

Matti Kaltiainen

Exception handling in OpenText StreamServe

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

9 February 2015

Author Title	Matti Kaltiainen Exception handling in OpenText StreamServe
Number of Pages Date	38 pages 9 February 2015
Degree	Tietotekniikan insinööri
Degree Programme	Tietotekniikka
Specialisation option	Ohjelmistotekniikka
Instructor	Yliopettaja Jaana Holvikivi
<p>Insinööriyön tavoitteena oli tutkia erilaisia virheidenhallintamenetelmiä OpenText StreamServe ohjelmistossa. OpenText StreamServe on tulostuksenhallintaohjelmisto, jota käytetään pääasiassa tulostuksen automatisointiin ja dokumenttien esillepanoon. Virheidenhallinta voidaan toteuttaa käyttämällä OpenText StreamServen omia sisäänrakennettuja työkaluja tai käyttämällä ohjelmiston ulkopuolisia työkaluja. Tämän vuoksi virheidenhallinnan toteuttamiseksi on olemassa runsaasti erilaisia vaihtoehtoja.</p> <p>Insinööriyötä varten toteutettiin OpenText StreamServe tulostuksenhallintaratkaisu, jonka avulla erilaisia virheidenhallintamenetelmiä pystyttiin testaamaan ja arvioimaan. Tämän jälkeen tulostuksenhallintaratkaisuun lisättiin useita erilaisia virheidenhallintaratkaisuja. Lopuksi vielä testattiin kaikkien toteutettujen virheidenhallintaratkaisujen toimivuus.</p> <p>Tästä insinööriyöstä voi olla hyötyä muille OpenText StreamServe kehitystyötä tekeville virheidenhallintaratkaisua suunnitellussa. Insinööriyötä varten kehitetty virheidenhallintaratkaisu soveltuu ainoastaan havainnollistamiseen ja testaamiseen. Se ei välttämättä toimisi kokonaisuutena tuotantoympäristössä, jossa muodostetaan suuria määriä asiakirjoja. Insinööriyötä voi kuitenkin käyttää apuna virheidenhallinnan suunnitteluun, sillä virheidenhallintaratkaisussa käytettyjä yksittäisiä virheidenhallintamenetelmiä voidaan käyttää myös tuotantoympäristössä.</p>	
Keywords	OpenText StreamServe, Exception handling

Author Title	Matti Kaltiainen Exception handling in OpenText StreamServe
Number of Pages Date	38 pages 9 February 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Jaana Holvikivi, Principal Lecturer
<p>The objective of this thesis was to study different methods OpenText StreamServe software provides for exception handling. OpenText StreamServe is output management software that is mainly used for automated document creation and presentation. Exception handling may be implemented in OpenText StreamServe using built-in tools included in the software or using external tools. This means there are plenty of ways to implement exception handling.</p> <p>A OpenText StreamServe printing solution was created for the project to enable the testing and evaluation of different exception handling methods. After creating the printing solution, various ways of exception handling were implemented to the solution. At the end, the functionalities of the implemented exception handling methods were tested.</p> <p>This thesis may be used as a reference by other OpenText StreamServe developers to help design and develop exception handling solutions for their output management projects. The solution created for the project is only for demonstrating and testing purposes and would not work in high volume production environment. However, it may be used for planning as the individual methods used in the solution could be implemented also in production environment.</p>	
Keywords	OpenText Streamserve, Exception handling

Contents

Abbreviations

1	Introduction	1
2	Exception handling	2
2.1	What is an exception	2
2.2	What is exception handling	2
2.3	Why exception handling is important	3
2.4	Exception handling methods	3
3	Introduction to OpenText StreamServe	5
3.1	Main components	5
3.1.1	Design Center	6
3.1.2	Control Center	8
3.1.3	Framework and StreamServer	9
4	Exception handling in OpenText StreamServe	10
4.1	Validating input XMLs	11
4.2	Error Message	12
4.3	Status Messenger	14
4.4	Scripting	15
4.4.1	Built-in script functions	15
4.4.2	Errors to Status Messenger	16
4.4.3	External programs	16
4.4.4	ODBC functions	17
5	Implementation	18
5.1	Installed software	18
5.1.1	OpenText StreamServe	18
5.1.2	Microsoft SQL Server Express 2012	18
5.2	OpenText StreamServe implementation	19
5.2.1	Creating StreamServe_StudentTrancript application	20
5.2.2	XML_split message	20
5.2.3	Student_transcript message	21
5.2.4	Error Message	22

5.3	Exception handling	22
5.3.1	Configuring XML validation	23
5.3.2	Configuring Error Message	24
5.3.3	Creating StreamServe_StatusMessenger application	25
5.3.4	Configuring database connection	27
5.4	Deploying the StreamServer applications	28
6	Testing the implementation	30
6.1	Output files generated by the application	30
6.2	Testing the application with valid XML	33
6.3	Testing XMLin validation	34
6.4	Testing Error Message	35
6.5	Testing StatusMessenger	35
6.6	Writing errors to database	36
7	Conclusion	37
	References	38

Abbreviations

AFP	Advanced Function Presentation
CCM	Customer Communication Management
DTD	Document Type Definition describes the structure of an XML file.
ERP	Enterprise Resource Planning
ODBC	Open Database Connection
PDF	Portable Document Format
SCF	StreamServe Service Component Framework
XML	Extensible Markup Language
XPath	XML Path language
XSD	XML Schema

1 Introduction

OpenText StreamServe is Customer Communication Management software which is most often used for automated document creation. StreamServe may be used with almost any Enterprise Resource Planning system as OpenText StreamServe is independent software that can be installed on various versions of Windows and Linux platforms. In most cases OpenText StreamServe may be linked to an existing business system with only a few changes.

Creation of document layouts and configurations is effortless with a user friendly graphical interface. Efficient document creation together with document bundling options and a large scale of output devices reduce printing costs. StreamServer applications are also high in performance, which means they are capable of handling large amounts of data over a short time period. These are some of the reasons why OpenText StreamServe is used widely in large companies in Finland and around the globe. [1.]

As all of the StreamServe solutions are made from scratch, they do not have automatically any exception handling included. The purpose of this thesis is to study different options available in the OpenText StreamServe product for handling exceptions. This thesis will present a creation of a StreamServe solution, which will include a comprehensive exception handling. For better understanding of the OpenText StreamServe product, the core components will also be briefly introduced. At the end, all the implemented exception handling methods will be tested and evaluated.

2 Exception handling

Exception handling is part of program code that alters the program flow in situations where something exceptional happens. The purpose of exception handling is to enhance the quality of a program by minimizing error situations and giving end users information about the exceptional situations. This chapter will describe more detail what exception handling is, where and how it should be used and how it differs from error handling.

2.1 What is an exception

Various problems can occur during program execution. For example, a user can enter invalid input data, file content may be corrupted and missing or network connection could have been lost. These errors can be called exceptions. [2.]

Situations where problems or exceptions occur may be predicted and therefore handled by exception handling. For example, in a case of lost internet connection, instead of crashing, a web browser will display a message about missing internet connection.

2.2 What is exception handling

The concept of exception and error handling is often confused and misunderstood. The meaning of exception and error handling may also differ between different programming languages. Even though it is almost impossible to give one exact explanation to exception handling, it is good to clarify the difference between these two concepts. Exception handling is briefly described below:

Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point. [3.]

The definition above indicates that the code execution can be continued after a specified exception by changing the normal flow of the code. In exception handling, these exceptional conditions are predicted to be possible and handled in the code. An error would be something like a failure in programming logic. This could cause the execution of the code

to terminate uncontrollably. Similar kind of errors could occur with hardware failures as well. Even though some of these could be predicted, they cannot be handled in the manner of exception handling. One of the main differences between exception and error handling is the predictability of the problem. Exceptions are easier to predict and therefore easier to handle than errors.

As mentioned earlier, the meaning of exception handling differs between different programming languages. As a specific definition for exception handling is not available for OpenText StreamServe, the description above will be used in this thesis. This means that this thesis will focus on situations that may be predicted and handled in a controllable way.

2.3 Why exception handling is important

One of the main purposes of exception handling is to enhance the quality of a program and user experience. A well-designed quality program runs smoothly and is easy to use. Exception handling can enhance the quality of a program by minimizing crashes and providing robust execution of code as described below:

A well-designed set of error handling code blocks can make a program more robust and less prone to crashing because the application handles such errors [4].

A simple example of exception handling could be a website that asks a user to submit username and password. If the username and password do not match, the user will be informed and suggested to try again. Alternative to this behavior could be an uninformative error message that the webpage cannot be viewed or another error situation. Exception handling can enhance the usability of a program by providing information to end users.

2.4 Exception handling methods

The methods of exception handling differ between programs as the programs may have different needs and functionalities. Even though there are various ways to handle exceptions, there is something common in most of them. The one common thing is that the

situations where exceptions may happen need to be recognized. Once the conditions and situations are recognized, the exceptions may be handled.

In programming languages such as Java and PHP, a concept called “try and catch” can be used to handle the exceptions. A brief description of this concept can be found below:

The **try** statement lets you test a block of code for errors.
The **catch** statement lets you handle the error.
The **throw** statement lets you create custom errors. [5.]

The exceptions should be caught as early as possible. This way the exception or error can be handled in the early phase and will not be able to do us much harm.

3 Introduction to OpenText StreamServe

OpenText StreamServe is Customer Communication Management software which offers tools for various areas of business. It is mostly used for creating business documents with different layouts, output formats and channels. These customized documents could be for example invoices, insurance policies, annual reports or advertisements. Other usages could be converting data from one format to another or forwarding data to an ODBC-compliant database. Even though StreamServe is designed for creating and storing business documents, it is easy to configure to handle data various other ways as well.

One of the main advantages of OpenText StreamServe is to be able to handle data from many different sources in various formats. For example, input data can be delivered to StreamServe using a directory-, http- or email-connector. XML is one of the most popular formats for input data as the fields are easy to identify and extract by using XML Path Language (XPath). Other supported input formats include for example, delimiter-separated, page based and preformatted data. There are also multiple output formats and channels available. As OpenText StreamServe is an independent software application, it can be used with almost any business system a company might have. StreamServer may be installed on various different Windows and Linux operating systems. However, the design tools can be installed only on the Windows operating system.

The other advantages of OpenText StreamServe are effortless customization of documents, high performance with large amounts of documents, adaptation to different needs of business and automatized document creation. Highly automatized document creation with OpenText StreamServe can also reduce costs of document creation, printing and management.

3.1 Main components

OpenText StreamServe software consists of various different components. This thesis will introduce only the three main components that are needed to design and run StreamServer application. As this is not the main topic of the thesis, the three components will be described only briefly.

