 'Redeliver', 'Delete', and 'Refresh'.

Figure 5. The JOBS GUI

 Please refer to the AXT JOBS GUI user and installation guides in your installation for more information.

8 Architectural changes

8.1 ADK DB-service (vs. log service)

A new service is being shipped with ADK 3.0. The derby database server which was an integrated part of the L&T service now runs in its own service, the ADK DB-service. Besides starting the database network server, the service exposes IDL-functions for retrieving user rights. It is the plan


that the service will later be expanded to include configuration support for the distributed ADK/AXT system.

The service is automatically installed with ADK and started with the service-runner. The Log-server will now use the database embedded in the DB-service rather than starting its own database.

The DB-service is started first, then the L&T-services and then the rest of the services in the service runner. This ordered start-up is configured in the service launch-configurations. As a consequence of this division, the L&T service may be started and stopped without influencing the user management and configuration functions of the DB-service.


However, for the L&T-service to be able to work properly you must ensure that the DB-service is running first.

8.2 Configuring the distributed system (multiple installations)

 *This section is only relevant if you use multiple communicating installations (projects). Single machine installations run out-of-the box.*

In a system using multiple communicating AGETOR installations (e.g. with a secure/semi-secure network division), the choice of a central log-server and a central db-server must be made. These components are then shared by all components of the system (applications logging or requesting user rights information etc.).

The log- and db-servers communicate with clients using IDL-methods thus the broker configurations of the installations must define how to reach the servers from all involved installations. This section describes how this configuration may be done.

 *At present the L&T management and the User rights management GUI's communicate directly with the db-service and thus require direct JDBC access to the server. I.e. management from other machines may not work if firewalls are in between.*

8.2.1 How to set up logging server routing

If the log-service is located at machine:project M1:P1 and another project M2:P2 wishes to use the log-server, then the log-service requests from P2 are simply routed to P1 in the broker configuration.

8.2.1.1 Direct access to the remote log-service

In the example below we pinpoint the address of the remote log-service (m2:30050). This requires ability to establish direct socket connection to the service and does not use whatever remote broker routing would be setup.

On m1:

```
<SERVICES>
  <SERVICE NAME="logserver" HOST="m2" PORT="30050" />
</SERVICES>
<ENVIRONMENT NAME="agetor">
  <QUESTION NAME="logserver" QNO="118" />
</ENVIRONMENT>
```

8.2.1.2 Indirect access to the remote log-service using the remote broker as proxy

This example forwards the request directly to the broker on the other machine. I.e. the configuration of the log-service on the other broker is used.

On m1:

```
<SERVICES>
```

```
<SERVICE NAME="m2broker" TYPE="broker" HOST="m2" PORT="20002" />
</SERVICES>
<ENVIRONMENT NAME="agetor">
  <QUESTION NAME="m2broker" QNO="118" />
</ENVIRONMENT>
```

8.2.1.3 Indirect access to the remote log-service using firewall tunnelling

On m1

```
<SERVICES>
  <SERVICE NAME="m2broker" TYPE="broker" IDENT="unique_id" REACHABLE="false"
CONNECTING="true" />
</SERVICES>
<ENVIRONMENT NAME="agetor">
  <QUESTION NAME="m2broker" QNO="118" />
</ENVIRONMENT>
```

On m2

```
<FIXEDCLIENTS>
  <BROKER NAME="m2broker" HOST="m1" PORT="20002" IDENT="unique_id" />
</FIXEDCLIENTS>
```

Please consult the ARE_guide (AGETOR Runtime Guide) for more information on firewall tunnelling options.

8.2.2 How to set up configuration server routing

Configuration of db-server (configuration and user rights) is done in a similar way to the log routing described above. Only the used QNO is 119 and the relative port number is 51.

E.g.

```
<SERVICES>
  <SERVICE NAME="dbserver" HOST="m2" PORT="30051" />
</SERVICES>
<ENVIRONMENT NAME="agetor">
  <QUESTION NAME="dbserver" QNO="119" />
</ENVIRONMENT>
```

8.2.3 Mapping L&T application information to AXT/DDS-servers

When multiple machines run DDS delivery and AXT transforms they send trace information to the central L&T service (in the usual configuration). However, some of the information – such as error and job details and the data files themselves are *not* sent to the central L&T server. This would simply be too inefficient and resource consuming.

