

OpenText™ Output Transformation for AppEnhancer

Integration Guide

This document provides information about the installation of and migrating content for OpenText Output Transformation for AppEnhancer. The information provided is intended for use by administrators.

VDTOTS240400-AIG-EN-1

OpenText™ Output Transformation for AppEnhancer Integration Guide

VDTOTS240400-AIG-EN-1

Rev.: 2024-Nov-20

This documentation has been created for OpenText™ Output Transformation for AppEnhancer CE 24.4.

It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111

Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440

Fax: +1-519-888-0677

Support: <https://support.opentext.com>

For more information, visit <https://www.opentext.com>

© 2024 Open Text

Patents may cover this product, see <https://www.opentext.com/patents>.

Disclaimer

No Warranties and Limitation of Liability

Every effort has been made to ensure the accuracy of the features and techniques presented in this publication. However, Open Text Corporation and its affiliates accept no responsibility and offer no warranty whether expressed or implied, for the accuracy of this publication.

Table of Contents

1	Introduction	7
2	Migrating from legacy ERMX applications	9
2.1	ERMX and OpenText Output Transformation Server functional comparison	9
2.1.1	Design and Production environments	10
2.1.2	Using Unicode UTF-8 instead of ASCII encoding for text documents ..	10
2.1.3	Comparing ERMX functionality to OpenText Output Transformation Server functionality	11
2.1.4	Comparing processing of supported file types	12
2.1.5	Extracting fields from a document in ERMX and OpenText Output Transformation Server	13
2.1.6	Relationship between Page and Line Procedures	14
2.1.7	XDS Script example in OpenText Output Transformation Server	14
2.1.8	Sample input file	16
3	Configuring Output Transformation for AppEnhancer projects	17
3.1	Basic anatomy of an Output Transformation for AppEnhancer project	17
3.1.1	Creating the configuration file in sequence	20
3.1.2	Creating an Output Transformation Engine project to index and transform the input file	21
3.1.3	Creating a new Output Transformation Engine project using the New Component Wizard	22
3.1.3.1	Select Project Name and Location	22
3.1.3.2	Select MFCT File	22
3.1.3.3	Select Parser, Components and Generators	23
3.1.4	Configuring the Output Transformation Engine project input and output file locations	25
3.1.4.1	Configuring the Parser FdInput parameter	26
3.1.4.2	Configuring the Linedata parser filedefs	26
3.1.4.3	Configuring the PDF Generator output file	27
3.1.4.4	Executing the project	27
3.1.5	Configuring the Output Transformation Engine project to extract fields for indexing	27
3.1.5.1	Setting up an XFT Document	28
3.1.5.2	Turning off DOM properties on XFT Documents	28
3.1.5.3	Defining XFT Text fields to extract data	29
3.1.5.4	Defining XFT fields that use dynamic positioning	32
3.1.6	Configuring the Output Transformation Engine project Indexer component	38

3.1.7	Configuring the Output Transformation Engine project Index Writer component	40
3.1.8	Validating Output Transformation Engine reports	40
3.2	Configuring the OpenText Output Transformation Server process flow	41
3.2.1	Configuring an AppEnhancer Loader component	41
3.2.2	AppEnhancer Loader component prerequisites	41
3.2.3	Configuring the AppEnhancer Loader component	41
3.2.4	Reading the AppEnhancer Loader audit logs	44
3.2.5	Configuring the FileEvent	47
3.2.5.1	Using FileEvent to locate source files for processing	47
3.2.5.2	Configuring the FileEvent XDoc mappings	49
3.2.5.3	Configuring the FileEvent file name job variables	49
3.2.6	Configuring the process flow to use XDocs and job variables	50
3.2.6.1	Setting up a new job variable to define the output folder	51
3.2.6.2	Mapping the Output Transformation Engine FdInput to XDoc	52
3.2.7	Configuring the Output Transformation Engine project to use the input file job variable	52
3.2.8	Configuring the Output Transformation Engine project to use default job variables for independent execution	53
3.3	Creating and configuring a process flow for end-to-end testing	54
3.3.1	Creating the process flow for end-to-end testing	54
3.3.2	Adding a GetFile component to the process flow	55
4	Setting up AppEnhancer	57
4.1	Required IIS settings	57
4.2	Minimum required permissions for AppEnhancer users	58
5	Recommended Settings	59
5.1	Optimizing performance in an Indexing project	59
5.2	Performance tuning	59
6	Special implementations	61
6.1	Skipping pages in Linedata Parser projects when loading into AppEnhancer	61
6.2	Creating multiple AppEnhancer documents that include the same page	64
6.3	Using scripting for complex document breaking rules	69
7	Troubleshooting	73
7.1	When loading input, I am getting a "Request Entity Too Large" error message.	73
7.2	After loading is finished, I don't see my overlays.	73
7.3	An error message stating "Expected next offset (page) at >=2 but found 1" was shown.	74

7.4	The audit log does not provide any error message details.	75
7.5	The audit log shows an “Err[Entity input stream has already been closed.]” message.	76
7.6	The audit log shows an “Err[org.glassfish.jersey.client.InboundJaxrsResponse cannot be cast to com.xenos.applicationxtender.api.rest.BatchesResponse]” message.	76
7.7	There is a discrepancy in the number of documents loaded using Output Transformation Server versus the number loaded by ERMX. ..	76
7.8	After upgrading AppXtender to AppEnhancer, I am getting a 404 error.	78
7.9	I received an “The index could not be saved because a duplicate index already exists” error message.	79

Chapter 1

Introduction

This document is intended for users who will be loading documents into OpenText™ AppEnhancer using OpenText™ Output Transformation Server.

This document also includes specific guidance for users who are migrating from existing projects created in OpenText™ ApplicationXtender Reports Management (ERM).

Chapter 2

Migrating from legacy ERMX applications

This section focuses on comparing and explaining the OpenText Output Transformation Server features that should be used when migrating projects from ERMX.

2.1 ERMX and OpenText Output Transformation Server functional comparison

ERMX is a Windows-only application that was specifically designed to perform this series of tasks:

1. Locate files added to a folder that need to be loaded into AppEnhancer.
2. If the content is a printstream (AFP, Metacode, or PCL) or a linedata file, convert it into PDF or ASCII.
3. Normalize the PDF or ASCII document content into a text image that has rows and columns of ASCII data.
4. Index the file into logical documents and collect index fields (metadata) using the user defined XDS procedures to process the text image content.
5. Load the documents (PDF or ASCII) into AppEnhancer using the indexed data.

In comparison to ERMX, OpenText Output Transformation Server is a more complex and flexible process flow engine that enables the creation of custom applications to meet many different document and information handling use cases. It also serves as a platform that runs specialized application services such as OpenText™ Document Accessibility and OpenText™ Alternate Text Manager.

Using a subset of the available components, OpenText Output Transformation Server can be configured to perform the same or equivalent tasks performed by ERMX.

OpenText Output Transformation Server runs on Java technology and therefore, is not restricted to only Windows platforms. For more information about supported operating systems and environments, see the OpenText Output Transformation Server Release Notes for your current product version.

2.1.1 Design and Production environments

Like ERMX, projects must be created in OpenText Output Transformation Server to locate and index documents based on user defined business rules. These projects are created and tested inside the Output Transformation Designer desktop application user interface and once completed, are deployed to an OpenText Output Transformation Server instance running in a test or production environment where they will execute all workloads submitted to them.

In a production environment, OpenText Output Transformation Server will run as a server-based application inside a Java application server, such as Apache Tomcat or Websphere Liberty, with Apache Tomcat being the most common deployment.

For more information about installing and launching Output Transformation Designer, see OpenText™ Output Transformation Server Installation Guide.

For more information about installing and deploying OpenText Output Transformation Server and projects to an application server, see OpenText™ Output Transformation Server User Guide.

2.1.2 Using Unicode UTF-8 instead of ASCII encoding for text documents

In the original ERMX application and help documentation, there are many references to ASCII encoding text files as the primary ingestion format. In the context of the more modern AppEnhancer and OpenText Output Transformation Server software, the preferred format is UTF-8, which is based on ASCII but provides full support for all Unicode characters. By using UTF-8 for all content, any encoding language issues will be avoided.

It is also recommended to configure the **JavaCodepageName** parameter in the Generic Index Writer to use the **UTF-8** value. This will ensure that any extended Unicode characters such as CJK or extended European characters are preserved correctly.



Note: Unicode UTF-8 encoding does not apply to PDF content.

For more information about setting up UTF-8 encoding using the Linedata Parser, see [“Configuring the Linedata parser filedefs” on page 26](#).

2.1.3 Comparing ERMX functionality to OpenText Output Transformation Server functionality

This section provides an explanation of key ERMX concepts and their OpenText Output Transformation Server equivalents:

ERMX functionality	OpenText Output Transformation Server functionality
Runs on Windows platforms	Runs on multiple platforms, including Windows and Linux For more information about supported platforms, see Output Transformation Server Release Notes for your product version.
Detecting input files using source specifications	Configuring a File Event to monitor a folder for file patterns, which then executes a named process flow for each file found.
AppXtender Reports Mgmt Print Stream Processor (converts AFP/Metacode/PCL print streams into PDF or ASCII files for loading into AppEnhancer)	Use Output Transformation Engine component inside OpenText Output Transformation Server process flows to convert the print streams into PDF. Print streams and PDF files cannot be converted into text only files.
AppXtender Reports Mgmt Report Processor	OpenText Output Transformation Server engine
AppXtender Reports Mgmt Report Configuration Administrator	Output Transformation Designer and Output Transformation Server Manager
Source specifications: Defines location and file name masks/INI file rules for detecting new files to process	Options in OpenText Output Transformation Server are flexible with File Event being the most equivalent option, but others exist.
Uploads reports as new documents to the AppXtender document storage location.	OpenText Output Transformation Server makes REST calls to the AppEnhancer server and then AppEnhancer server stores documents.
Direct Database update	OpenText Output Transformation Server makes REST calls to the AppEnhancer server and then AppEnhancer server updates database.
Extraction Definition Script (XDS)	XFT component graphical field definitions in conjunction with an Indexer field configuration. For complex business rules, a JavaScript component may also be used to supplement the XFT processing to finalize the index values.

ERMX functionality	OpenText Output Transformation Server functionality
For Visual Basic scripting, ERMX provided a way for users to write VB Script files and execute them from the XDS to perform custom business logic.	OpenText Output Transformation Server provides JavaScript (ECMA script) components to allow for custom business logic. Scripts may also be used as definitions for any configuration parameter in OpenText Output Transformation Server, allowing more dynamic parameter values. It also enables users to create custom Java plug-ins, which offer better performance.
XDS Visual Tester	Exploring the XFT results in Output Transformation Designer.

2.1.4 Comparing processing of supported file types

This section lists the types of input documents supported by ERMX and compares how they are handled in ERMX and OpenText Output Transformation Server.