3.1.1 Design Center

Design Center is a component for designing StreamServe projects. Once the project configuration is ready, the project needs to be exported and deployed to StreamServer application. Everything from input connectors to fonts and runtime settings are configured in Design Center.

Design Center consists of four main components:

- Platform
- Message
- Runtime
- Resource Set

Platform is where the input and output connectors are created and configured as shown in figure 1. The input connector defines how to connect with the backend system and the output connector defines how to connect with the output device.

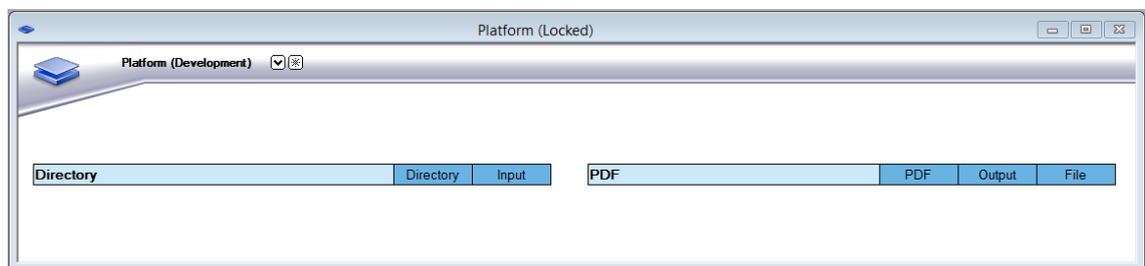


Figure 1. Platform with a directory input connector on the left side and a PDF output connector on the right side. Screenshot [6].

A message consists of two components, an event and a process. The event defines how to collect input data. There are five different types of events that can be used for different types of input data [7.]:

- Page based
- Field based
- Record based
- XML based

- Preformatted (AFP, PDF).

There are eight different process types that can be used for different output types as shown in figure 2. The process determines how to process the data that was collected in the event.



Figure 2. Available events are listed on the left side and processes on the right side [7].

There can be several events and processes in the same message. The events have to be similar but the processes can be different from each other. This allows processing of the same input data with multiple different processing tools. For example StoryTeller may be used to create page formatted output and XMLOUT to create an XML-file with matching data. As can be seen in figure 3, it is required to have at least one event in each message to allow the input data to be collected.

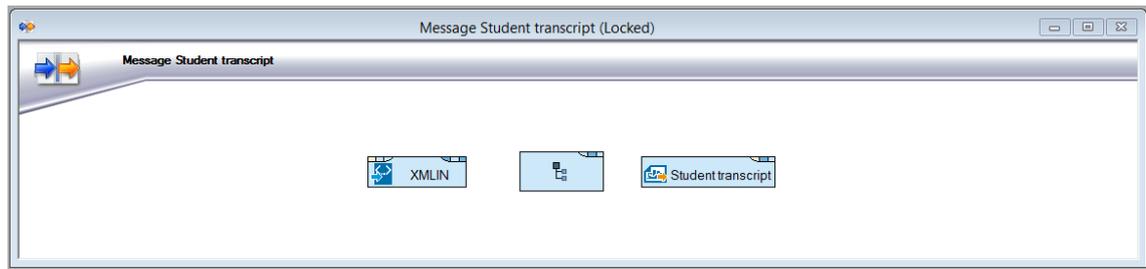


Figure 3. Message with XMLIN event and StoryTeller process. Screenshot [6].

Runtime connects the platform and messages together as shown in the figure 4. It defines the connectors that are used for specific events and processes. Messages are independent from the connectors, which allow using them more dynamically. For instance, the output connector can be chosen dynamically by assigning it a variable. The same output can also be sent to multiple output channels like printer, PDF, Word or AFP file.

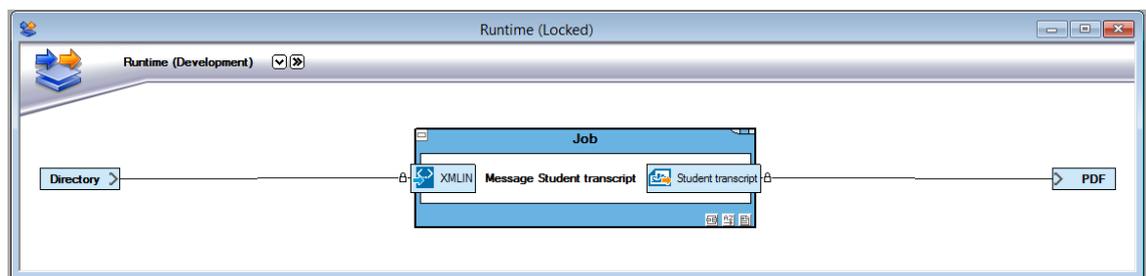


Figure 4. Runtime configuration. Screenshot [6].

A resource set is a place to store resources needed in a StreamServe project. These resources can be for example sample files, language tables, images and fonts. A resource set can be shared with other StreamServe projects. This also helps administration as multiple projects can use the same resources.

Once all the configurations have been done in Design Center, the project can be exported and deployed in Control Center.

3.1.2 Control Center

Control Center is an administration tool for managing StreamServe applications. The applications are divided into different sites and application domains as shown in figure 5.

One Control Center can be connected to multiple sites, which can have multiple application domains. Each site represents one environment, for example, development, test or production. Inside one environment, there can be multiple servers by creating multiple application domains to Control Center.

In smaller solutions, it is possible to have only one site that includes all the different environments.

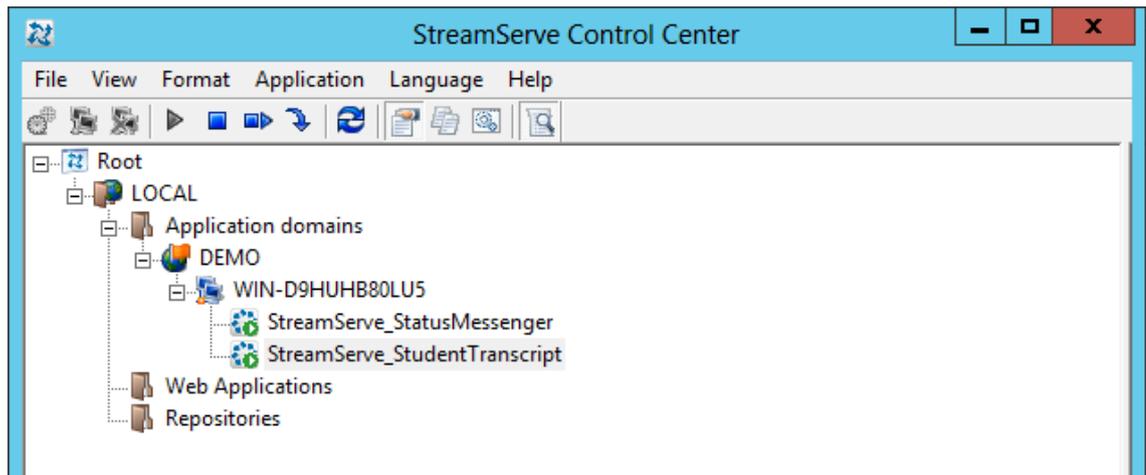


Figure 5. Control Center which has one Application domain "DEMO" including two applications. Screenshot [6].

3.1.3 Framework and StreamServer

StreamServe Service Component Framework (SCF) provides services for StreamServer applications. These services are required to run StreamServer applications. [8.]

4 Exception handling in OpenText StreamServe

Well designed and implemented exception handling enhances the quality of a printing solution. For example, it can ensure that all the documents are created and sent to customers or it may prevent sending incorrect documents to customers. Sending incorrect documents to customers is not only expensive but it also affects a company's image and reliability.

In smaller environments where document creation volume is fairly low, exception handling is often totally left out. The reason for this could be that the applications are quite simple and reliable and there is a feeling that exception handling is not needed. This may be true, as errors are easier to locate and handle in smaller environments. Documents may even be manually revised before being sent out to customers. Exception handling would not automatically provide significant enhancement to the solution in such environments. Moreover, the consequences of errors may not be as significant in smaller environments.

Exception handling is more beneficial in larger environments where thousands of documents are created and sent out to customers on a daily basis. Administrating a large environment may also be quite laborious. Revising all the documents is impossible and searching reasons for errors would make the administrating even more laborious. Exception handling can reduce the administrating work load by identifying error situations. For example, invalid data and documents are easier to locate and identify which helps solving the problems more efficiently. In a long run, this should reduce costs as there is less administrating work.

Before going to the methods of exception handling, it is good to take a look at the possible situations where errors may happen. Most often StreamServe applications are designed to handle large amounts of data. Basically this means a large amount of input and output operations which may itself lead into error situations. A large amount of operations also requires a good performance from the computers used to handle the data. Poor performance may hinder the operations and lead into error situations. The input data may also be invalid or corrupted and resources needed to create documents may not be available. The situations mentioned above seem to be the most common reasons for errors in

StreamServe. Invalid configurations in StreamServer applications are usually caught already during the testing phase. However, all the variations cannot always be tested. This is where exception handling is useful.

There are multiple methods for handling exceptions at different job execution phases. The StreamServer job execution can be divided into three phases: input phase, formatting phase and output phase [8]. During the input phase, input data is recognized and all the needed information is collected from the data. The formatting phase will format the data that was collected during the input phase. Formatted documents or data will be forwarded to different output channels during the output phase.

Invalid or missing input data can be caught during the input phase. Formatting errors, caused for example by missing resources (images, texts), can be caught during the formatting phase. Missing or invalid data may be also caught during the formatting phase, even though it could be better to catch them earlier during the input phase. Network connection problems or insufficient permission to perform operations can cause errors during the output phase.

There are three main ways to configure exception handling in Streamserve. They are input data validation, scripting and Status Messenger. Input data may be validated using XML validation. Scripting allows a large selection of options that can be used to implement exception handling. This includes StreamServe's internal scripting functions and tools as well as using external tools to report errors. Status Messenger is a configurable application that is mainly designed for collecting notifications and errors. Those errors can be forwarded to an application's administrator before the documents are delivered forward.

This chapter will explain methods that OpenText StreamServe offers for handling exceptions and errors.

4.1 Validating input XMLs

Incoming XML-files can be validated before they will be processed by StreamServer. By validating the incoming XML-files, errors in the data can be caught before they reach the formatting phase. There are five different validation levels in the XMLin event:

- DTD + XSD strict validation
- DTD + XSD lax validation
- DTD validation
- Preprocess
- No validation / preprocessing

[10.]

Once XML validation fails, the error can be forwarded to Error Message or Status Messenger for further processing. Error Message and Status Messenger will be described in more detail later on in this chapter.