Rather, the datasets and error/delivery information is held locally. Since ADK/AXT 3.0, access to this information is achieved through IDL-invocation of the DDS-service on the machine in question. The DDS-service is able to delivery the details on error/delivery as well as the stored datasets.

Since remote machines may be know by different names on the local machine a mapping file is required. This mapping is from application id's (that came with trace events).

To request the dataset from the DDS-service, we need to know the environment in the broker that should be queried – i.e. how we communicate with the DDS-server.

The XML configuration file located at AGETOR_HOME/conf/axt/servers.xml contains the mapping from L&T application source information onto broker-environments.

For each DDS (AXT)-service running in the total system a server entry is made in this file. The server entry has a unique id number and an AXT URL on the form

"axtorb://<host>:<port>/environment". I.e. the URL describes in one string the address to contact to speak with the DDS server using ORB-communication.

For each DDS server entry a number of mappings from application id's to the server may be registered. At present only two application types are supported; `axt` and `dds`, since only these two logical application scopes can communicate datasets and function internally in the system. If the URL has no `<host>` specification, the default host is used. If there is no environment, the default environment is used.

Application id's are strings with up to three components; 1) the hostname that came with the trace message, 2) the project name that came with the trace message and 3) the application type that came with a function event (`axt` or `dds`). Thus when the DDS-service on machine *m1* project *p1* generates an error and stores data it sends the following information: *m1/p1/dds*. To later request data we lookup the axtorb URL and request the right service.

The need for this file stems from the fact that the name of a certain machine may differ from one machine to the other. Also firewall-tunneling can be required and hidden by mapping to a certain environment that defines the specific way of reaching the DDS-service.

```
1. <?xml version="1.0" ?>
2. <axt-server-mapping-conf>
3.
4. <!--this default mapping states that the default environment on the local machine should be
used for all applications lookups from the local machine and project.
5. <server id="1" url="axtorb:///" description="Local AXT server (through local broker)">
6.   <applicationMapping applicationid="//dds"/>
7.   <applicationMapping applicationid="//axt"/>
8. </server>
9. <!-- To map a function from dds on machine m1 project p1 to the agetor3 environment on the
local broker configuration the following may be defined
10. →
11. <server id="2" url="axtorb://agetor3" description="Remote server through agetor3
environment routing">
12.   <applicationMapping applicationid="m1/p1/dds"/>
13.   <applicationMapping applicationid="m1/p1/axt"/>
14. </server>
15.
16. </axt-server-mapping-conf>
```

With the above configuration all `axt/dds` functions in the L&T database are resolved to DDS requests on either the local host and default environment if the source is the local host and local project OR to DDS request on the local host (broker) on environment `agetor3` if the source is an AXT or dads application running on host *m1* project *p1*.

Note that the latter requires a definition of environment `agetor3` in the broker configuration and a DDS-service defined there.

To directly request the information from the DDS-service the following could be stated instead:

```
17. <server id="2" url="axtorb://m1:20030/agetor3" description="Direct dds access">
18.   <applicationMapping applicationid="m1/p1/dds"/>
19.   <applicationMapping applicationid="m1/p1/axt"/>
20. </server>
```

Assuming the DDS service is running on `m1:20030`. Obviously there is no firewall tunnelling here.

8.2.3.1 The default mapping from application to environment

The `servers.xml` mapping file has a default mapping of application names to dynamically constructed URL's. This mapping will be used for all application names that are not explicitly mapped to a static URL as described above.

The mapping maps the request for AXT or DDS information to the local host on an environment name constructed from the name of the machine and project from which the original trace event came. I.e. if an AXT server on machine M1 and project P1 created a trace function in L&T, then the subsequent request for data from the AXT server would go through the environment named M1:P1 on the local host. Thus the environment must be defined for such remote applications.

```
<dynamicMapping url="axtorb://localhost/{machine}:{project}" applicationid=".*"/>
```

8.2.4 Example setup with two machines

To enable redelivery and error view lookup (and future communication between AXT services), it is necessary to configure the routing between AXT/DDS servers in the broker configuration of each installation that participates in the distributed system.

In the following the recommended method is described.

We describe a setup with two installations. On machine H1 we have project P1. On machine H2 we have P2. The log service and db-service on P2 is used as the central logging server for both machines. There is a firewall between H1 and H2 so firewall tunnelling (broker-broker) is configured such that H2 establish the outgoing connection to H1.