File type to be processed	AppXtender Reports Mgmt	OpenText Output Transformation Server
PDF files	Processed with AppXtender Reports Mgmt Report Processor and the files do not need to be converted before processing.	Use an Output Transformation Engine project with PDF Parser, XFT, Indexer and Index Writer components.
ASCII files	Processed with AppXtender Reports Mgmt Report Processor and the files do not need to be converted before processing.	Use an Output Transformation Engine project with Linedata Parser, XFT, Indexer and Index Writer components. Files can be loaded as ASCII or if PDF is required, then use the PDF Generator.
AFP, Metacode, or PCL print stream files	Files must be first converted with AppXtender Reports Mgmt Print Stream Processor. Then you can process the resulting PDF report files with the AppXtender Reports Mgmt Report Processor.	Files are loaded as a PDF. Use an Output Transformation Engine project with corresponding AFP/Metacode/PCL Parsers, indexing components (XFT, Indexer, and Index Writer), and the PDF Generator.

File type to be processed	AppXtender Reports Mgmt	OpenText Output Transformation Server
EBCDIC, EBCDIK, or non-standard ASCII print stream files	Files must be first converted with AppXtender Reports Mgmt Print Stream Processor. Then you can process the resulting standard ASCII report files with the AppXtender Reports Mgmt Report Processor.	<p>Use an Output Transformation Engine project with Linedata Parser, XFT, Indexer, and Index Writer components.</p> <p>Customize the Linedata Parser parameters to match the encoding, record format and carriage control properties of input data.</p> <p>Files can be loaded as ASCII or if PDF is required, then use the PDF Generator.</p>

2.1.5 Extracting fields from a document in ERMX and OpenText Output Transformation Server

In ERMX, users would create and configure “Procedures” to extract fields from the page content for indexing (or to create BI data fields). The data from which the fields are extracted were simple text-based files, organized by rows and columns.

If the input report was a PDF, ERMX would extract the text content from the PDF and create a flat text representation of the PDF to allow it to use row and column-based rules. This PDF-to-Text functionality has limitations, such as spacing issues (extra space characters or missing space characters) and overlapping data.

In OpenText Output Transformation Server, the behavior is the opposite: content is represented as full graphic capable pages instead of flat text data, which feature pages with margins, different font sizes, and full color images. OpenText Output Transformation Server will convert text and line data files into rendered pages with a specific font and point size that is configured by the user.

Instead of row and column numbers being used to define what data to extract, field bounding boxes are drawn on page using the mouse and keyboard in a high resolution (72000 DPI) coordinate system.

2.1.6 Relationship between Page and Line Procedures

In ERMX, there was a concept of Page Procedures and Line Procedures that are executed differently.

In OpenText Output Transformation Server, XFT field processing is executed on the entire page, so in this manner the field extraction concept is simplified.

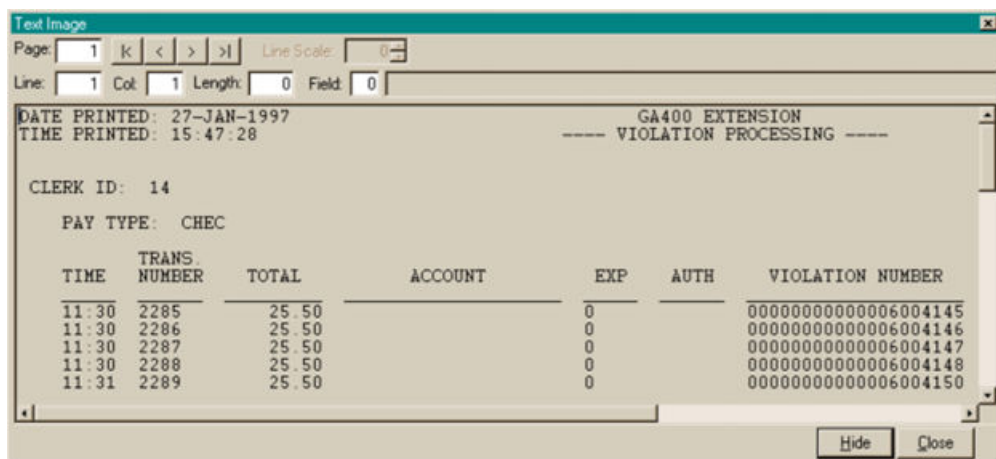
2.1.7 XDS Script example in OpenText Output Transformation Server

This section reviews the sample script from the ERMX documentation and lists the corresponding XDS script for reference. The relationship between ERMX XDS and OpenText Output Transformation Server XFT is also discussed.

Subsequent sections in this help document walk through the process of configuring OpenText Output Transformation Server to load the same document into AppEnhancer.

ERMX Text Image view

The following figure shows the text image from the ERMX user interface:



The following table examines the XDS script:

Segment number	XDS script	Description
1	LINEPROC Violations	The LINEPROC is named "Violations."

2.1. ERMX and OpenText Output Transformation Server functional comparison

2	COL 2 IF MATCH "CLERK ID:" BEGIN COL 13 EXTRACT clerkid 2 INLINE STORE clerkid "CLERK ID" END	At COL 2, if the string "CLERK ID" is matched, then the group of instructions between the BEGIN and END instructions will be executed. The group of instructions specify going to COL 13, extracting 2 characters as the variable clerkid, and storing the variable value in the AppXtender index field "CLERK ID."
3	COL 12 IF MATCH /\d\d\d\d/ BEGIN EXTRACT transno 4 INLINE STORE transno "TRANSACTION NO." END	At COL 12, if a pattern of 4 digits is matched, then the group of instructions between the BEGIN and END instructions will be executed. The group of instructions specify extracting 4 characters as the variable transno, and storing the variable value in the AppXtender index field "TRANSACTION NO."
4	COL 68 IF MATCH "000000" BEGIN COL 81 EXTRACT violno 7 INLINE STORE violno "VIOLATION NO." END	At COL 68, if the string "000000" is matched, then the group of instructions between the BEGIN and END instructions will be executed. The group of instructions specify going to COL 81, extracting 7 characters as the variable violno (the leading zeros will be skipped), and storing the variable value in the AppXtender index field "VIOLATION NO."
5	IF MATCH "REPORT NO." BEGIN COL 122 EXTRACT repno 10 INLINE STORE repno "REPORT NO." END	At COL 111, if the string "REPORT NO." is matched, the group of instructions between the BEGIN and END instructions will be executed. The group of instructions specify to go to COL 122, extract 10 characters as the variable reportno, and store the variable value in the AppXtender index field "REPORT NO."

6	END	The last END instruction specifies the end of the LINEPROC Violations.
---	-----	--

2.1.8 Sample input file

When creating ERMX projects, users are required to provide a sample input file that is used for testing the project. OpenText Output Transformation Server does not have a dedicated concept to enforce this.

However, there is a common strategy for OpenText Output Transformation Server users to follow for this practice. For more information about configuring the `txtdemo-load-process-test.xProcessFlow` sample project to reference files for project development and testing, see [“Basic anatomy of an Output Transformation for AppEnhancer project” on page 17](#).

Chapter 3

Configuring Output Transformation for AppEnhancer projects

This section discusses the configuration of OpenText Output Transformation Server and Output Transformation Engine to create an Output Transformation for AppEnhancer project.

3.1 Basic anatomy of an Output Transformation for AppEnhancer project

Projects created in OpenText Output Transformation Server to load documents into AppEnhancer can be configured to meet basic and complex use cases, depending on the business requirements.

The most basic projects follow this structure:

1. FileEvent configured to listen for new input files added to a folder.
2. FileEvent invokes a named Process Flow to process each new file.
3. The Process Flow contains an Output Transformation Engine project, followed by the AELoader component.
4. The Output Transformation Engine project will:
 - a. Extract metadata and divide the file into individual documents through document breaking.
 - b. Optionally, convert the document format and/or manipulate the document structure. For instance, to remove unwanted or blank pages.
 - c. Create an index file with page ranges and index values that will be loaded into AppEnhancer.
5. AELoader component processes the index file created by Output Transformation Engine and loads the required data into AppEnhancer.
 - a. The data to load can be the original input file, or a modified/transformed version of the file created by Output Transformation Engine.
6. A test process flow to manually test the entire load process using a chosen input file.

More complex projects will perform the same functionality as previously described, but may also perform additional steps, such as conditional routing to different paths in a process flow, custom email notifications, archiving of data files, or database inserts or lookups. Additional components must be added to the Process Flow configuration to implement these requirements.

The following shows a sample folder structure for a basic project named `txtdemo`:

```
txtdemo/
├── txtdemo-file-listener.xFileEvent
├── txtdemo-index-transform.d2eproj
├── txtdemo-load-process.xProcessFlow
├── txtdemo-load-process-test.xProcessFlow
├── input/
│   └── test-input.txt
├── output/
├── watchdir/
└── resources/
    ├── txtdemo-index-convert-fontmap.d2emfct
    ├── txtdemo-index-convert.d2efontenv
    └── txtdemo-get-test-file.xGetFile
```

An overview of each file and folder is shown in the following sections and additional topics in this help document go on to explain in more detail how each file should be created.

txtdemo-file-listener.xFileEvent

This file configures the File Event that listens for files to appear in the `watchdir` folder. For each file that appears, the `txtdemo-load-process.xProcessFlow` project will be executed to process that file and load it into AppEnhancer.

txtdemo-index-transform.d2eproj

This file configures the Output Transformation Engine component to:

- Identify and extract index fields from the input document.
- Break the input file up into logical documents and index them.
- If required, transform the input file into a format supported by AppEnhancer (ASCII/UTF-8 or PDF).
- Create an index file that includes all indexing information and page ranges for all logical documents. This data is used later by the OpenText Output Transformation Server AELoader component.

txtdemo-load-process.xProcessFlow

A Process Flow is a chain of OpenText Output Transformation Server components that are executed in sequence.

The process flow in this file contains two components:

- An Output Transformation Engine component that calls the `txtdemo-index-transform.d2eproj` project.

- An AELoader component that reads the index file created by the Output Transformation Engine project and loads the data into AppEnhancer using the AppEnhancer REST API.

txtdemo-load-process-test.xProcessFlow

This is a simple process flow used for testing the process during project development. It is configured to use a Get File component that points to the `input/test-input.txt` input file and associate that document with an XDoc input mapping. It then executes `txtdemo-load-process.xProcessFlow` to process the XDoc.

In a production environment, `txtdemo-file-listener.xFileEvent` actively submits input files through the XDoc when calling the `txtdemo-load-process.xProcessFlow` process. This test process provides an alternative to that.

The input file can be easily changed to test with different input samples.

input/test-input.txt

The `input` folder contains one or more sample input files that can be used for project development, testing, and troubleshooting.

In production runtime environments, files gathered by the File Event are used as input instead of this test file.

output/

This folder contains output files that are generated when the `txtdemo-load-process.xProcessFlow` is executed.

Configuring the output files to live in a subfolder of the application folder is convenient during project development because the files are easily accessible to be inspected and reviewed. However, in production environments, this practice may be not desirable. For production environments, the output files can be configured to direct them to a dedicated output location separate from these configuration files. This is especially true in environments with many applications.

watchdir/

This is the hot folder monitored by the File Event. For each file that is added to this folder, `txtdemo-load-process.xProcessFlow` is executed to process that file. As files are picked up, they are renamed by appending `_running` to the file name to indicate that the file is being processed by a running OpenText Output Transformation Server job.