As the XMLin validation level gets stricter, the performance of a StreamServer application will slow down [10]. This has to be recognized when designing StreamServer applications.

4.2 Error Message

Error Message is a configurable message inside a Design Center project. It can be used in error situations to alter the normal flow of the code. As can be seen in figure 6, if the XML validation fails, the job will be forwarded to the Error Message instead of the primary message. If an error has been caught during the job execution, the Error Message can be triggered also from a script inside the primary message using a CancelJob script function [11].

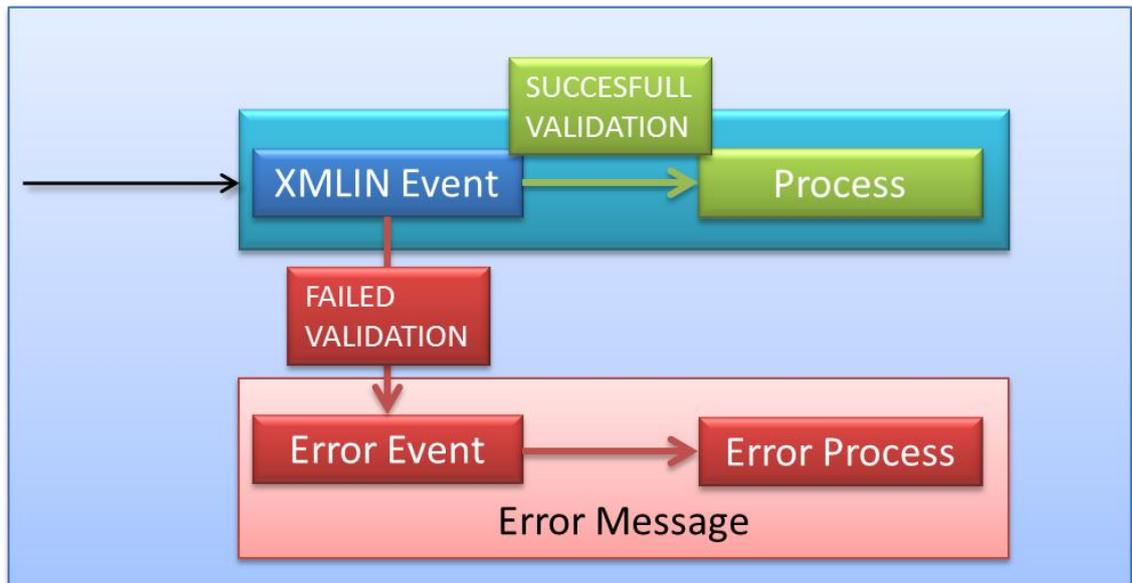


Figure 6. This figure shows how the job execution will be changed if the XML validation fails.

Error Message can be configured to send information to desired output channels, for example, email, SMS or file. This makes administration of an application less effortless.

As can be seen in figure 7, Error Message consists of a preconfigured MessageIN event [12], which gets the data from an internal connector and a process, which can be any type that suits for the desired output.

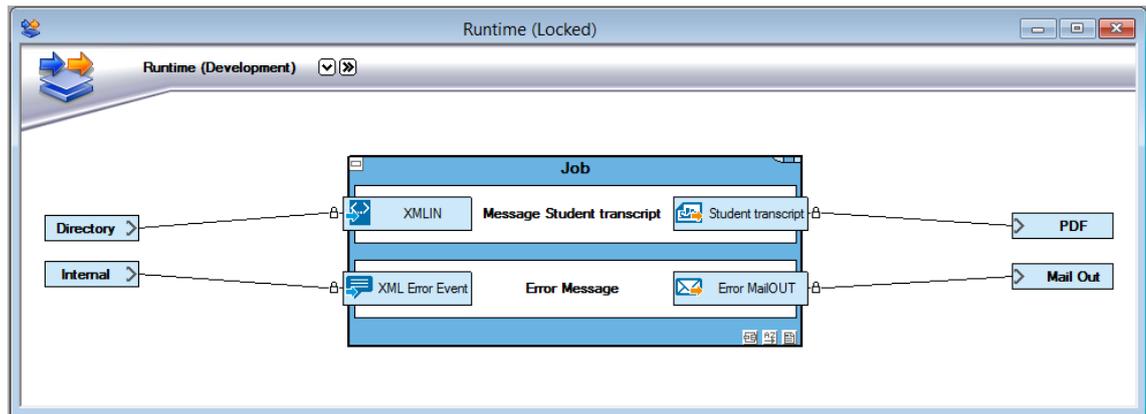


Figure 7. Runtime configuration where Error Message is connected to the Internal connector. Screenshot [6].

4.3 Status Messenger

Status Messenger is an independent StreamServer application that collects information about errors in different applications. The purpose of Status Messenger is to collect errors from other applications and forward them to desired output channels such as e-mail, SMS or Windows Event Log [13]. One of the main advantages of Status Messenger is that it enables centralized error reporting even in larger environments with multiple applications. In such configuration, Status Messenger would be its own StreamServer application that collects notifications from other StreamServe applications. In smaller environments Status Messenger may be included into one of the main applications.

Status Messenger is configured in Design Center just like any other StreamServe application. It can be configured to catch all the errors in a job or only the ones that are needed. To use Status Messenger, StreamServe applications need to be configured to store notifications to a repository where Status Messenger can collect them from. The Status Messenger procedure can be seen in figure 8. As Status Messenger may decrease the overall performance, it is recommended to collect only notifications from components that are needed.

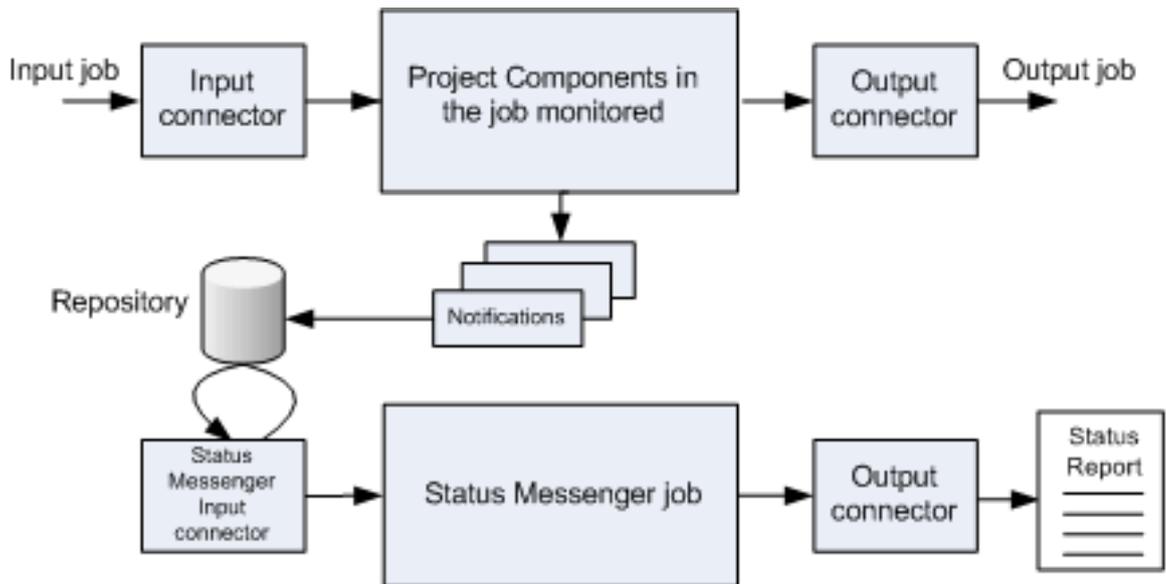


Figure 8. Status Messenger procedure [13].

The configuration of Status Messenger input connector includes two modes, basic mode and advanced mode. The notifications to be collected are specified in these configurations. The basic mode is a preconfigured set up where the user can choose from a list of properties. The advanced mode enables wider modification, which includes choosing from different log identifications and external job identifications. More detailed explanation of the configuration will be included in the part of the thesis that describes the implementation.

4.4 Scripting

OpenText StreamServe has its own scripting language that can be used in various places of a Design Center configuration. Scripting can also be used for exception and error handling. There is also a large amount of built-in script functions that are able to return values that can be used to monitor applications. Script functions may also be used to alter the normal flow of the code in error situations. Most of the error situations may be caught using scripting together with scripting functions.

4.4.1 Built-in script functions

Built-in script functions can be used to monitor the status of the current StreamServer job. They can also be used to alter the job execution or even terminate the running

StreamServer application. This chapter will introduce some of the built-in functions that can be used as part of the exception handling. A full listing of built-in scripting functions can be found from OpenText StreamServe online help [14].

The GetJobStatus function returns the status of the current job [15]. This function can be used to monitor the status of the job.

The CancelJob function can be used to call another “error event” if something unwanted occurs during a job [11].

The IoErrText function returns an error message from StreamServe’s own IO functions such as FileOpen, FileWrite, FileDelete and FileClose [16]. These functions can be used for example to delete or edit files on a disk. The IoErrText function enables better monitoring of the file operations.

The Terminate function terminates the running application after the current job has been finished [17].

The SetJobFailed function sets the current job status to “failed”.

4.4.2 Errors to Status Messenger

Errors can be delivered to Status Messenger from different StreamServe applications. This can be done by setting an external job identifier using the the SetExtJobId function in a script. The External job identifier needs to be configured in the Status Messenger project. The identifier can be any string with acceptable characters.

4.4.3 External programs

External programs can be called from StreamServe scripts using the Execute function. The function can be called from any scripting phase without interrupting the job execution. This could be used for example to deliver error messages to external error reporting programs. This allows the usage of almost any application that may be run from command prompt.

4.4.4 ODBC functions

OpenText StreamServe has built-in ODBC functions to create a database connection and to execute SQL statements. This may be used to write errors directly to a database instead of writing them to a file for example. Establishing a database connection and entering data to a database may hinder the performance of the application. This has to be noticed when designing the solution.

5 Implementation

This chapter will explain the core components and software which was installed and used in this project. This chapter also explains also the OpenText StreamServe solution that was created for the project.

5.1 Installed software

The operating system used in the project was a Windows Server 2012 virtual machine. The virtual machine was created with VMware Workstation version 9.0.0. To run applications, StreamServer requires a database that stores information about servers, applications and application domains. For this purpose, Microsoft SQL Server Express 2012 was also installed.