Machine H1, project P1	Machine H2, project P2
<p>1) Configuring the remote machine as a service (this assumes broker-broker firewall tunnelling)</p> <pre><SERVICE NAME="broker-int" TYPE="broker" IDENT="unique_id" REACHABLE="false" CONNECTING="true" /></pre> <p>2) Configure the service mapping in the default environment. Log and configuration requests (118, 119) are sent to the internal broker.</p> <pre><ENVIRONMENT NAME="agetor"> <QUESTION NAME="broker-int" QNO="118,119"/> <QUESTION NAME="servicerunner" QNO="100"/> <QUESTION NAME="docdeliver" QNO="256"/> </ENVIRONMENT></pre> <p>3) configure the local services in the “local environment” which is named according to machine and project for convenience only:</p> <pre><ENVIRONMENT NAME="m1:p1"> <QUESTION NAME="logserver" QNO="118"/> <QUESTION NAME="dbserver" QNO="119"/> <QUESTION NAME="servicerunner" QNO="100"/> <QUESTION NAME="docdeliver" QNO="256"/> </ENVIRONMENT></pre> <p>4) [Optional] Configuring how to reach services on the remote project (defining the dads, log and</p>	<p>1) configuring that this broker makes a connection so that the other broker becomes a fixed client</p> <pre><FIXEDCLIENTS> <BROKER NAME="broker-int" HOST="godzilla" PORT="20002" IDENT="unique_id"/> </FIXEDCLIENTS></pre> <p>Besides the local services we define the DDS on H1 for direct access.</p> <pre><SERVICE NAME="dds-int" RPORT="30"/> <SERVICE NAME="dds-ext" PORT="20030"/></pre> <p>2) Configure the service mapping in the default environment.</p> <pre><ENVIRONMENT NAME="agetor"> <QUESTION NAME="logserver" QNO="118"/> <QUESTION NAME="dbserver" QNO="119"/> <QUESTION NAME="servicerunner" QNO="100"/> <QUESTION NAME="dds-int" QNO="256"/> </ENVIRONMENT></pre> <p>3) Configure services in the local environment which is named according to machine and project for convenience only</p> <pre><ENVIRONMENT NAME="m2:p2"> <QUESTION NAME="logserver" QNO="118"/> <QUESTION NAME="dbserver" QNO="119"/></pre>

<pre>cfg-services): <ENVIRONMENT NAME="m2:p2"> <QUESTION NAME="broker-int" QNO="256, 118, 119"/> </ENVIRONMENT></pre>	<pre><QUESTION NAME="servicerunner" QNO="100"/> <QUESTION NAME="dds-int" QNO="256"/> </ENVIRONMENT></pre> <p>4) Map the external DDS service in the “external environment”. This allows the local L&T GUI to find AXT and DDS files on the other machine through ORB-communication.</p> <pre><ENVIRONMENT NAME="m1:p1"> <QUESTION NAME="dds-ext" QNO="256"/> </ENVIRONMENT></pre>
--	---

The distinction between the default (used) environment (“agetor”) and the local environment is that the local environment is used for defining the services running in this project. The default (agetor) environment is the services to use and these may be present on other machines. The local-environment is *only* used for information look-up from Log & Trace at present.

When naming the environment according to the machine and project services relate to we automatically have a lookup from Log & Trace since the default mapping of the AGETOR_HOME/conf/axt/servers.xml states that trace function requests should be resolved by creating an ORB-request to the environment named <m>:<p> where m is machine and project is project of the tracing application.

9 Miscellaneous

9.1 New DDS parameters and properties

9.1.1 Retry pattern

The number of retries and the time interval between them may be specified by the `retryPattern` attribute instead of the usual combination of the `tentatives` and `retryInterval` attributes. If this attribute takes on a value other than `none`, it is interpreted as a list of retry intervals. The number of intervals in the string is the number of retries performed and thus the total maximum number of delivery attempts is 1 + the number of intervals in the string. The form of the string is “10 sec; 30 sec; 5 min; 1 hour” – i.e. a semi-colon separated string of time expressions. Please refer to the DDS user guide for details on the format.

The retry pattern is by default not used for any media type (the value is `none` in the default media configuration file). From AXT The pattern could be provided for a delivery as below:

```
<filter class="dk.bording.axt.client.docdeliver.DocDeliverOutlet" output="input">
  <param name="media" value="ftp"/>
  <param name="mask" value="file%s.txt"/>
  <param name="ftpserver" value="acme.main.server"/>
  <param name="retryinterval" value="30;60;600"/>
  ...
  ...
  ...
```

The above usage specifies that the job should be attempted redelivered after 30, 60 and 600 seconds. Note that this may produce four delivery attempts in total.