After the job completes, the file name is renamed using a `_finished` extension and any jobs that end in failure are renamed using the `_errored` extension.

Like the output folder, this subfolder is conveniently located under the application folder for convenience during development. In production environments, it may be more suitable to configure the File Event to consume files in a separate area in the file system.

resources/

This folder contains any supporting configuration files or other resources such as images or font files that may be required to complete the Output Transformation Engine transform.

The two files stored in the folder, `txtdemo-index-convert-fontmap.d2emfct` and `txtdemo-index-convert.d2efontenv`, are Output Transformation Engine configuration files with font mapping resources used by the Output Transformation Engine project. These files may need configuration when transforming AFP, Metacode, or PCL print streams into PDF. For most text and PDF-based inputs, these files can use default values and do not require additional configuration during project design.

This folder also can hold the Get File component configuration (`txtdemo-get-test-file.xGetFile`) that is used by the `txtdemo-load-process-test.xProcessFlow`.

3.1.1 Creating the configuration file in sequence

This section provides an overview of creating each configuration file in sequence, with each step building upon the previous step.

The following outlines the steps to build an Output Transformation for AppEnhancer application from scratch. As users become more experienced in creating these applications, they can streamline this process or clone existing applications to speed up development.

1. Create the Output Transformation Engine project to read sample documents from the input directory, index them, and create an index file to use for loading into AppEnhancer. This project is executed iteratively and independently during project design until the index structure and output appear correct.
2. Configure an Output Transformation Server process flow to connect the Output Transformation Engine project with an AELoader component to load the documents into AppEnhancer based on the index file created by the Output Transformation Engine project in the previous step.
3. Configure a FileEvent that will pick up input files to be processed and send them to the process flow using XDoc mappings and job variables.
4. Adjust the Output Transformation Server process flow and Output Transformation Engine project created in the previous step to use the XDoc mappings and job variables defined in the FileEvent instead of exact file names. This allows the process flows to run more dynamically.
5. Create a process flow to use for testing the process without the FileEvent. After configuring the Output Transformation Engine project to use the FileEvent's

XDoc mappings and job variables, the process flow can no longer be run in test mode. To address this, you can create a simple process flow that maps the input sample file(s) to the same XDoc mapping and job variables before calling the main process flow.

3.1.2 Creating an Output Transformation Engine project to index and transform the input file

Output Transformation Engine is a core component in OpenText Output Transformation Server that handles document parsing, indexing, and conversion tasks. Like Output Transformation Server, it provides many components that can be configured to meet a variety of document related use cases beyond those related to AppEnhancer.

In the context of Output Transformation for AppEnhancer, the Output Transformation Engine project typically performs these functions:

- Parsing an input file in any of the supported formats:
 - Printstreams, such as AFP, Metacode, or PCL
 - Text based ASCII or Linedata files (EBCDIC)
 - PDF documents
- Extracting indexing information from the document using user defined fields known as XFT templates.
- Document Breaking (splitting the input file up into individual logical documents).
- Document editing or modification (for example, adding watermarks or removing filler pages)
- If required, converting the document into a format supported by AppEnhancer (PDF, ASCII, or UTF-8 text files)
- Producing an index file containing page ranges and index fields for all sub-documents to be used by the OpenText Output Transformation Server AELoader component to load the content into AppEnhancer

Typically, creating the complete Output Transformation Engine project will be the most time-consuming step in building the solution.

This step must be completed first to complete the full OpenText Output Transformation Server Process Flow and FileEvent configuration. The Process Flow must call the Output Transformation Engine component before the AELoader component.

3.1.3 Creating a new Output Transformation Engine project using the New Component Wizard

To create a new Output Transformation Engine project using the New Component Wizard, in Output Transformation Designer, click **File > Menu** to open the New Component Wizard. Then on the **Select a File Type** screen, select **Processes > Transform > Output Transformation Project**.

The subsequent sections describe the remaining screens in the wizard.

3.1.3.1 Select Project Name and Location

Enter a **project name** of your choice. In traditional Output Transformation Engine projects that transform a file from one format to another it is common to use the format:

`<app-name>-<input2output>.d2epro`

where

`<app-name>` is the name of the application.

`<input2output>` is the input and output file formats.

For example:

`txtdemo-afp2pdf.d2epro`

In Output Transformation for AppEnhancer implementations, a suitable format may also be `<app-name>-index-transform.d2epro` so that the file name describes the role that this project file plays in the entire application: indexing and optionally transforming the data.

3.1.3.2 Select MFCT File

The MFCT file is a configuration file used to control Font Mappings, Color Mappings, and Encoding Translation tables.

Typically for text based input files (ASCII / UTF-8 or Line Data) and PDF input files, the default settings will be used and no additional configuration is required in the MFCT file.

However, the MFCT file usually requires some additional configuration when the input files are in printstream formats, such as AFP, Metacode, or PCL, and are being converted to PDF to load into AppEnhancer. For these use cases, see Output Transformation Engine User Guide for more information on how to configure your MFCT.

For our purposes, click **Create a new MFCT file based on the default template** to create a default MFCT and then select the **Use project name as MFCT file name** check box.

3.1.3.3 Select Parser, Components and Generators

On this screen, you must select the components required for your project.

The following table can be used as a guide to select the correct components:

Table 3-1: Required Output Transformation Engine components for storing in AppEnhancer

Input format	Format to store in AppEnhancer	Required Output Transformation Engine components
Text format (ASCII / UTF-8 or Line Data)	ASCII / UTF-8	<ul style="list-style-type: none">• Line Data Parser• XFT Field Technology• Indexer• Index Writer• (No generator required)
Text format (ASCII / UTF-8 or Line Data)	PDF	<ul style="list-style-type: none">• Line Data Parser• XFT Field Technology• Indexer• Index Writer• PDF Generator
PDF	PDF (original unmodified input file)	<ul style="list-style-type: none">• PDF Parser• XFT Field Technology• Indexer• Index Writer• (No generator required)
PDF	PDF (modified, e.g. filler pages removed)	<ul style="list-style-type: none">• PDF Parser• XFT Field Technology• Indexer• Index Writer• PDF Generator
AFP	PDF	<ul style="list-style-type: none">• AFP Parser• XFT Field Technology• Indexer• Index Writer• PDF Generator

Metacode	PDF	<ul style="list-style-type: none">• Metacode Parser• XFT Field Technology• Indexer• Index Writer• PDF Generator
PCL	PDF	<ul style="list-style-type: none">• PCL Parser• XFT Field Technology• Indexer• Index Writer• PDF Generator

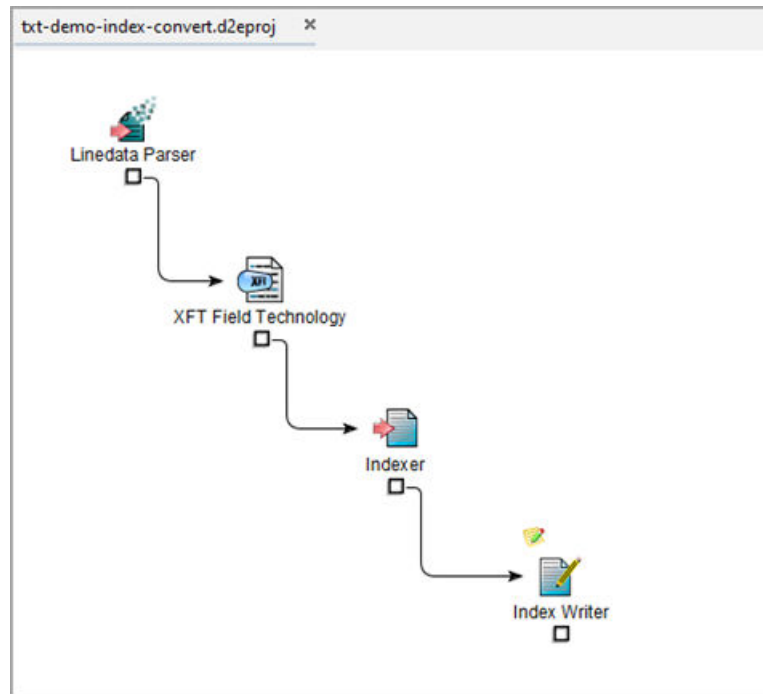
When you are finished making your selections, click **Finish** to create the Output Transformation Engine project.

The previous table includes the most commonly used components in OT for AppEnhancer projects. Be aware that in some use cases additional components may be needed in the Output Transformation Engine project to solve additional business requirements. For example:

- Script component. Allows users to write custom JavaScript logic to solve more complex or conditional processing of index fields.
- Skip Page component. Remove unwanted filler pages in a Printstream2PDF or a PDF2PDF transformation.

These components will most commonly occur in between the Indexer and Index Writer components in the Output Transformation Engine project flow.

The following is an example of a basic Output Transformation Engine project structure to index ASCII text or Line Data:



3.1.4 Configuring the Output Transformation Engine project input and output file locations

In the full end-to-end process of a typical Output Transformation for AppEnhancer application, the input data file is picked up by a FileEvent and sent into a Process Flow that hands the data to an Output Transformation Engine project for indexing and transformation (when applicable).

Afterwards, the index file produced by the Output Transformation Engine project, as well as any newly created output PDF or text files (if applicable), are provided back to the Process Flow so that the AELoader component can complete the load into AppEnhancer.

During the project design stage, it is most convenient to run the Output Transformation Engine project in a standalone, isolated manner without running the additional FileEvent, Process Flow, and AELoader components. Once the Output Transformation Engine project is producing the desired output and index file structure, it can be integrated with the other components.

In Output Transformation Engine, most input or output file settings use a parameter type called a "FileDef". FileDefs are quite flexible to meet a variety of use cases and even support the ability to use custom user-written plug-ins called "I/O Handlers" to override the inputs and outputs.

At this stage of project design, only the basic features of FileDefs will be used to map to simple filename patterns. At subsequent stages, you can adjust these FileDefs to

use job variables to integrate the Output Transformation Engine project into a larger process flow.

3.1.4.1 Configuring the Parser **FdInput** parameter

In an Output Transformation Engine project, every Parser component will have a filedef parameter named **FdInput** that defines the location of the input file as well as any special IO Directive instructions that may be used to describe the formatting of the file.

In the sample `txtdemo` project, configure **FdInput** to use the following value:

```
{d2eAppPath}input/test-input.txt
```

The `{d2eAppPath}` variable is dynamically resolved to be the parent folder containing the `.d2epro` file.

3.1.4.2 Configuring the Linedata parser filedefs

The Line Data Parser has two filedefs that are used:

- **FdInput**. The input text ASCII or UTF-8 or Linedata file.
- **FdAsciiOutput**. The normalized ASCII or UTF-8 version of the input file written out by the Linedata Parser that will be loaded into AppEnhancer.