5.1.1 OpenText StreamServe

OpenText StreamServe 5.6.1 Build 136 was installed on the Window Server 2012 virtual machine with the following OpenText StreamServe components:

- Framework and StreamServer (x64)
- Control Center
- Design Center

5.1.2 Microsoft SQL Server Express 2012

Microsoft SQL Server Express 2012 with Tools was installed on the Windows Server 2012 virtual machine using SQL Server Installation Center. Three repositories were created to the SQL Server as shown in figure 9. Two of the repositories were created to the SQL Server using Control Center's Database Creation wizard:

- StreamServe Enterprise Repository: strsSER
- StreamServe Runtime Repository: StrsData

A third repository, StrsError, was created for error messages using Microsoft SQL Server Management Studio. This repository stores data about errors occurring in the application. The ODBC functions were used from StreamServer application to connect the database and to insert and update values. ODBC data source was also configured to allow the ODBC functions to connect the SQL Server database.

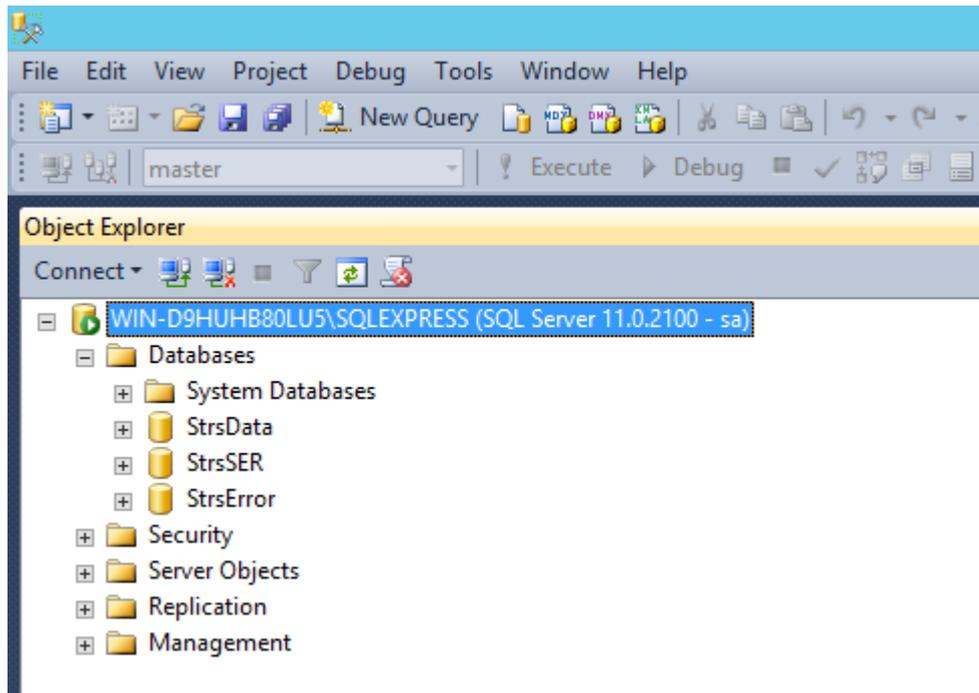


Figure 9. SQL Server databases. Screenshot [18].

5.2 OpenText StreamServe implementation

This chapter will describe the creation of the OpenText StreamServe solution which creates student transcripts from XML-input data. The StreamServer application will read XML-files from a directory and create PDF-files. Depending on the content of the XML input file, the output PDF-files are created to a directory or sent directly to a recipient via email. Different methods of exception handling will be added to the solution to test and demonstrate the possible error situations.

This OpenText StreamServe solution will have two applications. The main application is for collecting input data and creating and sending PDF documents from the collected

data. The other application is Status Messenger which follows the main application and collects the notifications/errors occurred.

5.2.1 Creating StreamServe_StudentTranscript application

As can be seen in figure 10, the main application has three messages: XML_split, Student_transcript and Error Message. XML_split is used for splitting XML-files with multiple documents into single documents. Student_transcript creates PDF-documents and forwards them to output connectors. Error Message is triggered with XMLIN validation errors and canceled jobs called from scripts.

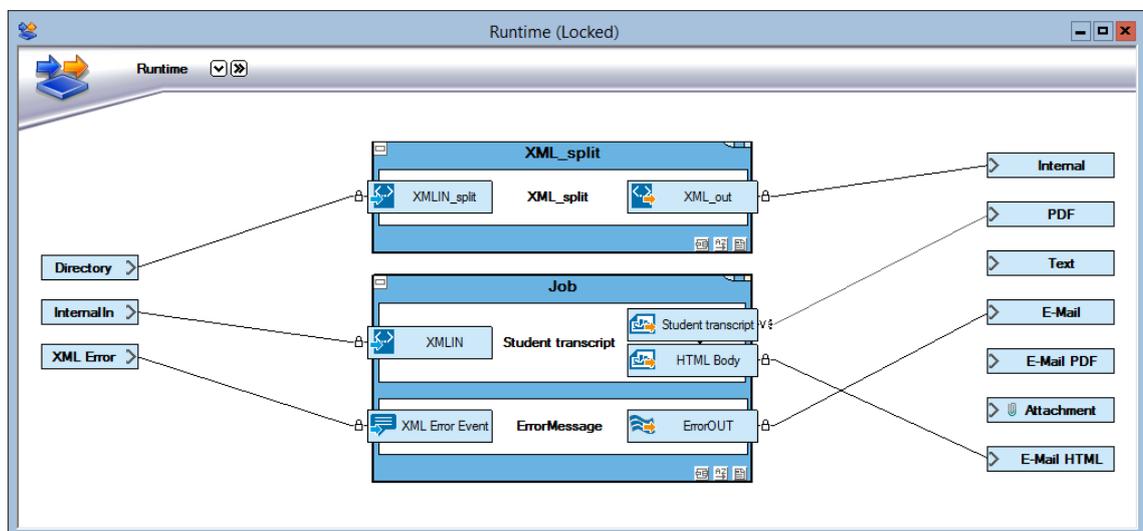


Figure 10. Student_Transcript runtime with three messages. Screenshot [6].

5.2.2 XML_split message

XML_split message is used for splitting XML-files with multiple documents into single documents. This allows better control over individual documents as each document will start a new job. Even though splitting documents gives better control over documents, it may hinder performance.

The message consists of XMLIN event “XMLIN_split” and XMLOUT process “XML_out”. XMLIN event collects XML-files from a directory located on the server. The structure of

the split files will remain the same after they have been processed by the XMLOUT process. The individual XML-files are forwarded to an internal connector which redirects them to the Internalln connector.

5.2.3 Student_transcript message

The Student_transcript message has an XMLIN event “XMLIN” that is used for collecting data from input XML delivered by the Internalln connector. All the fields in the input XML are collected so they can be used later in the scripts and PDF-documents. The XML structure is described in the figure 11.

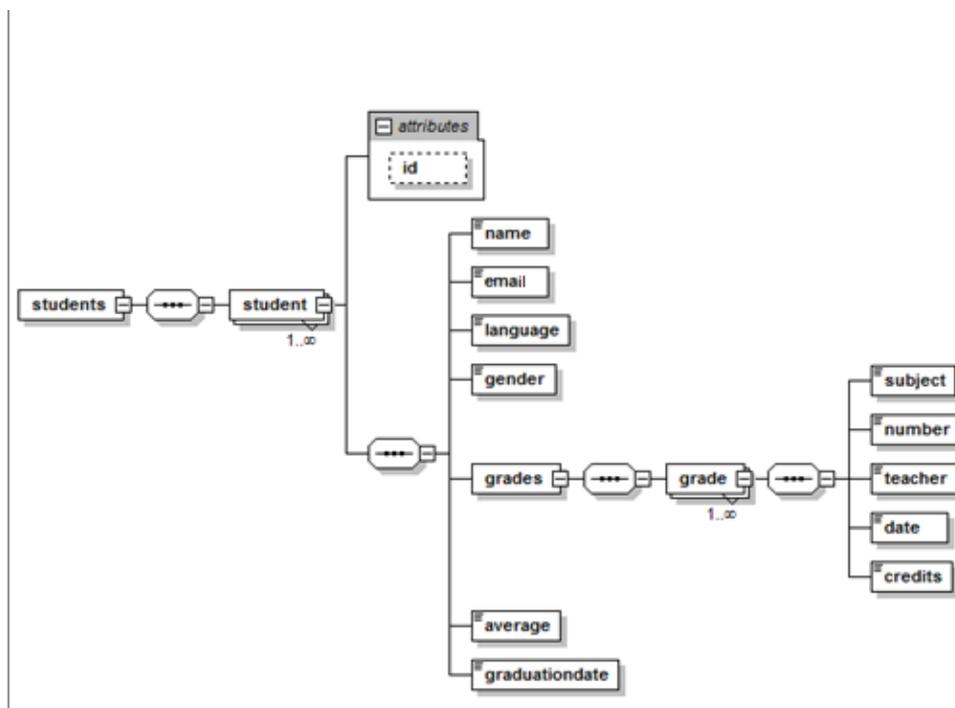


Figure 11. XML Schema used for creating XML document for the StreamServe project. Screenshot [19].

The message has two StoryTeller processes. Student_transcript-process creates the PDF-documents. HTML_body-process creates the content for the HTML email which is used for sending PDF-documents via email.

5.2.4 Error Message

Error Message consists of MessageIn event and StreamOut process. The Error Message is triggered if XMLin validation fails or the process flow is altered from a script using the CancelJob scripting function.

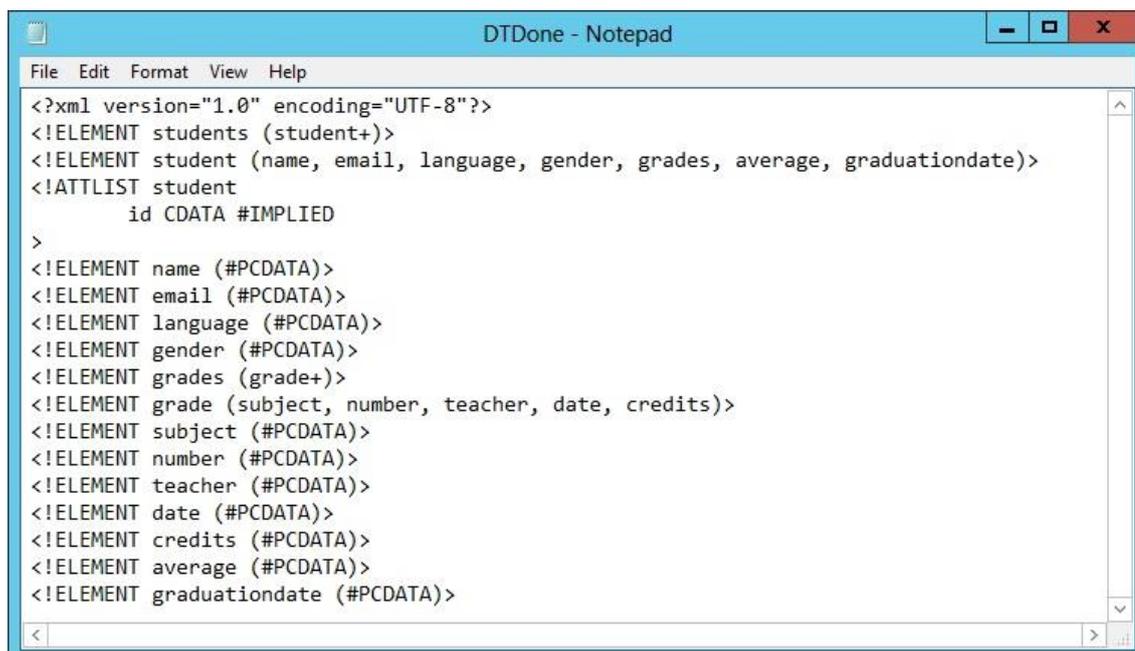
5.3 Exception handling

Exception handling will be added to all the three processing phases. XML validation will be added to the input phase. The formatting and output phase will be monitored by different scripting functions. Selected errors will also be forwarded to the Status Messenger application which will forward the errors to the administrator via email.