9.1.2 Controlling checkpoint generation on delivery success and failure from AXT

DDS now supports the generation of trace checkpoints to the L&T system after successful delivery or delivery failure. Two default media properties control this:

Parameter	Type	Description	Default
checkpointTextOnError	Optional	A string that will be used to generate a Trace checkpoint in case the delivery fails	<i>None</i>
checkpointTextOnSuccess	Optional	A string that will be used to generate a Trace checkpoint in case the delivery succeeds	<i>None</i>

By default no checkpoint is generated but if the parameters are specified for example from AXT configuration they will be used.

9.1.2.1 Using variables in checkpoints

The checkpoint text may include variable references to the following type of information

- other job properties
- document properties
- trace properties

The trace and document properties are resolved when the checkpoint is persisted in the log server. If the referred to properties are not present in the database they will not be resolved.

Job properties are resolved by DDS at the time the job either succeeds or fails and the checkpoint text is submitted to the log server.

Properties to be substituted are entered in the checkpoint text string using the following notation:

```
[<type>:<name>]
```

Where type is one of `job`, `traceprop`, `docprop` and name is a valid property name of the given type. I.e. name could be `media` or `description` if the type was `job` since jobs do have these properties.

Valid trace and document properties depend on what properties have been attached by configuration.

The following example illustrates how a checkpoint text might be defined:

```
<filter class="dk.bording.axt.client.docdeliver.DocDeliverOutlet" output="input">
  <param name="checkpointTextOnError" value="The file ([docprop:ref]) sent by [docprop:sender]
  with did=[job:_did] for media [job:media] on trace-type [traceprop:type] in group
  [traceprop:group] could not be delivered!"/>
  ...
  ...
```

Checkpoint generation on delivery attempts

If DDS is configured to retry delivery a number of times, only the last failure will result in an error in the trace system. However, DDS can generate checkpoints to the L&T system after each failed attempt to document the delivery attempt. These checkpoints are by default generated with a severity level of “info”:

/ T-90				
✓	26-06-2007 10:19:33:056	i	Delivery attempt 1 of 3 failed. Next retry at Tue Jun 26 10:19:43 CEST 2007	T-90/bbo-pc2/adk2.2/JVM-2/dds/core(1)
✓	26-06-2007 10:19:49:841	i	Delivery attempt 2 of 3 failed. Next retry at Tue Jun 26 10:20:19 CEST 2007	T-90/bbo-pc2/adk2.2/JVM-2/dds/core(1)
✓	26-06-2007 10:20:35:602	i	Delivery attempt 3 of 3 failed. Next retry at Tue Jun 26 10:20:45 CEST 2007	T-90/bbo-pc2/adk2.2/JVM-2/dds/core(1)

The generation of such checkpoints and their severity level is controlled by the following two DDS properties (conf/properties/docdeliver.properties file):

docdeliver.checkpointsonretries	Optional	States if checkpoints should be made each time the DDS is unable to deliver a file but will retry later. The checkpoint will state the time of the subsequent retry	True
docdeliver.checkpointsonretrieslevel	Optional	The level of the checkpoint generated in connection with retries (see above). Possible values are "info", "warn", "error".	Info

10 API news (for developers)

10.1 Generating AXTRetryableProcessingExceptions from custom AXT filters

AXT transformations can have their processing retried under the following conditions:

- 1) The transformation must be initiated by the DDS server (this will be the case for FTP, FILE and MailInlet input) since it is the DDS server that does the retrying
- 2) A failing filter in the *main* transformation throws a AXTRetryableProcessingException

The AXTRetryableProcessingException can be created and throw as show below

```
throw new AXTRetryableProcessingException(new AXTProcessException("throw this nested one.."));
```

I.e. wrapping an existing exception but also taking a string parameter or both. Typical usage would be in places where the error is due to temporary communication problems.

10.2 Communicating precise error context from custom AXT filters

The new AXT Error view GUI enables viewing of the error context from custom AXT filters. Since all AXT filters throws an AXTEException in case of failure, this is abstract class has been enriched with an `ErrorList` query method. Which all subclasses must implement in order to make use of the added error view GUI. A special `ErrorData` class has been created which contains the data viewable in the GUI, such as exact error location, Error snippet, error message etc.