In the `txtdemo` project, configure the **FdAsciiOutput** to use the following location:

```
{d2eAppPath}output/{d2eJobId}.txt
```

Using this location value will write the normalized text file into the `output` subfolder. It uses an additional built-in job variable `{d2eJobId}` that will be unique for each job that is executed.



Note: Although uncommon, in some cases the Output Transformation Engine project may be configured to convert the Linedata input file into PDF for loading into AppEnhancer. In this situation, **FdAsciiOutput** does not need to be configured.

Also, when using **FdAsciiOutput**, it is recommended to configure the **OutputEncoding** parameter to use **UTF8**. This ensures that the file produced uses UTF-8 encoding to ensure all languages are supported.

3.1.4.3 Configuring the PDF Generator output file


If your project contains a PDF Generator component to create new PDF files to load into AppEnhancer, configure the PDF Generator **FdOutput** parameter location to use the following value:

```
{d2eAppPath}output/{d2eJobId}-{0}.pdf
```

This value uses the {0} variable placeholder, which will be filled in automatically with the document number. The PDF Generator may produce more than one PDF file in some situations, such as when an input file exceeds the AppEnhancer limit of 32767 pages, and it must be split into multiple output PDF files.

3.1.4.4 Executing the project

Once the required filedefs have been configured in the Output Transformation

Engine project, execute the project by clicking **Execute**, , in Output Transformation Designer.

If your project is configured correctly, the process should run and finish successfully. You can review your output files in the **Results** tab.

If there are any errors, view them in the **Log** tab to troubleshoot and resolve the issues.

When the project is completing successfully, you can continue to the next steps.

3.1.5 Configuring the Output Transformation Engine project to extract fields for indexing

The XFT Field Technology component is a core component in Output Transformation Engine that is used to extract page content into “fields”. It provides many features that can be used for many different use cases, such as extracting simple text fields, complete data table structures, or identifying and extracting graphic images.

In the context of OT for AppEnhancer projects, only some of the fundamental features of XFT are required. Many of the more advanced features are rarely used in these types of projects.

These fundamental XFT features are:

- **XFT Locators.** Searching for trigger values using static search strings or regular expressions
- **XFT Text Fields.** Extracting text strings from the page content that will be stored as an index field in AppEnhancer.

3.1.5.1 Setting up an XFT Document

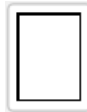
You can set up XFT documents using the XFT editor in the **XFT Field Technology** component. In an Output Transformation Engine project flow, double click the XFT Field Technology component or right-click the component and select **Open**.

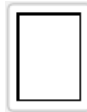
You can configure your document in the XFT Design pane. The data that has been captured for the currently displayed page using the current parameters are automatically shown in the **XFT Results** pane.

For more information about the XFT Window, see *OpenText Embedded Output Transformation Engine - User Guide (VDTOTS-H-UTE)*.

You must create a document as a first step to setting up the XFT Field Technology component. To create an XFT Document:

1. In the **XFT Window**, click **Add New > Add Document**.
2. On the **General** tab in the **XFT Document** dialog box, enter a custom **Name** for your document.
When you are finished, click **OK**.
3. In the **Auto Detect Sections** dialog box, you must indicate whether you want to automatically detect the sections in your document. Click **Yes**.
4. In the **Detection Options** dialog box, you must indicate the number of sections



you want **Split To Try To Get**. Click  to create a single Section that will cover the entire page area.

When you are finished, click **OK**.

For the purpose of loading documents into AppEnhancer, most projects only need to define a single XFT Document that contains a single XFT Section. More complex use cases of XFT may require the defining of multiple Documents with multiple Sections. You can explore some of the included sample projects, such as SFWealth or BestBank, to see such examples.

3.1.5.2 Turning off DOM properties on XFT Documents

XFT also provides a feature to build a DOM (Document Object Model) from the XFT results. This feature is commonly used in creating Accessible PDF documents and other use cases where the entire structure of the document is being mapped and extracted by XFT.

For OT for AppEnhancer and other projects that primarily focus on indexing for loading into an archive system, the DOM creation is not required. To simplify the project and reduce processing overhead, the DOM properties feature should be turned off.

To turn off DOM properties, open your XFT Document and on the **General** tab of the **XFT Document** dialog box, clear the **Enable DOM Properties** check box.

For more information about turning off DOM properties, see [“Optimizing performance in an Indexing project” on page 59](#).

3.1.5.3 Defining XFT Text fields to extract data

Once an XFT Section has been created, you can define fields within the Section to capture the text you want.

To define an XFT Text field:

1. In the XFT Design pane, right-click a Section in your Document and from the context menu that appears, click **Add Text Field**.
2. When you move the cursor over the Preview pane, the cursor changes from an arrow to a crosshair. With the crosshair cursor, click and drag to draw a bounding box around the data you want to capture. The area that you want to mark as a text field must be within the boundaries of a previously defined section.

For example, the following figure shows drawing a box around the text 14 because that is the value of the Clerk ID you want to capture:

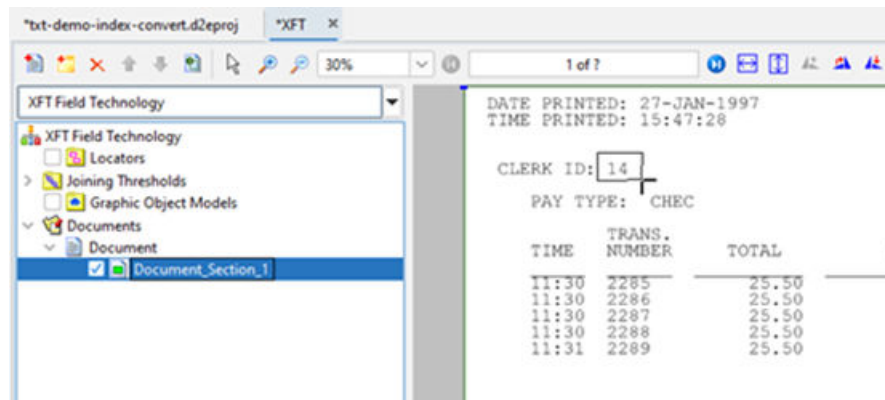


Figure 3-1: XFT window showing a bounding box around the Clerk ID value to capture

3. On the **General** tab in the **XFT Text Field Properties** dialog box, enter a custom **Name** for your document.

For our example, you will use **CLERK ID** because that is the exact name defined in AppEnhancer for the corresponding index field so you want that same name to be used in the index file produced by Output Transformation Engine.

When you are finished, click **OK**.

After the text field has been created, you can verify the extraction by expanding the **XFT Results** tree to see that the value 14 has been extracted. The following figure shows the extracted 14 value:

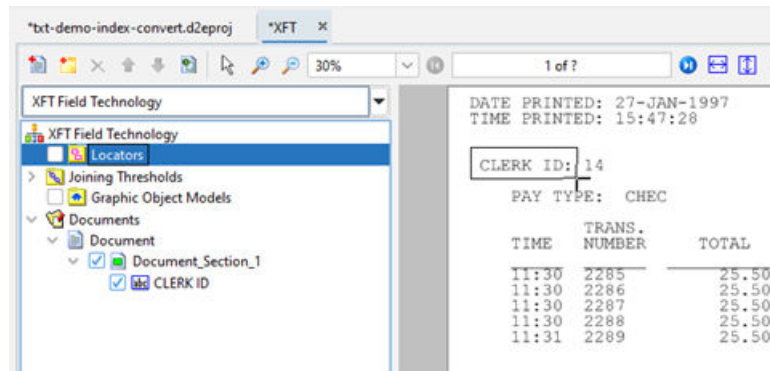


Figure 3-3: XFT window showing bounding box around the CLERK ID text for XFT Locator creation


3. On the **General** tab in the **XFT Text Field Properties** dialog box, enter a custom **Name** for your document.

The name should be descriptive of the text field, such as “Clerk ID Locator”.

4. You can also set a literal text string or regular expression using the options in the **Search Pattern** area. In this example, type “Clerk ID:” in the **Literal Text** box.

When you are finished, click **OK**.

With the set up for the Clerk ID Locator complete, you are ready to associate the locator with the text field. To associate the Clerk ID Locator with the text field:

1. In the **Design** pane in the XFT window, expand the node for your XFT Document and double-click the CLERK ID text field.
2. On the **General** tab in the **XFT Text Field Properties** dialog box, select the **Enable Identification Processing** check box to make the **Identification** tab available.
3. On the **Identification** tab, the parameters shown define the rules for when the field exists and should be extracted. To add the rule that the text field exists when the Clerk ID Locator is found, in the **Identifiers** area, click **Add Identification**, .
4. In the **Identifiers** table, select **Clerk ID** in the **Locator Name** column and then select the **Required** check box.

When you are finished, click **OK**.

To verify your configuration in the XFT Results pane, you should see the CLERK ID text field showing that it uses the Clerk ID Locator as its Start Locator.

As a further test, you can click on the **Clerk ID** Locator under the **Locators** node and then reposition the bounding box so that it no longer covers the “CLERK ID:” text on

the page. You will see that the XFT Results no longer show the CLERK ID field because the locator search condition was not satisfied.

3.1.5.4 Defining XFT fields that use dynamic positioning

For the purposes of this sample case, you also want to capture each Transaction number in the “TRANS. NUMBER” column, and each Violation number in the VIOLATION NUMBER column. These fields occur multiple times throughout the document and in different areas of the page.

To extract all of these field instances on the page, you must use more advanced features of the text field known as Dynamic Positioning.

To start, you must define a locator for the Transaction Numbers. This locator will be similar to the one previously created in this sample use case, but the search area for the locator will be drawn to include the entire area that could possibly hold a transaction number. A regular expression Search Pattern of “\d\d\d\d” that matches any four digits will also be used.

Then you will create an XFT field called “TRANSACTION NO.” that will use the new locator as its required start identifier, like you did with the Clerk ID field.

In the following figure, the XFT Results show that the transaction numbers in the “TRANSACTION NO.” field have been captured into a single Field Result.