5.3.1 Configuring XML validation

Incoming XML files will be validated against a DTD-file. This requires a DTD-file that matches the input XML (f. 12). For this purpose, a DTD-file was created from XML schema (XSD) using Altova XMLSpy software. Once the DTD-file is ready, it is copied to the application server where it is visible to the application. The physical location of the file is added to the input XML to allow the application to locate the DTD-file. It can be added for example in the doctype declaration:

```
<!DOCTYPE students SYSTEM "C:\StreamServe\DTD\DTDone.dtd">
```



```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT students (student+)>
<!ELEMENT student (name, email, language, gender, grades, average, graduationdate)>
<!ATTLIST student
    id CDATA #IMPLIED
>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT language (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT grades (grade+)>
<!ELEMENT grade (subject, number, teacher, date, credits)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT number (#PCDATA)>
<!ELEMENT teacher (#PCDATA)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT credits (#PCDATA)>
<!ELEMENT average (#PCDATA)>
<!ELEMENT graduationdate (#PCDATA)>
```

Figure 12. DTD file that is used for XML validation. Screenshot [20].

To enable XML validation, the correct level of validation has to be selected from the XMLIN event settings as shown in figure 13. This is also where the XML error input connector needs to be configured. After this the XMLIN event is enabled for validation.

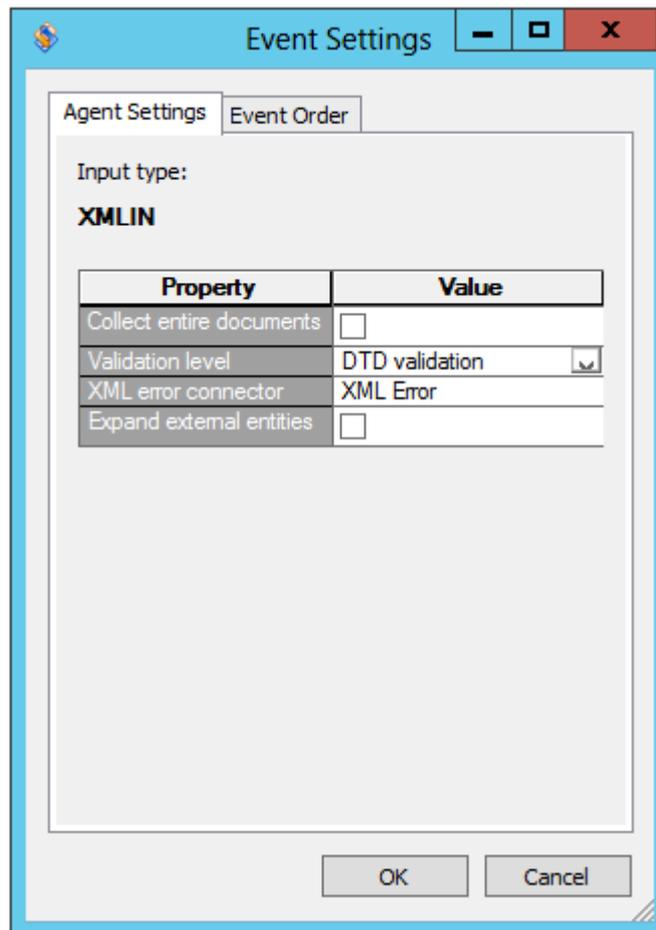


Figure 13. XMLIN event settings dialog, where the XML validation level can be set. Screenshot [6].

5.3.2 Configuring Error Message

Error Message event will catch the errors that occur in XML validation. The event used for collecting errors is MessageIN with preconfigured configuration. As can be seen in figure 14, the configuration is first added to the project's resource set and then imported to the MessageIN event. The configuration includes all the fields and settings needed for the event. The event is triggered automatically if the XML validation fails for some reason.

The Error Message may be used with other errors as well. It can be triggered from a script using the CancelJob scripting function.

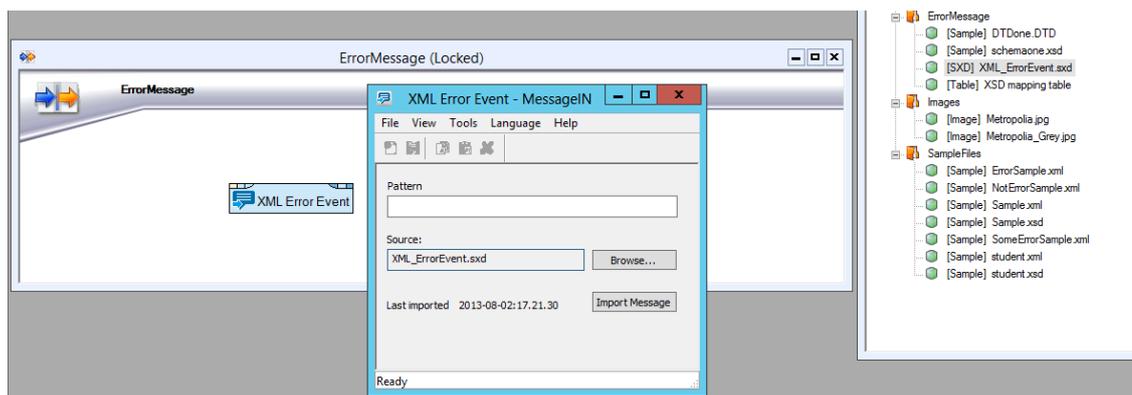


Figure 14. Error Message's event settings. Screenshot [6].

5.3.3 Creating StreamServe_StatusMessenger application

Status Messenger will be configured in an advanced mode. This allows specifying only the needed log identifiers. External job identifiers need to be used together with the log identifiers [12]. This means that an external job identifier needs to be set in a script. For example, the external identifier may be an input filename or student ID number that can be used later to track down the document in error situations.

The log identifiers that are under surveillance needs to be listed in the configuration as shown in figure 15. Status Messenger looks for those identifiers together with external identifiers. Once it finds matching identifier, it will process the notification related to it.

Status Messenger will be configured to run as a separate StreamServer application. This would allow the same Status Messenger application to collect notifications from multiple StreamServer applications.

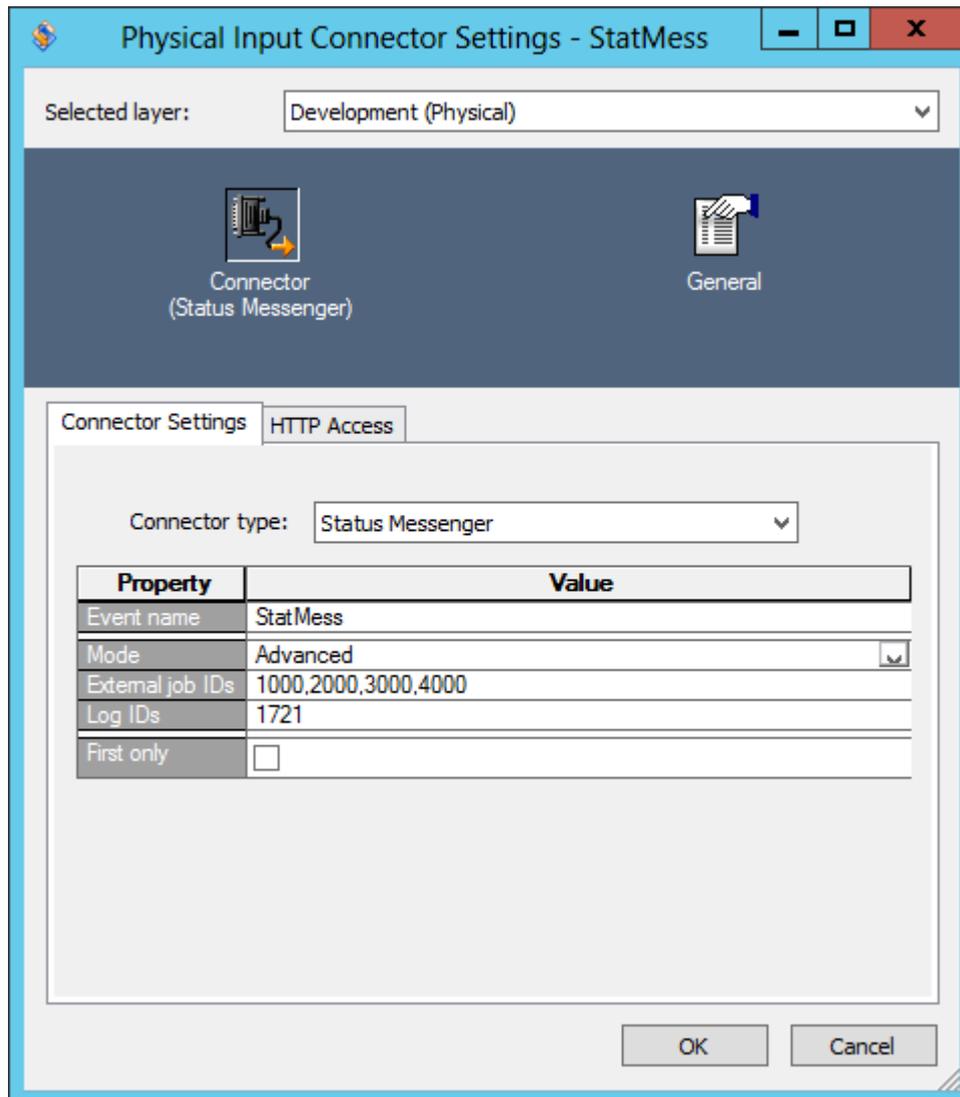


Figure 15. Status Messenger input connector settings. Screenshot [6].