The following example creates an `ErrorData` object from a `Transformer` exception, setting the line and column number:

```
public class MyAXTEException extends AXTEException {
```

```
private List errors;
...
...
public MyAXTException(Throwable t) {
    super(t);
    errors = new ArrayList();
    if (t instanceof TransformerException) {
        errors.add(getErrorData(t);
    }
}

public ErrorData getErrorData(TransformerException exception) {
    ErrorData errordata = new ErrorData(exception);

    SourceLocator sl = exception.getLocator();
    if (sl!=null) {
        errordata.setLineNo(new Long(sl.getLineNumber()));
        errordata.setColumnNo(new Long(sl.getColumnNumber()));
    }
    Return errordata;
}

public List getErrorDataList()
{
    return errorList;
}
}
```

AXT will automatically try to extract the source context from the supplied line and column number, or byte position, if supplied by the `ErrorData` object. See the JavaDoc for more details on usage.

10.3 Adding function support in L&T

Please refer to the Log & Trace programmer's guide where examples and API are presented.

10.4 Accessing transformer instance and document ID in an XSL context

In order to support the generation of document events such as receive, transform, send, attachment of information or checkpoints in the context of XSL and in Java-classes called from XSL, AXT 3.0 provides access to the current TID (transformer instance and event factory) as well as the id of the current document (For introduction to the Log & Trace API please consult the API-guide).

The current transformer instance is passed to XSL scripts as the parameter `_TID`. The document id of the input dataset is passed by the name `_did`. The `_did` parameter is a string. The `_TID` parameter is an instance of the `dk.bording.inside.trace.TID` class. The instance is for the invoked `XSLProcess` filter.

The intended usage is to pass this parameter to the XSL-script and from there to any Java-class that wish to produce trace-events. At present no specific event-extension is provided for XSL. We shall illustrate how the transformer and document id could be used from Java with a simple example.

In our example an XSL-script receives XML-input that should be processed. Each element in the XML-file is processed and we wish to produce checkpoints after processing each element and when we have processed all elements successfully. We use an example Java-class (`TracingJavaXSL`) for creating the checkpoints. The Java-class is initiated and called from XSL.

The input file has the format shown below.

```
<?xml version="1.0" ?>
<elements>
  <element qty="12" price="200" id="EA12344332"/>
```

```

<element qty="11" price="330" id="PTP22344332"/>
<element qty="11" price="330" id="IR22344332"/>
</elements>

```

The XSL-script looks as follows.

```

21.     <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
22.         xmlns:TID="xalan://dk.bording.inside.trace.TID"
23.         xmlns:CPCls="xalan://dk.bording.axt.examples.TracingJavaXSL"
24.     >
25.     <!-- we may expect a tranformer factory (_TID) and a document id (_did) that are set by
the AXT core
26.     -->
27.     <xsl:param name="_TID"/>
28.     <xsl:param name="_did"/>
29.
30.
31.     <xsl:template match="/elements">
32.         <test>
33.             <!-- create a new instance of our own java class -->
34.             <xsl:variable name="cpCls" select="CPCls:new()"/>
35.             <!-- set the transformer and the doc id on our class -->
36.             <xsl:variable name="dummy" select="CPCls:init($cpCls, $_TID, $_did)"/>
37.             <!-- for each element we calculate the total (to illude processing) -->
38.             <xsl:for-each select="element">
39.                 <element>
40.                     <xsl:attribute name="id">
41.                         <xsl:value-of select="@id"/>
42.                     </xsl:attribute>
43.                     <xsl:attribute name="qty">
44.                         <xsl:value-of select="@qty"/>
45.                     </xsl:attribute>
46.                     <xsl:attribute name="price">
47.                         <xsl:value-of select="@price"/>
48.                     </xsl:attribute>
49.                     <total>
50.                         <xsl:value-of select="@qty*@price"/>
51.                     </total>
52.                 </element>
53.
54.                 <!-- have our class make a new checkpoint (with it's own sub-transformer id) -->
55.                 <xsl:variable name="dummy" select="CPCls:doCheckPoint($cpCls, concat('Processed
element with id ',@id))"/>
56.             </xsl:for-each>

```

The passing of the parameters is defined in the lines 7-8. Besides a namespace definition for the transformer class is made in line 3. This is necessary if we wish to invoke functions directly on the class in the XSL-script.