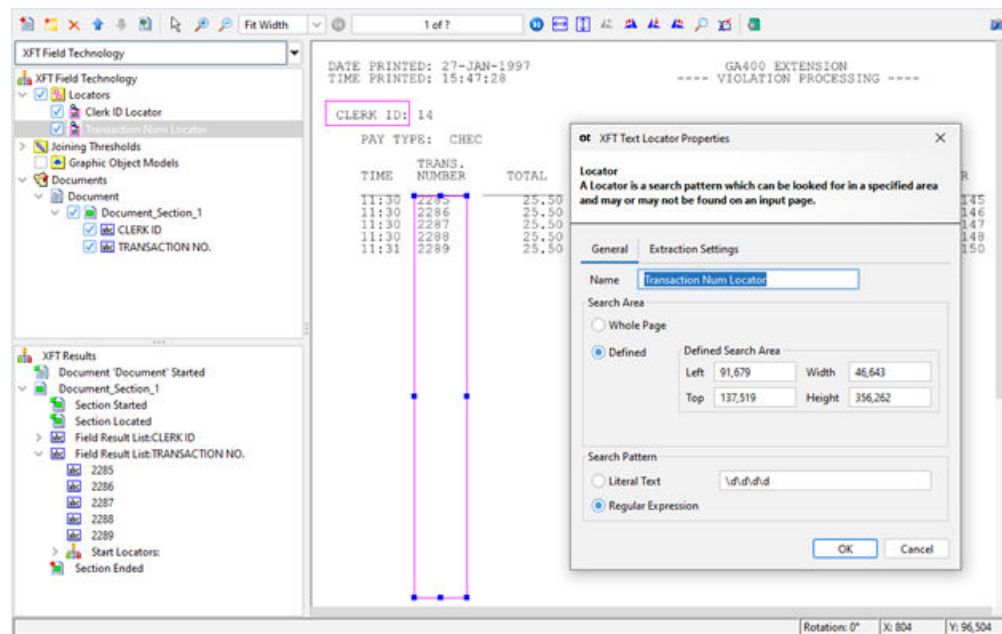


Figure 3-4: XFT Results with captured values in a single Field Result

All of the values were captured in a single Field Result because the bounding box for the field is static and covers the area that contains all of the transaction numbers.


However, this is not the desired outcome. Instead, you want to capture the values as five separate Field Results, each having one transaction number each. To achieve this goal, you must use dynamic positioning.

To set up an XFT field with dynamic positioning:

1. In the **Design** pane in the XFT window, expand the node for your XFT Document and double-click the **TRANSACTION NO.** text field.
2. On the **General** tab in the **XFT Text Field Properties** dialog box, select the **Enable Dynamic Positioning** check box to make the **Positioning** tab available.
3. On the **Positioning** tab, the options allow you to override the four different components of the field bounding box: Top, Left, Bottom (Height), Right (Width).

For the TRANSACTION NO. field, the Left and Right positions will not change from the bounding box you drew for the column that holds the transaction numbers. However, to capture each line of transaction numbers independently, you need to override the Top and Bottom locations.

Select the **Top** and **Bottom (Height)** check boxes to turn on their location overrides.

4. Click **Add**, , to add a Locator to the table and then set the **Locator Name** to be the locator you created.

When you are finished, click **OK**.

Next, you must determine the values to use as the Offset Top and Offset Bottom columns. To determine the Offset Top and Offset Bottom values:

- In the XFT Results pane, expand the nodes for **TRANSACTION NO. field > Start Locators** and then click **Transaction Num Locator**.

The area on the page where the locator was found is highlighted. Notice that when you move the cursor around the page, you can see the page coordinates in the XFT window Status bar.

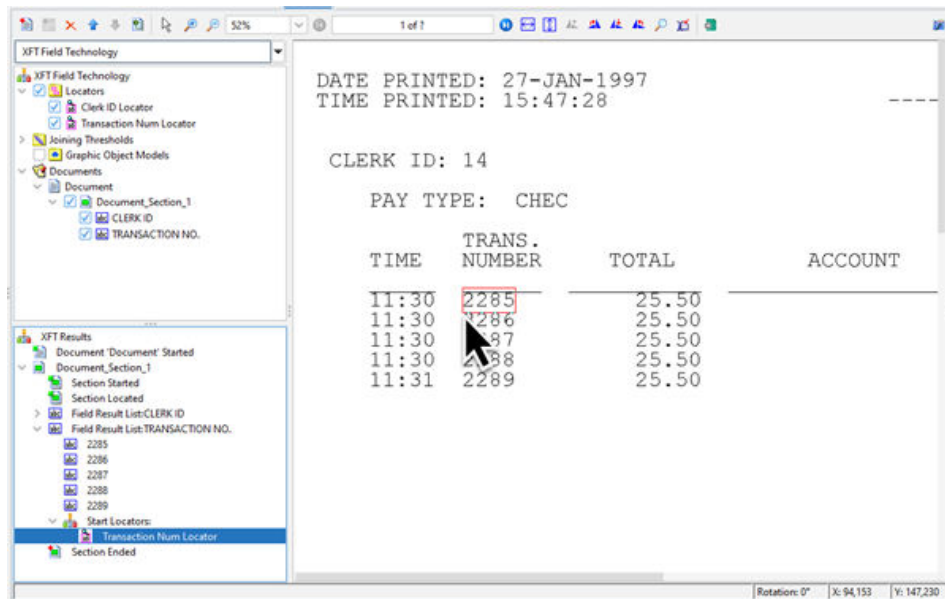


Figure 3-5: Start Locator field node selected and related locator highlighted

In XFT, text is captured by a locator or text field position based on the baseline X and Y position of the text in question. If the baseline position is found inside the field or locator's defined area, it will be captured.

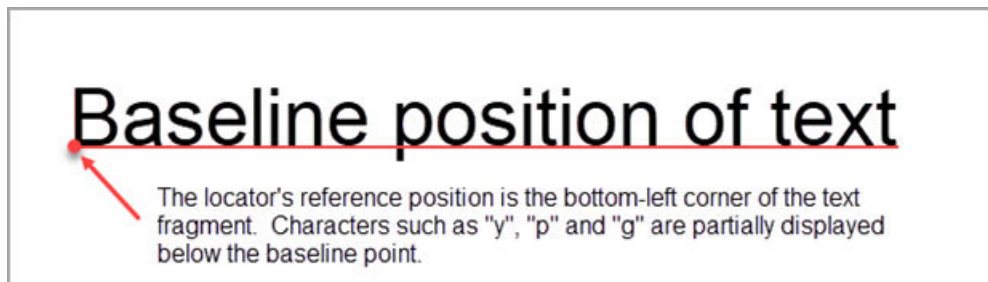


Figure 3-6: Baseline position of text explanation

For our use case, you can specify the **Offset Top** as **-9000** (a minus value will shift up from the baseline reference point), and the **Offset Bottom** as **1000** to create a new bounding box that will capture the transaction number correctly. Using these values, XFT will find all five Transaction Num Locators (one for each 4 digit number matching our regex pattern in the search area).

For each found locator, an instance of the TRANSACTION NO. text field will also be found. Your XFT Results pane should appear similar to the following figure:

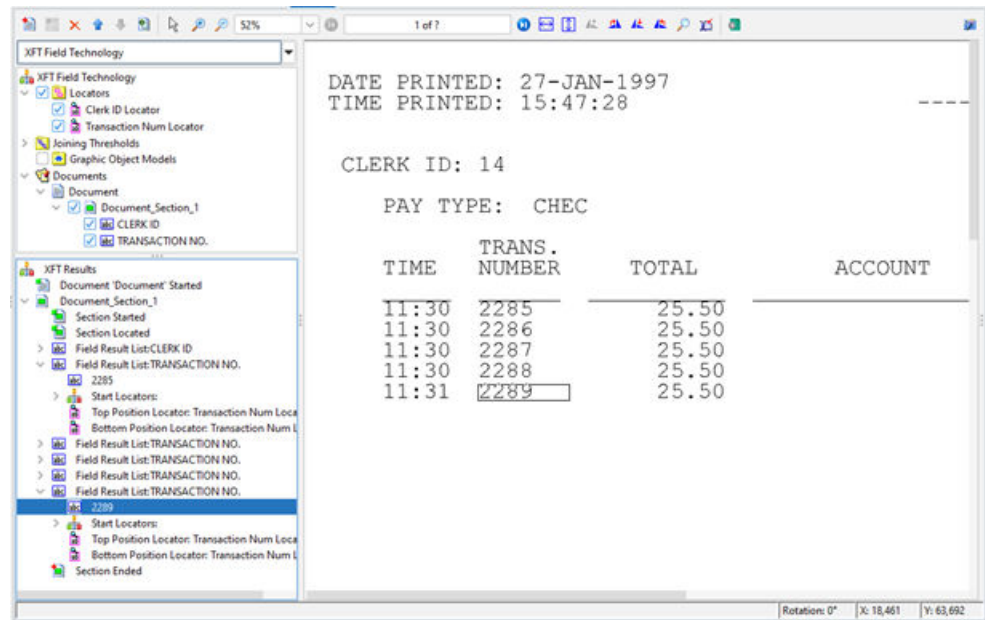


Figure 3-7: XFT Results pane showing five separate Transaction Num locators

A similar strategy can be used to find the Violation Numbers. However, you will also remove the leading zeros so that instead of 00000000000006004145, the application will only extract 6004145.

By default, Output Transformation Engine will parse the input data into individual words, using space characters as delimiters. To view the results of the parsing, in the Development window, right-click on the Parser component in your project flow and from the context menu that appears, click **Page View**. Then on the **Page View** tab, in the **Summary** pane click **Text** to view all of the text elements on the current page. An example of the parsing results is shown in the following figure:

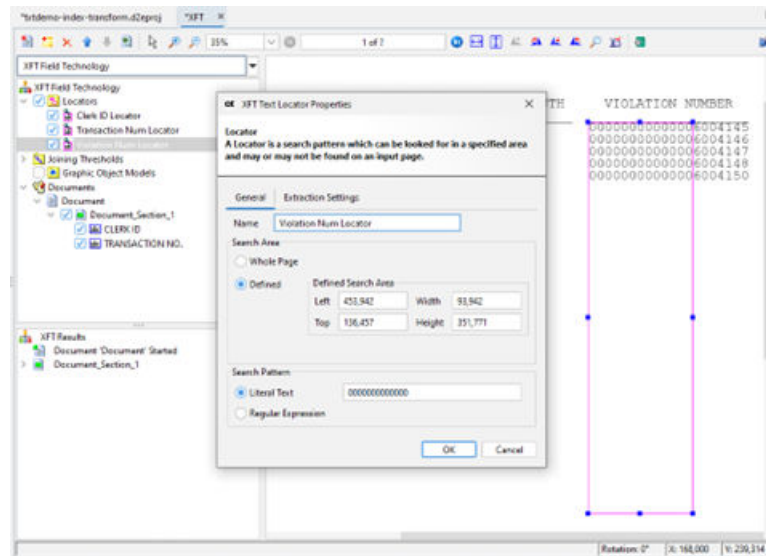


Figure 3-9: XFT Text Locator properties with XIF options configured to split smaller text fragments

You can then create an XFT Text Field called “VIOLATION NO.” that follows the same strategy for Identification Processing and Dynamic Positioning that was used for the “TRANSACTION NO.” text field. However, in this case you will use the Violation Num Locator instead of the Transaction Num Locator.

Once the Violation Num Locator is created and you return to the XFT Results pane, you will see that for the “TRANSACTION NO.” and “VIOLATION NO.” fields there are five results for each.

multiple times for the same page in the document. You want each combination to appear in AppEnhancer as a unique search result, but each result corresponds to the same page/document.

Setting up the Document Breaking tab

On the **Document Breaking** tab, you will define how the input file should be split into individual logical documents.