The main application will be configured to create notifications for specified components. This can be done in Project Export Settings. The components that are used for the Error message are deselected to avoid overlapping error messages (see figure 16).

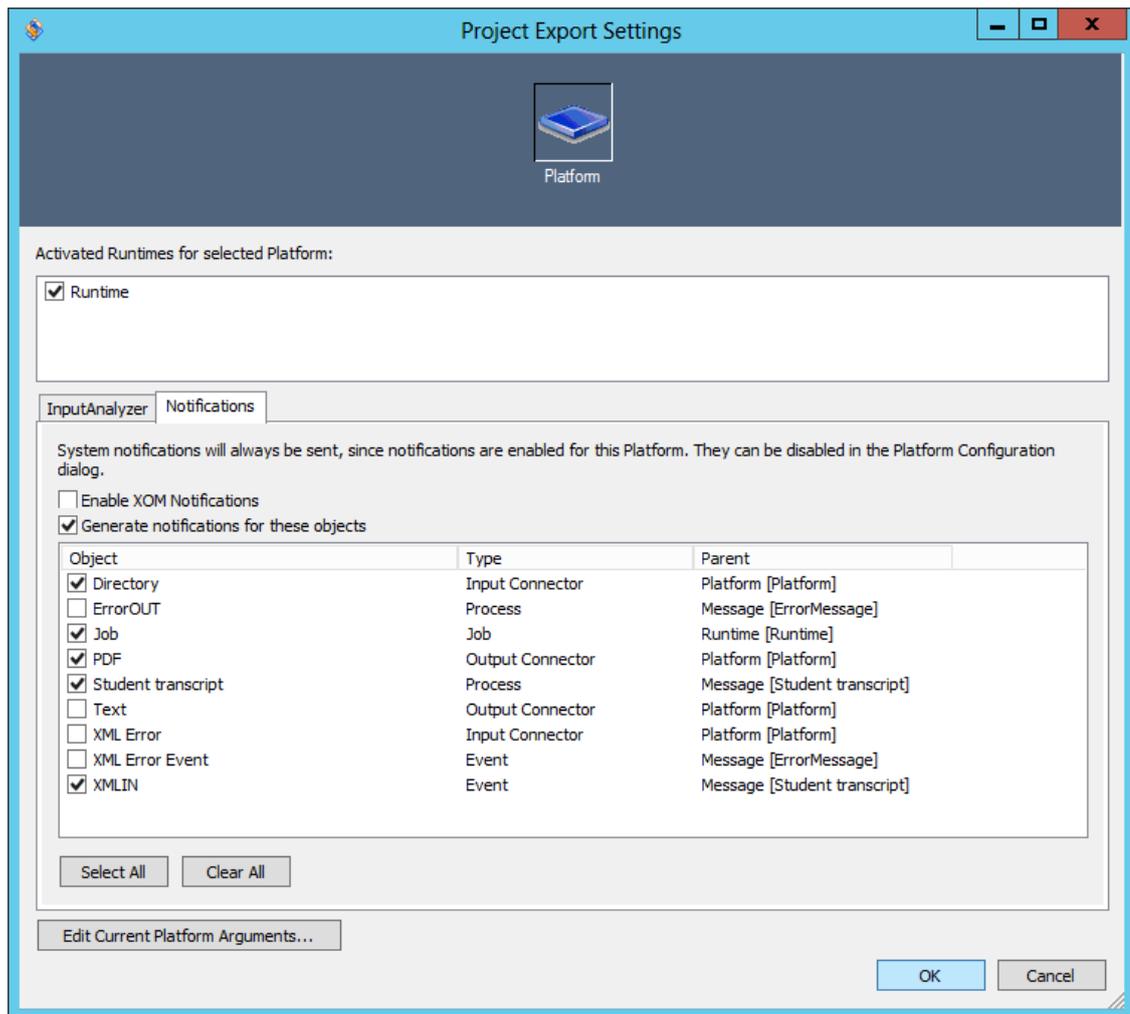


Figure 16. Selecting the components that will create notifications for Status Messenger. Screenshot [6].

5.3.4 Configuring database connection

StreamServe connects the database using the ODBC data source. Once the data source and database have been created, they can be used from StreamServe by using ODBC-functions.

As can be seen in listing 1, the first step is to establish a database connection by using the OdbcConnect function. Once the connection has been established, the data can be inserted into the database using different ODBC-functions. In this case the OdbcExecute function is used to enter data about errors to the StrsError database. The last step is to close the connection by the OdbcDisconnect function.

```
OdbcConnect("ConnectionId", "ODBC_Streamserve64", "sa", "sa");  
OdbcExecute("ConnectionId", $sql);  
OdbcDisconnect("ConnectionId");
```

Listing 1. ODBC functions that are used to open and close database connection.

5.4 Deploying the StreamServer applications

Deploying a StreamServer application is a two-step process. First the configurations need to be exported from Design Center. The second step is to deploy the export-file to an application in Control Center. Once the deployment has been done the application can be started from Control Center.

Two StreamServer applications were created in Control Center for this project. StreamServe_StudentTranscript is the main application that creates student transcript PDFs and sends them as attachments to a student via email. The other application is StreamServe_StatusMessenger which collects errors/notifications from the StreamServe_StudentTranscript application. The error messages are forwarded to an administrator via email. The application properties can be seen in Control Center as shown in figure 17.

The export/deployment process is not going to be covered in more detail, as it is not the main subject of this thesis.

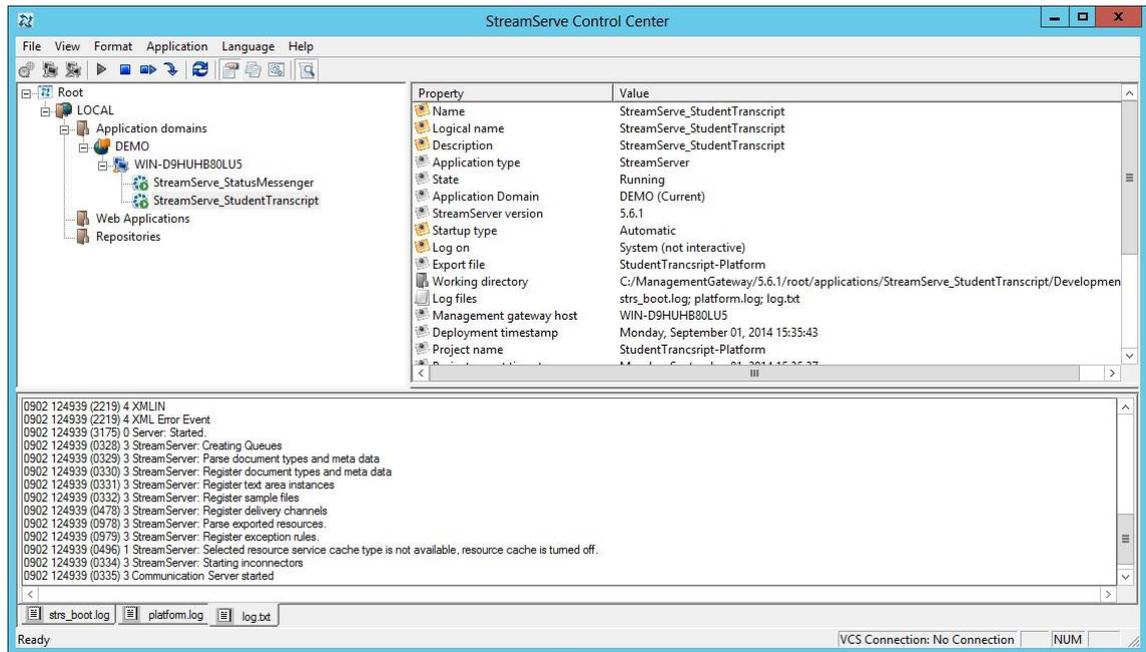


Figure 17. StreamServe Control Center with two StreamServer applications. Screenshot [6].

6 Testing the implementation

Testing the previously configured error handling solution was done by modifying the input XML-files. First the StreamServe_StudentTranscript application was run with valid input data to confirm the application was working correctly. When testing the error handling the XML-files were modified so they included incorrect or missing data. This allowed the testing of all the configured parts of the OpenText StreamServe solution. This chapter will describe the test cases.

6.1 Output files generated by the application

As can be seen in figures 18 and 19, the output of the XML-file is a student transcript PDF which is delivered to the recipient via email and as PDF-file which is stored on disk for archiving purposes. The StreamServe_StudentTranscript application is capable of processing input XML-files that contain data from one or more students. Each student has their own block in XML.

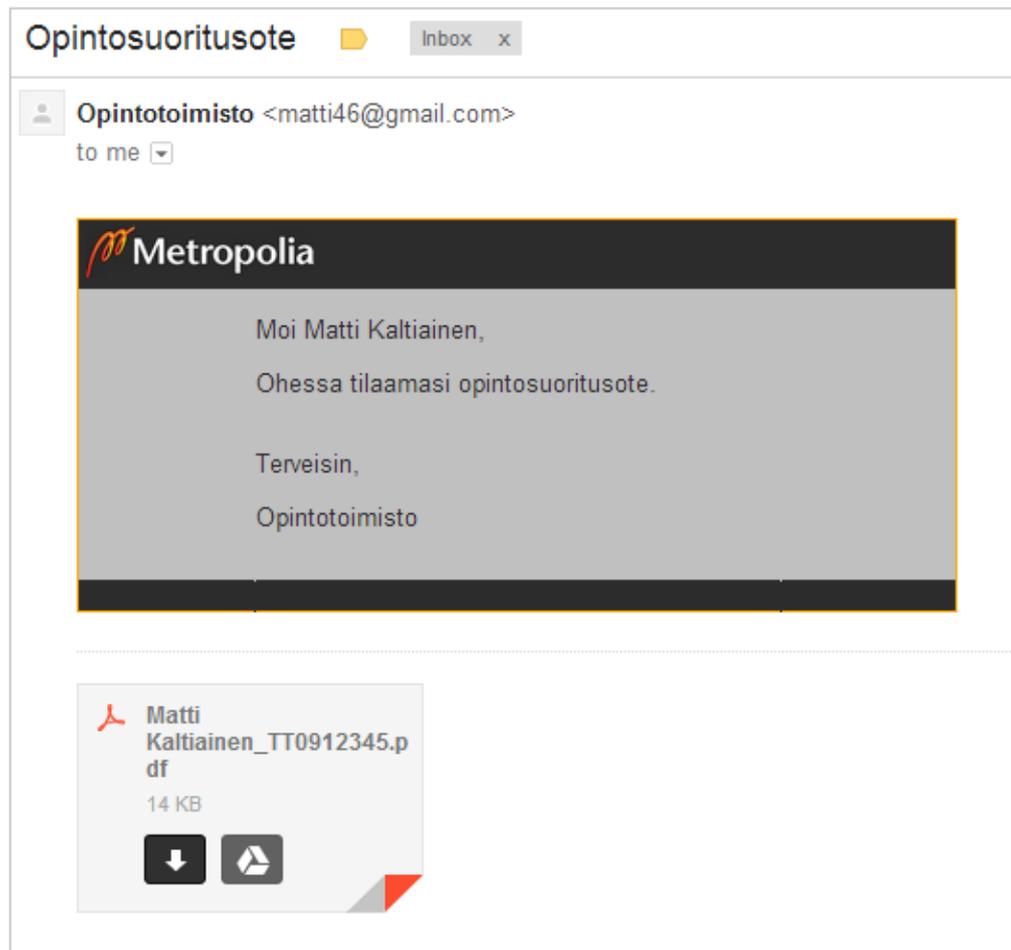


Figure 18. The student transcript is attached to the email as a PDF-file. Screenshot [21].