The construct from 18-36 is a loop selecting all element tags of the input. Immediately before selecting these elements we create an instance of the Java-class that should create checkpoints. We pass the `_TID` and `_did` values to the class in line 16 (calling an `init()` method). The `init` method creates a sub-transformer instance (representing itself) and stores the `_did` for later use. The `init` method is:

```

public void init(TID tid, String did) {
    this.did = did;
    this.subTid = tid.createChildTID("TracingJavaXSL");
}

```

The line 19-32 simply generates XML-output which has the original shape except we add a total attribute (`total`). This part is simply illustrating some processing to give sense to our example.

In line 35 we invoke the `doCheckpoint()` method of our Java-class.

```
public void doCheckpoint(String label) {
    CheckPointEvent cpEvent = subTid.getCheckPointEvent (did);
    cpEvent.setLabel(label);
    cpEvent.setLevel("INFO");
    cpEvent.commit();
}
```

In this code we generate a checkpoint event having the label passed from the XSL-script. The label is a string consisting of the text “Processed element with id” followed by the extracted id of the element.

The complete Java-helper class is shown below.




```
5 public class TracingJavaXSL {
6
7     private String did;
8     private TID subTid;
9
10    public void init(TID tid, String did) {
11        this.did = did;
12        this.subTid = tid.createChildTID("TracingJavaXSL");
13    }
14
15    public void doCheckpoint(String label) {
16        CheckPointEvent cpEvent = subTid.getCheckPointEvent (did);
17        cpEvent.setLabel(label);
18        cpEvent.setLevel("INFO");
19        cpEvent.commit();
20    }
21 }
```

The result of the processing and checkpoint generation is reflected in the L&T-trace of the transformation. The events are shown in “flat” mode below.









Trace Results				
Trace List		Trace Events		
/ T-334				
Status	Time	Event	Description	
✓	27-11-2006 11:45:14:523		Received <u>d1</u> From invoking client(browser?)	T-334/bbo-laptc
✓	27-11-2006 11:45:14:523		Sent <u>d1</u> To <u>main(4)/XSLProcess(4)</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:523		Received <u>d1</u> From <u>docentry("trace in xsl" 4)</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:575		Processed element with id <u>EAI2344332</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:575		Processed element with id <u>PTP22344332</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:575		Processed element with id <u>IR22344332</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:575		Done checkpoints from JAVA/XSL!	T-334/bbo-laptc
✓	27-11-2006 11:45:14:585		Transformed <u>d1</u> To <u>d2</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:595		Sent <u>d2</u> To <u>docentry("trace in xsl" 4)</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:595		Received <u>d2</u> From <u>main(4)/XSLProcess(4)</u>	T-334/bbo-laptc
✓	27-11-2006 11:45:14:595		Sent <u>d2</u> To ?	T-334/bbo-laptc

In hierarchical view mode the XSL/Java-class has now their own sub-transformer (TracingJavaXSL) below the XSLProcess filter. The use of a sub-transformer may or may not be appropriate. This depends on the processing semantics. In this example we did it to illustrate the possibility but the semantics of the processing would probably call for no sub-transformer.

/T-334 /bbo-laptop/adk2.1/JVM-1/axt /core(1) /docentry("trace in xsl" | 4) /main(4) /XSLProcess(4)

Status	Time	Event	
✓	27-11-2006 11:45:14:523		Received d1 From  docentry("trace in xsl" 4)
✓	27-11-2006 11:45:14:575 to 27-11-2006 11:45:14:575		 TracingJavaXSL(2)
✓	27-11-2006 11:45:14:585		Transformed d1 To d2
✓	27-11-2006 11:45:14:595		Sent d2 To  docentry("trace in xsl" 4)

/T-334 /bbo-laptop/adk2.1/JVM-1/axt /core(1) /docentry("trace in xsl" | 4) /main(4) /XSLProcess(4) /TracingJavaXSL(2)

Status	Time	Event	
✓	27-11-2006 11:45:14:575		 <i>Processed element with id EA12344332</i>
✓	27-11-2006 11:45:14:575		 <i>Processed element with id PTP22344332</i>
✓	27-11-2006 11:45:14:575		 <i>Processed element with id IR22344332</i>
✓	27-11-2006 11:45:14:575		 <i>Done checkpoints from JAVA/XSL!</i>