There are multiple types of criteria you can use to set up document breaking. For example, if every page is a unique document, you can simply use a Page Count of 1. As a project designer, you should review the structure of your documents and identify the business rule you need to apply to identify your document breaks. A common example is “whenever the text ‘Page 1 of’ appears in the top right corner of the page, it indicates that this is the start of a new document”. In that case you could define an XFT text field to capture that text and use the **XFT field equals** option with a value of **Page 1 of**.

In our sample case, every new document has a unique Clerk ID therefore, you can select the **XFT field change** check box and use it with **Field name** of **CLERK ID**.

Setting up the XFT Index tab

On the **XFT Index** tab, click **Add all** to move all available fields from **Available Fields** to **Selected Fields** to add them to the index file.

Indexing original document names

The index file that is created based on the indexer’s configuration will include the file name of the data file to load into AppEnhancer as well as the associated metadata for each logical document.

When the Indexer’s **Index Original Document** option is enabled, the input file that is loaded into AppEnhancer will be the input file, rather than a new output file created by a PDF Generator component. When this option is turned on, the **Original Document Indexing** tab is activated.

In the Indexer’s wizard, the Original Document Indexing tab is activated whenever the **Index Original Document** check box is enabled. On this tab, you must enter a value in the **Document name** box to indicate to the Indexer where the input file is located.

Linedata Parser input files

When loading text documents into AppEnhancer using the Linedata Parser, on the **Original Document Indexing** tab you must set the **Document name** box to the same value as the location used in the Linedata Parser’s **FdAsciiOutput** parameter.

For example, if you used this value for the **FdAsciiOutput** location:

```
{d2eAppPath}output/{d2eJobId}.txt
```

Then you must use the same value for the **Document name** parameter.

PDF Parser input files

In this case, you can use the special job variables that are automatically populated based on the input file that the PDF Parser uses as runtime. On the **Original Document Indexing** tab, set the **Document name** parameter to the following value:

```
{d2eInputFilePath}{d2eInputFileName}{d2eInputFileExtension}
```

3.1.7 Configuring the Output Transformation Engine project Index Writer component

The role of the Index Writer component is to create a physical index file based on the index structure created by the Indexer component. The Index Writer can create index files in several possible formats. For Output Transformation for AppEnhancer projects, the **Generic Index Format** is used.

To configure the Index Writer, in the Development window in Output Transformation Designer, double-click on the **Index Writer** component in your project flow to display its properties. Once the wizard is open, in the **Index writer type** list, click **Generic**.

The **Generic Options** in the Index Writer also contain an optional string parameter, **BatchFormName**, that can be set to the name of a form that should be assigned to all pages in the document when loading text data into AppEnhancer.

It is also recommended to configure the **JavaCodepageName** parameter to use the value of **UTF-8**. This will ensure that any extended Unicode characters such as CJK or extended European characters are preserved correctly.

3.1.8 Validating Output Transformation Engine reports

It is highly recommended you do not wait until you load into AppEnhancer to verify the structure.

Instead, it is recommended to run your Output Transformation Engine project in isolation and inspect the Index File. You should also verify the pages ranges and index field values along with inspecting the Job Report.

Once you are comfortable with this level of inspection, continue on to loading the data into AppEnhancer.

3.2 Configuring the OpenText Output Transformation Server process flow

A Process Flow is called to process each input file picked up by a FileEvent.

Process flows provide a lot of flexibility in what they can be configured to do, but in the most basic case for AppEnhancer projects, they may simply consist of an Output Transformation Engine project and AppEnhancer Loader component.

3.2.1 Configuring an AppEnhancer Loader component

To create a new AppEnhancer Loader component:

1. Go to **File > New**.
2. In the **New Component Wizard**, go to **Connectors > AppEnhancer Loader**.

3.2.2 AppEnhancer Loader component prerequisites

To set up AppEnhancer Loader to connect with an AppEnhancer instance, you must have the following additional elements ready:

- An OpenText Output Transformation Server license with permissions enabled for the AppEnhancer component
- A fully installed and configured AppEnhancer instance, which must contain the base product as well as the following additional components:
 - AppEnhancer Administrator, AppEnhancer Web Access Server, and AppEnhancer REST Services. (For more information on installing these AppEnhancer products, see *OpenText™ AppEnhancer Installation Guide*.)



Note: For the latest version compatibility information, refer to the Product Compatibility Matrix in the Release Notes.

3.2.3 Configuring the AppEnhancer Loader component

The AppEnhancer Loader component must be configured with your server info, user credentials, and information about your load package before you can connect with your AppEnhancer instance.

The following parameters must have values added to them to facilitate the connection and conduct loading to the AppEnhancer server.

- **AuthType.** Specifies the authentication type to use for the REST loader connection.
- **Username.** Specifies the username for the AppEnhancer data source. The method of authenticating your credentials depends on your AppEnhancer data source security configuration.

- **Password.** Specifies the password for the associated user to the AppEnhancer data source. The method of authenticating your credentials depends on your AppEnhancer data source security configuration.
- **HostURL.** Indicates the host URL for the server to connect to. AppEnhancer REST services must be running on the server. If you installed AppEnhancer REST services on the default path, then `http://<hostname>/AppEnhancerREST` is the default host URL. SSL protocols are supported if the JVM and necessary security certificates are properly configured.
- **DatasourceName.** Denotes the name of the data source to load onto the server. When the Username, Password, and HostURL parameters are filled in correctly, clicking the ellipsis button retrieves a list of available data sources to select from.
- **ApplicationName.** Denotes the name of the application to load onto the server. When the Username, Password, and HostURL parameters are filled in correctly, clicking the ellipsis button retrieves a list of available Applications from the data source to select from.
- **IndexFile.** Designates the file path and name of the index file for the load package. Note that this is not the input file itself, however, the index file contains information about the input file. The index file format should be in the Generic index format created by Output Transformation Engine.
- **AuditFileName.** Indicates the full file path and name of the audit file to create. If the file already exists, it will be overwritten. The audit file logs all load operations and can be used for load troubleshooting or load reconciliation.

The following parameters related to document handling options, batch creation, and date formats are optional and do not need to be set if they do not meet your project requirements.

- **DocumentOptions.** Contains options related to document handling during loading.
 - **OnError.** Designates the behavior when an error is encountered while loading documents. The following options are available:
 - **FAIL.** Specifies that the loading process stops. This is the default behavior.
 - **CONTINUE.** Specifies that the loading process continues and any document related failures are skipped. Warning messages are still logged for failures.
 - **OnDuplicateIndexEntry.** Designates the behavior when the document being loaded already exists in the AppEnhancer instance.
 - **DO_NOTHING.** Indicates that AppEnhancer returns an error and does not load the document. During loading, no extra flags are set to handle the duplicate index entries. This is the default behavior.
 - **APPEND_EXISTING.** Indicates that pages with duplicate indexes are appended to the existing document in the target application. During

loading, the MergeDuplicateEntry flag is set to handle the duplicate index entries.

- **NEW_DOCUMENT**. Indicates that a new document with a duplicate index is created in the target application. During loading, the IgnoreDuplicateIndex flag is set to handle the duplicate index entries.
- **KeyReferenceUpdate**. When enabled, designates that new index values are added to the key reference table if the key value is not found, and data reference fields are updated if the key value is found. If disabled, new index values are added to the key reference table only if the key value is not found. By default, this parameter is disabled.
- **IgnoreDIsViolation**. Indicates that document level security for the selected application is bypassed and loading is allowed to continue when the user and index information do not satisfy the application's document level security requirements.
- **COLDFormName**. Designates the name of a form overlay to associate with the batch AELoader upload. The overlay name is also associated with all pages in a document. This value must be a valid overlay name that already exists in the AE_FORMS application on the AppEnhancer server. When the Username, Password, HostURL, and DataSourceName parameters are filled in correctly, clicking the ellipses button retrieves a list of available form overlays from the data source to select from.
- **SubmitFullText**. Indicates whether documents created from the uploaded batch are sent to the Full Text Queue. By default, this option is turned off.
- **BatchOptions**. Contains options related to batch creation.
 - **BatchIDProperty**. Denotes the name of the AppEnhancer Application field to write the generated OpenText Output Transformation Server load ID. You must have a field with this name predefined in the Application before using this parameter. This property is used for auditing and load reconciliation.

If this parameter is left blank, no load ID is saved with the document index record, which may cause confusion. The source of the uncertainty is due to AppEnhancer being unable to link loaded documents to a given OpenText Output Transformation Server load. However, the reconciliation can also be performed using the AELoader audit log since it records the document IDs that were created for the load process.
 - **BatchIDFormat**. Designates the format of the load ID generated by OpenText Output Transformation Server. By default, a universally unique identifier is generated by the AELoader and placed in the job variable AEBatchUuid. When this value is not set, the default format `0tsLoad: ${AEBatchUuid}` is used.
 - **BatchSizeMode**. Indicates how input files (LineData only) are split. You can choose from AUTO or CUSTOM. **AUTO** denotes that when the number of pages in the input file (based on the index file page count) is greater than the AppEnhancer page limit of 32,767 pages, then the file is split into upload batches of approximately equal pages. **CUSTOM** allows you to specify the

number of pages to include in an upload batch when used in conjunction with the BatchCustomSize property.

- **BatchCustomSize.** Specifies the maximum number of pages included in an upload batch. When setting this value, you must set BatchSizeMode to CUSTOM otherwise this value is ignored.
- **DateFormat.** Contains options related to the conversion of date fields. When these parameters are used, any fields with the Date data type are converted using the properties from SourceDateFormat and TargetDateFormat. Be aware that when using the date conversion feature, you must set values for both SourceDateFormat and TargetDateFormat. If these parameters are blank, then all index values are passed through as is.
 - **SourceDateFormat.** Designates the format of input date fields. This parameter uses Java SimpleDateFormat syntax, which is also used to interpret date fields.
 - **TargetDateFormat.** Designates the target format for date fields when loading into AppEnhancer. All date index values are converted to this format when documents are loaded.
- **ReliableTransfer.** Contains options related to the reliable transfer mechanism that automatically restarts or resumes a transfer if an error occurs. For more information, see *OpenText Output Transformation Server - User Guide (VDTOTS-H-UGD)*.

3.2.4 Reading the AppEnhancer Loader audit logs

The AppEnhancer Loader audit file logs all load operations, which can be useful for troubleshooting or load reconciliation. A single record is added to the log file for each REST call that occurs and a status code representing the last known state of the particular REST call is shown so you can easily determine the current standing.

The audit file has three distinct sections. The first section displays general information about the application. The next section displays application and field attributes while the final section displays the audit records from load operations.