Metropolia 02.09.2014

OPINTOSUORITUSOTE

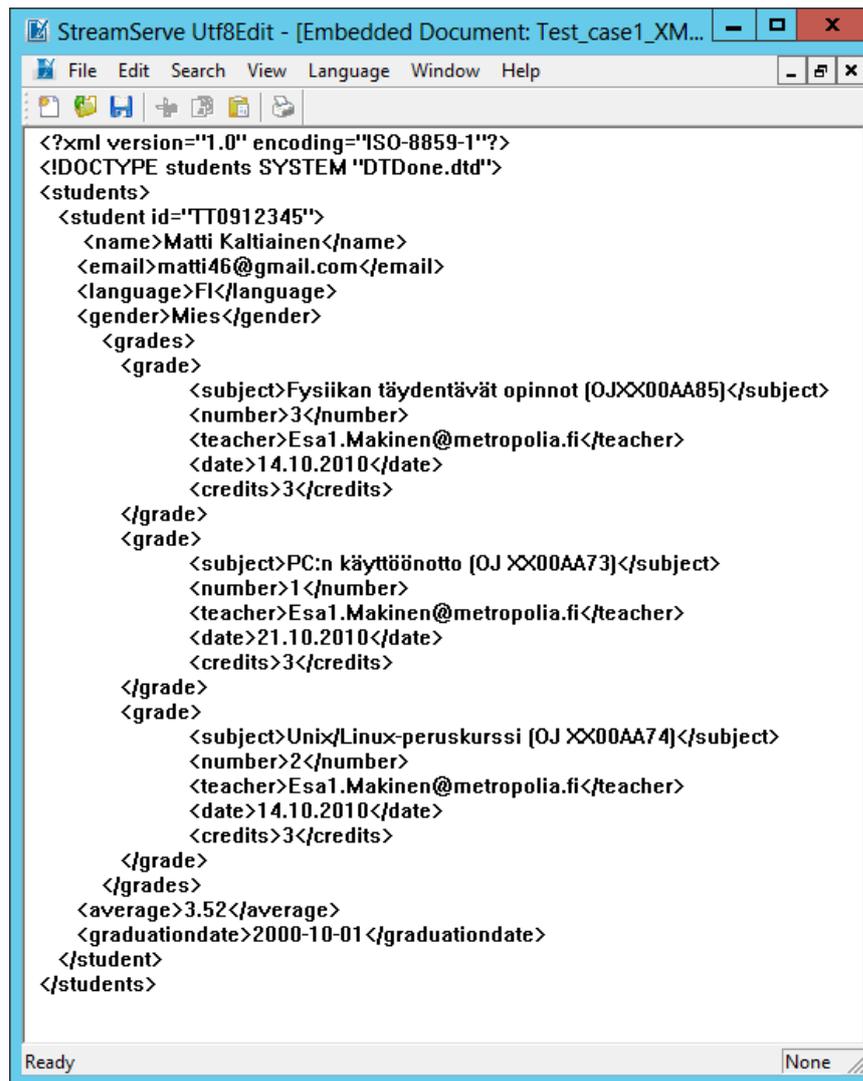
Opiskelijan nimi: **Matti Kalliainen**
ID: **TT0912345**

Opinnon nimi ja tunnus	Arviopvm	Laajuus	Arvosana
Fysiikan täydentävät opinnot (OJXX00AA85)	14.10.2010	3	3
PC:n käyttöönnotto (OJ XX00AA73)	21.10.2010	3	1
Unix/Linux-peruskurssi (OJ XX00AA74)	14.10.2010	3	2
		Yht: 9	Keeklarvo: 2.00

Matti Meikäläinen
Matti Meikäläinen Espoossa 02.09.2014.

Figure 19. Student transcript PDF-file. Screenshot [22].

The XML-file in figure 20 was used for the test cases. The file was modified for testing specific exceptional situations.



```

StreamServe Utf8Edit - [Embedded Document: Test_case1_XM...
File Edit Search View Language Window Help
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE students SYSTEM "DTDone.dtd">
<students>
  <student id="TT0912345">
    <name>Matti Kaltainen</name>
    <email>matti46@gmail.com</email>
    <language>FI</language>
    <gender>Mies</gender>
    <grades>
      <grade>
        <subject>Fysiikan täydentävät opinnot [OJXX00AA85]</subject>
        <number>3</number>
        <teacher>Esa1.Makinen@metropolia.fi</teacher>
        <date>14.10.2010</date>
        <credits>3</credits>
      </grade>
      <grade>
        <subject>PC:n käyttöönotto [OJ XX00AA73]</subject>
        <number>1</number>
        <teacher>Esa1.Makinen@metropolia.fi</teacher>
        <date>21.10.2010</date>
        <credits>3</credits>
      </grade>
      <grade>
        <subject>Unix/Linux-peruskurssi [OJ XX00AA74]</subject>
        <number>2</number>
        <teacher>Esa1.Makinen@metropolia.fi</teacher>
        <date>14.10.2010</date>
        <credits>3</credits>
      </grade>
    </grades>
    <average>3.52</average>
    <graduationdate>2000-10-01</graduationdate>
  </student>
</students>
Ready None

```

Figure 20. XML-file that was used for testing the implementation. Screenshot [6].

6.2 Testing the application with valid XML

The first test was done with valid input data including all the required elements. As it can be seen in listing 2, the StreamServer application log shows that the DTD-file is located and input data is being processed. It is also visible on the log that the PDF-file was created and email was sent to the recipient.

```

0902 115817 (0506) 0 directoryscanner/1.0: C:\StreamServe\In\*.xml found: C:\Stream-
Serve\In\Test_case1_XMLin_valitdation.xml
0902 115817 (1719) 2 JobBegin:directoryscanner=Directory
0902 115817 (4330) 3 Found XML resource: DTDone.dtd
0902 115817 (1726) 2 Collect Event:XMLIN_split < directoryscanner=Directory
0902 120704 (0048) 3 File outconnector: File C:\StreamServe\Out\Matti Kaltiainen_TT0912345.pdf created.
0902 120708 (1052) 3 SMTP connector: Mail was successfully sent

```

Listing 2. XML resource used for validation is found.

The test was successful as the output student transcript PDF was created to disk and delivered to the recipient as expected.

6.3 Testing XMLin validation

Input data is validated in the XMLin event using XML validation. The validation is done against the DTD-file which specifies the elements and values that are mandatory.

The second test was done modifying the input file that was used in the first test. One of the required elements “<name>” was removed from the XML-file to test the validation. After running the test, the result is visible on the application log as shown in listing 3.

```

0902 121722 (1719) 2 JobBegin:directoryscanner=Directory
0902 121722 (4330) 3 Found XML resource: DTDone.dtd
0902 121722 (1719) 2 JobBegin:internal=XML Error
0902 121722 (1726) 2 Collect Event:XML Error Event < internal=XML Error
0902 121722 (5026) 3 Sender of job set to:
0902 121722 (1730) 3 preproc message:XML Error Event < internal=XML Error
0902 121722 (1731) 2 Doing message:XML Error Event < internal=XML Error
0902 121722 (1739) 2 Process:ErrorOUT (v. 13) > E-Mail(scfservice=outconnector/E-Mail)
0902 121722 (5223) 4 Entering StreamOUT process: ErrorOUT
0902 121722 (5225) 4 Leaving StreamOUT process: ErrorOUT
0902 121722 (5105) 3 Receiver of created document set (Document, reciever);,
0902 121722 (1723) 2 JobEnd:internal=XML Error
0902 121722 (3367) 0 XMLIN: An invalid element was encountered (parent, child): student, email
0902 121722 (1723) 2 JobEnd:directoryscanner=Directory
0902 121722 (0201) 1 Failed to process queue item 3E1EF08E-1860-479F-951B-B9CE55CFFF17
0902 121725 (1052) 3 SMTP connector: Mail was successfully sent

```

Listing 3. An invalid element was found in the input XML.

Because the input data is invalid, the job execution is altered and the XML Error event is triggered. The application will create an error message and forward it to an administrator via email.

6.4 Testing Error Message

For testing the Error Message the student ID data is removed from the XML-file. Once the application finds a missing student ID, it alters the job execution using the CancelJob-scripting function. Once the CancelJob function is called, the application will cancel the execution of the current message and move on to Error Message. As it can be seen in listing 4, the job is cancelled and redirected to Error Message.

```
0902 145651 (0506) 0 directoryscanner/1.0: C:\StreamServe\in\looped\*. * found: C:\Stream-
Serve\in\looped\20140902_1Test_case2_Missing_student_id_error_message.xml
0902 145651 (1719) 2 JobBegin:directoryscanner=Internalln
0902 145651 (1726) 2 Collect Event:XMLIN < directoryscanner=Internalln
0902 145651 (5026) 3 Sender of job set to:
0902 145651 (1730) 3 preproc message:XMLIN < directoryscanner=Internalln
0902 145651 (1721) 1 Job canceled:directoryscanner=Internalln
0902 145651 (1730) 3 preproc message:XML Error Event < directoryscanner=Internalln
0902 145651 (1731) 2 Doing message:XML Error Event < directoryscanner=Internalln
0902 145651 (1739) 2 Process:ErrorOUT (v. 13) > E-Mail(scfservice=outconnector/E-Mail)
0902 145651 (5223) 4 Entering StreamOUT process: ErrorOUT
0902 145651 (5225) 4 Leaving StreamOUT process: ErrorOUT
0902 145651 (5105) 3 Receiver of created document set (Document, reciever);,
0902 145651 (1723) 2 JobEnd:directoryscanner=Internalln
0902 145651 (0000) 4 Processing queue item: D90704D8-584F-4B25-A3E0-FE509017939B, connector: 7E197DF1-
29F5-46A1-9D14-8526C0453049
0902 145653 (1052) 3 SMTP connector: Mail was successfully sent
0902 145653 (0000) 4 Finished processing queue item: D90704D8-584F-4B25-A3E0-FE509017939B, connector:
7E197DF1-29F5-46A1-9D14-8526C0453049.
```

Listing 4. The job is cancelled and redirected to Error Message.