Application and field attributes

The application attributes are defined as follows:

Flag	Description
KR	Key Reference
MI	Multi-Index
DLS	Document Level Security

The field attributes are defined as follows:

Flag	Description
R	Required
S	Search
DLS	Document Level Security
PUK	Part of Unique Key
DDE	Dual Data Entry
KR	Key Reference
DR	Data Reference
AI	Auto Index
VM	Validation Mask
TS	Time Stamp

Audit operation records

Records are displayed in the following format:

```
<Date>,<Hash>,<IndexId>,<Code>,<AEdocId>,<BatchId>,<AEdocPageCount>,<TextInfoField>
```

The record fields are defined as follows:

- **<Date>**. Specifies the current date.
- **<Hash>**. Used to help validate the integrity of a record for the <IndexId>, <Code>, and <AEdocId> fields when handling reload/unload scenarios.
- **<IndexId>**. Specifies the index ID generated by the AELoader to uniquely identify an uploaded document page.
- **<Code>**. Designates the status code. A valid event-related code should always be displayed; NUL is an indication of a code error. See the following descriptions for the meaning of each code:
 - **NUL**. Null. When this is shown, it indicates an error in the record.
 - **UBS**. Upload_Batch_Start.
 - **UBE**. Upload_Batch_End. At the end of an UBE record, the Pages= attribute indicates the number of pages processed after an upload. Checking this value can help verify that AppEnhancer is handling the same number of pages as the Output Transformation Engine index file.
 - **CDS**. Create_Document_Start. If KeyReferenceUpdate is disabled and reference data fields are removed from the load request, a dash (-) symbol is used in place of the CDS record to indicate that it has been omitted.
 - **CDE**. Create_Document_End. If KeyReferenceUpdate is disabled and reference data fields are removed from the load request, a dash (-) symbol is used in place of the CDE record to indicate that it has been omitted.

- **KRU**. Key_Reference_Update. If KeyReferenceUpdate is enabled and existing key and reference data is found in an uploaded document's index information, this code is appended to the new data reference values.
- **CDEM**. Create_Document_End_Merged. Indicates a loaded document was merged into an existing document. In the audit records, the document ID represents the ID of the merged document.
- **STR**. Started. Denotes an AELoader process was started.
- **ABR**. Aborted. Indicates that the AELoader was stopped due to a user manually aborting an AELoader process.
- **ERR**. Errored. Designates that the AELoader was stopped due to an error occurring during loading.
- **FIN**. Finished. Indicates that the AELoader stopped and finished loading with no errors.
- **SKIP**. Skipped. Denotes a document was skipped and not loaded.
- **TCON**. Test_Connection. Denotes when a connection error occurs.
- **IVAL**. Index_Validation. Designates when an error occurs during AELoader's index validation process.
- **DVAL**. Document_Load_Validation. Designates when an error occurs during AELoader's document load validation before a load retry attempt.
- **INFO**. Displays informational messages, such as Index Field information.
- **IDS**. Index_Document_Start. Indicates the starting point of an index document.
- **IDE**. Index_Document_End. Indicates the ending point of an index document.
- **CBS**. Create batch start. Only used when processing Linedata.
- **CBE**. Create batch end. Only used when processing Linedata. The page count and temporary batch file location are also displayed.
- **RIF**. Denotes that a problem was encountered while reading an input file.
- **<AEdocId>**. Denotes the doc ID assigned by AppEnhancer to the particular document. If an ID is not assigned, -1 is used.
- **<BatchId>**. Denotes the batch ID assigned by AppEnhancer to the uploaded batch file.
- **<AEdocPageCount>**. Designates the current page count of the document after a successful call.
- **<TextInfoField>**. Specifies any general information such as index fields or error messages. The field is not always populated, but events that result in a CDE status code display index information while NUL events display information related to the error.
-

3.2.5 Configuring the FileEvent

This section discusses configuration of the FileEvent component in OpenText Output Transformation Server.

3.2.5.1 Using FileEvent to locate source files for processing

In ERMX, a source specification is a set of parameters that indicate how AppXtender Reports Mgmt detects files for processing and how AppXtender Reports Mgmt handles those files when processing is complete. The key parameters are the source directory and the method of source file recognition. Supported methods of source file recognition include file name masks and INI files.

In OpenText Output Transformation Server, the direct equivalent for this is to create and configure a FileEvent event.

The FileEvent component has many configurable options, but the required parameters are:

- **DirsToScan.** Configure one or more directories that will be scanned for input files.
- **ProcessToRun.** Specifies the file path and name of the process flow that is executed for each input file that shows up in a scanned directory. This file path is relative to the base of the OpenText Output Transformation Server mount point that must reside under.
- **InputMapName.** A new XDoc is created for each new process flow that is triggered by the FileEvent. The input file that is detected by the File Event is attached to the XDoc so that other components in the process flow can read it as the process flow executes.

For more information about other FileEvent parameters, see Output Transformation Server User Guide.

In Output Transformation Designer, a File Event process flow looks like the following figure:

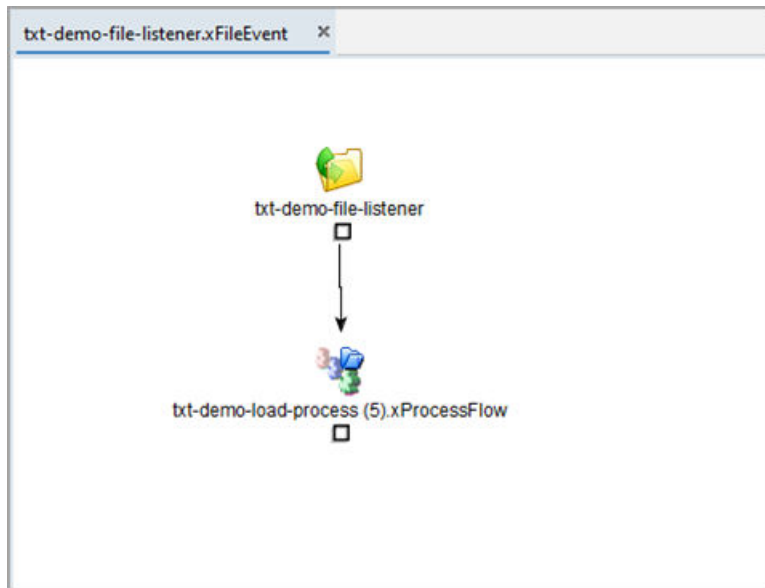


Figure 3-11: Sample File Event connected to a configured process flow

The input file picked up by the FileEvent needs to be fed into the Process Flow so that the Output Transformation Engine project can process it. In the previous steps in this guide, the parser component contained in the Output Transformation Engine project was configured to locate the input file for testing using the `FdInput` parameter. Now you want to modify that configuration so that the Output Transformation Engine parser will use the input file provided by the FileEvent.

In OpenText Output Transformation Server, there are two main methods of sharing files and data between components during the execution of a job:

1. **Job Variables.** These are `name=value` variables that get stored in the JobTicket that is passed around to all components during the lifetime of a single job.
Commonly, job variable values are Java string objects, such as file names, but they can be any type of Java Object, such as a byte array of data.
2. **XDocs.** An XDoc is an efficient, flexible wrapper for a file in the file system or contents of a file (Java `byte[]` or `char[]`). There can be many XDocs in a single job and it supports both reading existing files and writing out new files. The JobTicket for the job has mappings of Input XDocs and Output XDocs. Each mapping is assigned a configurable name such as `InputFile`, `DataToConvert`, or `FinalOutputFile`.

For the FileEvent instance in our sample, a combination of both options will be used: an XDoc will be created to read the contents of the input file picked up by the FileEvent, and the file name of that input will be stored in a Job Variable. Then the Process Flow will be configured to stop using hard coded input file names and start reading the data from the XDoc input mapping instead.

The Output Transformation Engine index writer will also be configured to use the FileEvent's Job Variable instead of the Output Transformation Engine specific `{d2eInputFileName}` variable.

3.2.5.2 Configuring the FileEvent XDoc mappings

In the FileEvent, configure the **InputMapName** parameter to be a key for the XDoc. Other components in the job will use this key to get access to the XDoc through the Job Ticket.

In this example, the input map name will be `FILE`.

3.2.5.3 Configuring the FileEvent file name job variables

The FileEvent component can populate four optional job variables with the file name of each input file processed. The parameters are:

- FullnameVariable
- PathnameVariable
- FilenameVariable
- ExtensionVariable

Users can configure the value of each variable. For this sample project, you want to use the FilenameVariable in the Output Transformation Engine project so set **FilenameVariable** to `fileName`.

The completed FileEvent parameters should look similar to the following:

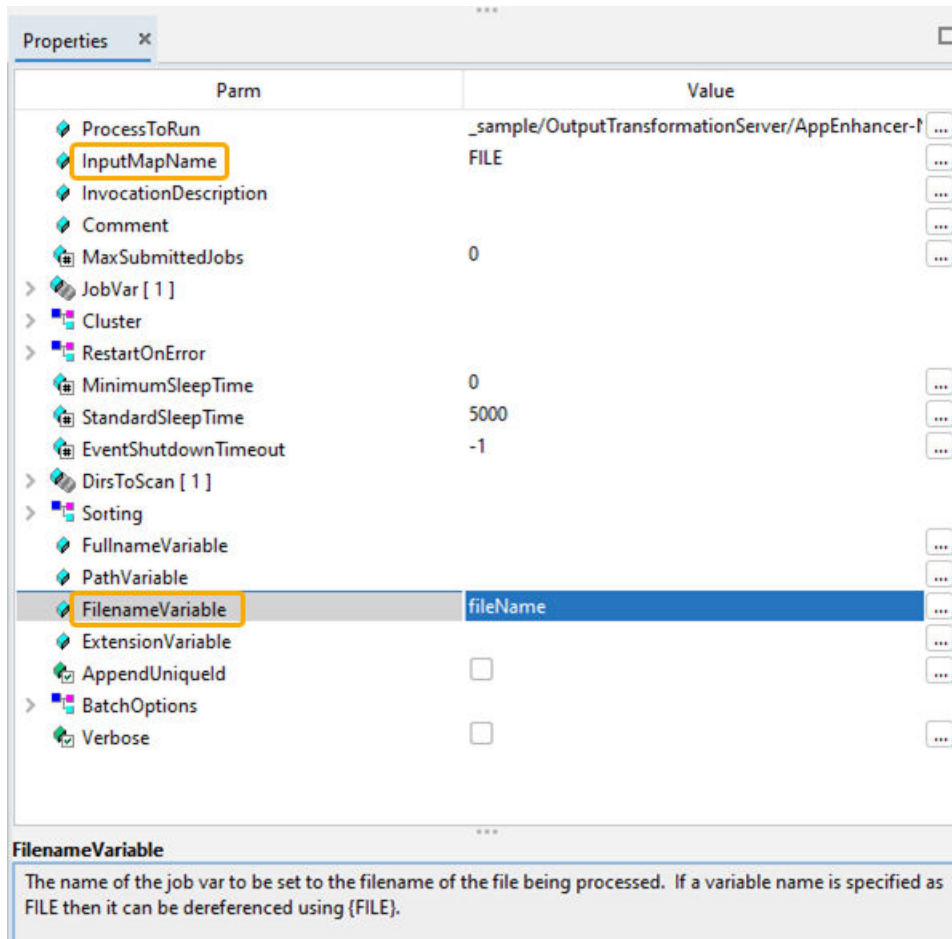


Figure 3-12: FileEvent configuration with InputMapName and FilenameVariable parameters highlighted

3.2.6 Configuring the process flow to use XDocs and job variables

This section discusses how to set up the process flow to use the previously configured XDocs and job variables.