6.5 Testing StatusMessenger

StatusMessenger was tested removing the recipient's email address from the XML-input data. Once the application notices that there is no recipient set for the email, it sets an external job ID that is configured in the StatusMessenger's connector settings. As it can be seen in listing 5, the external job ID has been set to 1000.

```
0907 090350 (0506) 0 directoryscanner/1.0: C:\StreamServe\in\looped\*. * found: C:\Stream-
Serve\in\looped\20140907_1Test_case3_Missing_email_data_status_messenger.xml
0907 090350 (3147) 0 External jobid set to: 1000
0907 090350 (1093) 1 SMTP connector: No valid TO, CC or BCC address for connector SmtOutConnector2
0907 090350 (1077) 1 SMTP connector: Error when sending e-mail: (0x8000001f: Strs::ErrorGeneric::invalidConfigu-
rationError)
0907 090350 (1053) 1 SMTP connector: Mail was not sent
```

Listing 5. The external job ID is set.

StatusMessenger recognizes the External job ID and collects the notification from the database. The notification will trigger the StatusMessenger job, which will process the error message and forward it to an administrator via email as shown in figure 21.

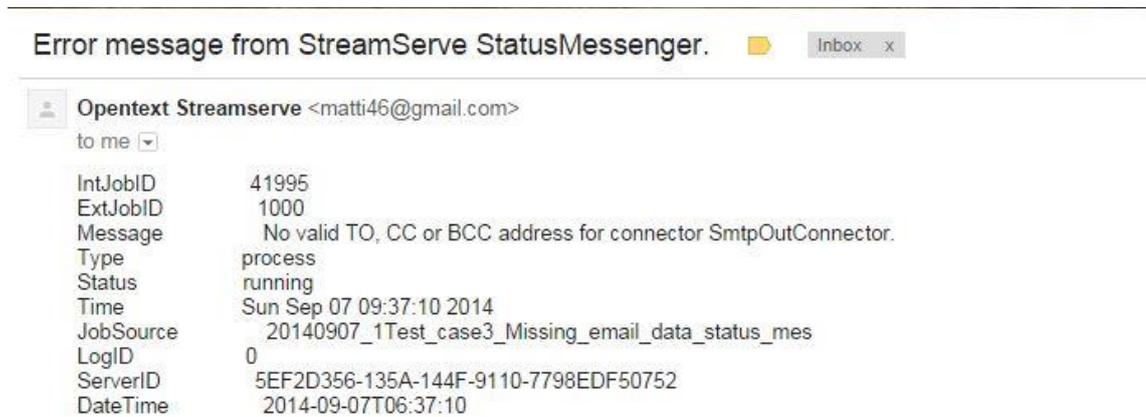


Figure 21. The ExtJobID and error message are included in the email message. Screenshot [21].

6.6 Writing errors to database

As can be seen in figure 22, the errors occurring in the test cases were written to the StrsError database. The StreamServe jobs were monitored by the GetJobStatus function, which gives information about failed jobs. All the failed StreamServe jobs were successfully written to the database which can be seen in the screenshot below.

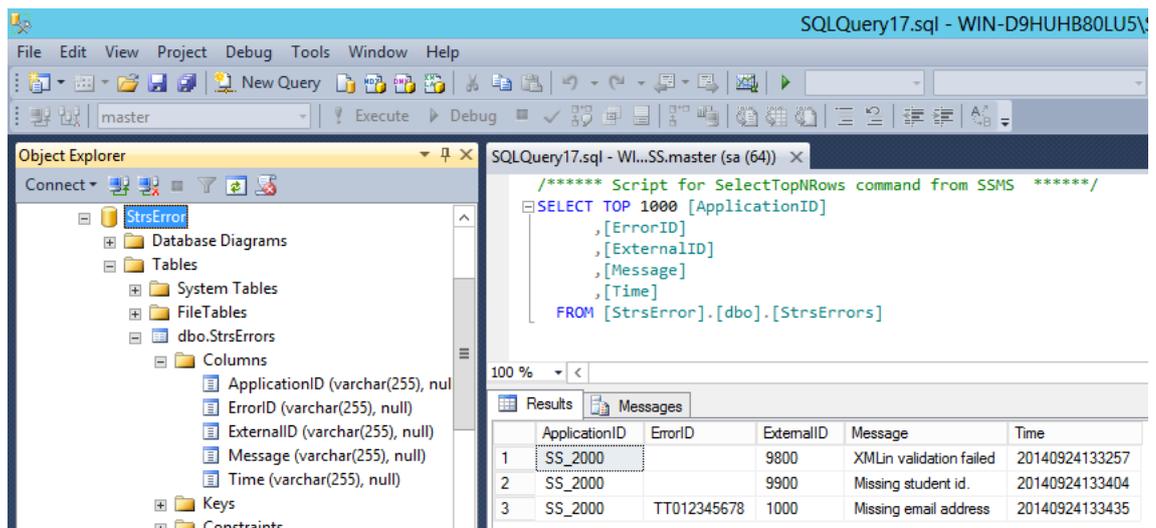


Figure 22. The StrsError database. Screenshot [18].

7 Conclusion

OpenText StreamServe offers a wide range of tools for exception handling. As the StreamServe solutions can differ considerably from each other, each exception handling solution needs to be designed for that specific setup. There are many factors that need to be considered when designing an exception handling solution. For example, it is important to consider what kinds of exceptions need to be covered and how exceptions can be identified and caught.

This thesis introduced most of the options OpenText StreamServe offers for exception handling. Some of these options were also configured and tested in the implementation part. The implemented and tested configurations worked as they were planned. All the three processing states (input, formatting and output) were covered in the test cases. However, performance testing was not included as it would have required larger test cases.

It has to be noted that some of the configurations overlapped with each other, which would affect negatively the performance. Because of the unnecessary overlapping of the exception handling, this setup would not be suitable in larger environments where high performance is needed. However, the independent components such as XMLin-validation or Error Message could be used as they were implemented. As exception handling may hinder the performance, it is recommended to implement exception handling only where it is necessary.

It was found easier and more effective to implement exception handling in solutions where documents are not being delivered forward as soon as they are created. That is because there is more time to react to the situation and prevent forwarding invalid documents. Exception handling is more challenging in online printing where documents are delivered as soon as they are created. In such a situation there is no time to react to the exception and the StreamServe application may not be able to give informative feedback to the end user.

The individual solutions introduced in this thesis could be implemented in production environment. This thesis can also provide guidance for other StreamServe developers to design comprehensive exception handling solutions in their projects.

References

1. OpenText Customer Communications Management [online].
URL: <http://www.opentext.com/what-we-do/products/customer-experience-management/customer-communications-management/opentext-streamserve>.
Accessed 14 January 2015.
2. Tutorialspoint [online].
URL: http://www.tutorialspoint.com/java/java_exceptions.htm.
Accessed 10 January 2015.
3. PHP Exception Handling [online].
URL: http://www.w3schools.com/php/php_exception.asp.
Accessed 10 January 2015.
4. Microsoft Developer Network [online].
URL: [http://msdn.microsoft.com/en-us/library/seyhszts\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/seyhszts(v=vs.110).aspx).
Accessed 10 January 2015.
5. W3Schools.com [online].
URL: http://www.w3schools.com/js/js_errors.asp
Accessed 10 January 2015
6. OpenText StreamServe [computer program]. Version 5.6.1.
Open Text Corporation; 2011.
7. OpenText StreamServe 5.6.1 Design Center User Guide [pdf]; 2013.
8. StreamShare InDepth Part1 – StreamServer Stefan Cohen [online]; 2010.
URL: <http://streamshare.streamserve.com/Articles/Article/?articleId=157>.
Accessed 10 January 2015.
9. OpenText StreamServe 5.6.1 Online Help, Job Execution Phases [online]; 2013.
URL: <http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=Scripting%20reference/ScriptLangRef.1.05.html>.
Accessed 10 January 2015.
10. OpenText StreamServe 5.6.1 XMLIN User Guide [pdf]; 2013.
11. OpenText StreamServe 5.6.1 Online Help, CancelJob [online]; 2013.
URL: <http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=Scripting%20reference/ScriptingFunctions.2.074.html>.
Accessed 10 January 2015.

12. OpenText StreamServe 5.6.1 Online Help, Creating an XMLIN validation error Message [online]; 2013.
URL: http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=XMLIN/XMLIN_validate.4.2.html.
Accessed 10 January 2015.
13. OpenText StreamServe 5.6.1 Status Messenger User Guide [pdf]; 2013.
14. OpenText StreamServe 5.6.1 Online Help, Script Functions Reference [online]; 2013.
URL: <http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=Scripting%20reference/ScriptingFunctions.2.001.html#1141279>.
Accessed 10 January 2015.
15. OpenText StreamServe 5.6.1 Online Help, GetJobStatus [online]; 2013.
URL: <http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=Scripting%20reference/ScriptingFunctions.2.179.html>.
Accessed 10 January 2015.
16. OpenText StreamServe 5.6.1 Online Help, IOErrText [online]; 2013.
URL: <http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=Scripting%20reference/ScriptingFunctions.2.229.html>.
Accessed 10 January 2015.
17. OpenText StreamServe 5.6.1 Online Help, Terminate [online]; 2013.
URL: <http://onlinehelp.stream-serve.com/5.6.1/en/wwhelp/wwhimpl/js/html/wwhelp.htm#href=Scripting%20reference/ScriptingFunctions.2.471.html#1359693>.
Accessed 10 January 2015.
18. Microsoft SQL Server Express 2012 [computer program]. Version 11.0.2100.60. Microsoft Corporation; 2012.
19. Altova XMLSpy [computer program]. Altova; 2014.
20. Notepad [computer program]. Microsoft Corporation; 2012.
21. Google Gmail [computer program]. Google Inc; 2014.
22. Adobe Reader XI [computer program]. Version 11.0.4. Adobe Systems; 2014.