3.2.6.1 Setting up a new job variable to define the output folder

In previous steps to setup the Output Transformation Engine project, output files were configured to use `{d2eAppPath}output/` which ensured the output files would get written to an output subfolder of the `{d2eAppPath}`, which is the directory where the `.d2epro` lives.

That works well when testing the Output Transformation Engine project in isolation, however, it could be desirable to set the location to something else when running in the larger context of the process flow.

In the `txtdemo-load-process.xProcessFlow`, add a **JobVariableSetterProcess** to the process flow by dragging the component from the palette to the Development window. Position the component at the start of the process flow, ahead of the Output Transformation Engine project.

Select the new component and configure the properties to add an entry to the **ConstantJobVariables**. Set the name of the variable to be `outputFilePath` and the value to be something of your choosing. You can use `${config.path}/output` to put the files in an “output” subfolder based on the process flow’s parent directory.

The updated process flow should appear similar to the following:

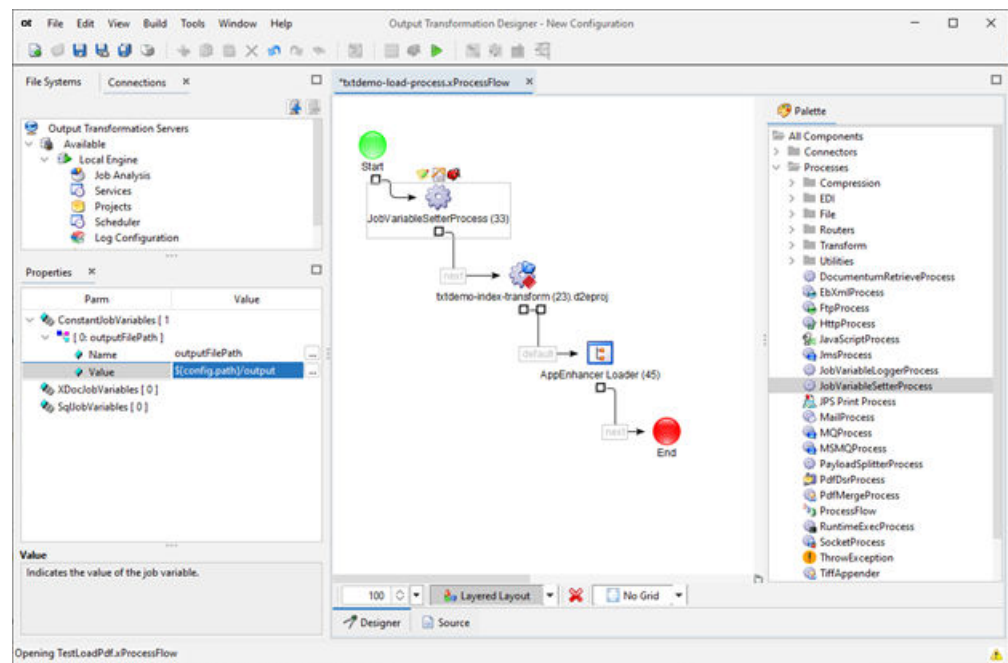


Figure 3-13: txtdemo-load-process with ConstantJobVariables defined

3.2.6.2 Mapping the Output Transformation Engine FdInput to XDoc

To map the FdInput parameter to use the XDoc:

1. Open the `txtdemo-load-process.xProcessFlow` project.
2. In the **Development** window, right-click on the Output Transformation Engine project (`txtdemo-index-transform.d2epro`) and from the context menu that appears, click **Show Mappings**.
3. In the **Source and Result Mappings** dialog box, switch to the **Source XDocs** tab.
4. In the Source XDocs table, in the FdInput parameter row, change the **Input Name** from the default value to **FILE**, which is the XDoc input mapping name.

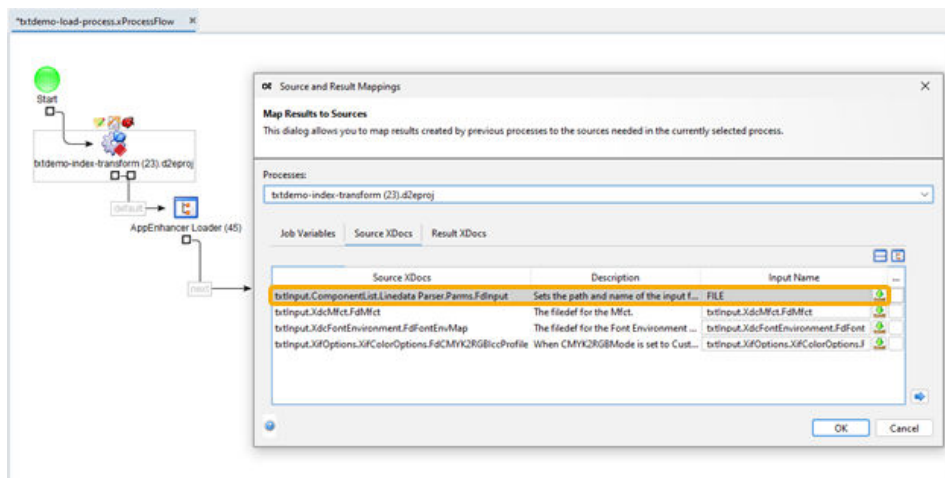


Figure 3-14: FdInput parameter set to FILE XDoc input mapping name

5. When you are finished, click OK.

3.2.7 Configuring the Output Transformation Engine project to use the input file job variable

There are a few different scenarios when the `${inputFile}` job variable can be used.

In the `txtdemo` project, you can configure any output files created to use the same base file name as the input file. For example, if the input file `c:\data\input\myInput.txt` is picked up by the File Event, it sets the `FilenameVariable` (which is set to `inputFile` in the sample) to be `myInput`.

Subsequently, you can use this as the base name for the `FdAsciiOutput` file.

Open the `txtdemo-index-transform.d2epro` Output Transformation Engine project and update the Linedata Parser's **FdAsciiOutput** parameter to the following value:

```
${outputFilePath}/${inputFile}-${JobTicket.id}.txt
```

In the Indexer component for the txtdemo project, you are indexing the original data file instead of using an Output Transformation Engine Generator component to create new output files to be loaded into AppEnhancer. Therefore, you must also modify the Indexer's Original Document Indexing project to use the same `${outputFilePath}/${inputFile}-${JobTicket.id}.txt` value used for FdAsciiOutput. This value shows up in the index file to tell the AELoader component where to find the data file(s) that should be loaded.



Note: Be aware when using the `${JobTicket.id}` job variable. This variable is similar to the previously used `${d2eJobId}` that provides a unique string based on the Output Transformation Engine job name. The `${JobTicket.id}` variable provides a unique string based on the OpenText Output Transformation Server Job Ticket that gets created for each OpenText Output Transformation Server job. In these examples, the JobTicket will be initialized by the FileEvent prior to invoking the process flow.

3.2.8 Configuring the Output Transformation Engine project to use default job variables for independent execution

In previous steps in this guide, the Output Transformation Engine project was configured so that it could be executed directly, before it was included in a process flow.

In more recent steps in this section, the Output Transformation Engine project was modified to use the following job variables that originate from the FileEvent and process flow's **JobVariableSetter** component:

- `${inputFile}`
- `${outputFilePath}`
- `${JobTicket.id}`

With this configuration, it means that running the Output Transformation Engine project directly for testing will fail because those variables are not defined or available unless the FileEvent triggers the process flow to execute. However, it is still desirable to sometimes execute the Output Transformation Engine project by itself if further tuning to XFT or Indexer parameters are required. To solve this issue, we can configure the Output Transformation Engine project to set some default values for these variables.

With your Output Transformation Engine project open, in the **Transformation Projects** tab, right-click the project name and from the context menu that appears, click **Show Properties**.

Then in the project tab in the Development window, you can add a new job variable by right-clicking the **JobVariables** node and selecting **Add**. You must create new entries for each of the following job variables:

- **Name:** outputFilePath

- **Value:** {d2eAppPath}output
- **Name:** fileName
 - **Value:** {d2eInputFileName}
- **Name:** JobTicket.id
 - **Value:** {d2eJobId}

Following these updates, the Output Transformation Engine project can be run successfully in isolation.

When the Output Transformation Engine project is run independently, these new default values will be used. However, when the FileEvent triggers the Output Transformation Engine project to run, these new job variables will already be defined in the OpenText Output Transformation Server JobTicket when the Output Transformation Engine project initializes at runtime.

3.3 Creating and configuring a process flow for end-to-end testing

The previous section explained how to ensure the Output Transformation Engine project can be run in isolation by defining default job variables. Running the Output Transformation Engine project this way is useful during project development to manually inspect and verify the output files, such as PDF output files or the index file to ensure page ranges and index fields for each document look accurate.

During development it is also useful to run tests that run an end-to-end process of indexing and loading a test file into AppEnhancer so you can view and verify the content has been loaded correctly in AppEnhancer. This means we want to run the entire process flow, including the Output Transformation Engine project and the AELoader component. A test project can be created without using the FileEvent by creating a process flow that feeds in a hard coded test file through the XDoc.

3.3.1 Creating the process flow for end-to-end testing

To create a process flow for end-to-end testing:

1. In Output Transformation Designer, go to **File > New**.
2. In the **New Component Wizard**, go to **Processes > ProcessFlow** and then click **Next**.
3. In the txtdemo sample project, there is a **txtdemo-load-process.xProcessFlow** so you will create a test version of the process flow.

On the **Name and Location of New ProcessFlow** screen, enter **txtdemo-load-process-test.xProcessFlow** into the **Name** box. Then you want to save the new process flow in the same folder as the existing process flow so you can indicate its location using the **File System** and **Directory** boxes.

When you are finished, click **Finish**.

4. With this new process flow open in the **Development** window, from the **File Systems** tab drag the existing **txtdemo-load-process.xProcessFlow** into the new flow.

3.3.2 Adding a GetFile component to the process flow

To add a GetFile component to the testing process flow:

1. Open **txtdemo-load-process-test.xProcessFlow** in Output Transformation Designer.
2. In the **Palette**, go to **Processes > File** and drag the **GetFile** component to the test process flow.
3. In the **Add Component** dialog box, select **Shared** and enter a **Name** for the component, such as **get-test-file**. Using the **File System** and **Directory** boxes, either choose the same folder as the process flow or alternatively, you can use the **resources** subfolder.

When you are finished, click **OK**.

The new instance of GetFile opens on a new tab in the Development window.

4. Use the **DirName** and **FileName** parameters to indicate the file path for your test file,

When you are finished, **save** your changes.

5. Return to the test process flow tab. Right-click the **GetFile** component and from the context menu that appears, click **Show Mappings**.
6. In the **Source and Result Mappings** dialog box, switch to the **Result XDocs** tab. On this tab, you should see that the configuration is already setup to map the selected file to a **Destination** XDoc named **FILE**.

In the previous steps, the sample used the XDoc mapping of **FILE** so this default value is correct. However, if you have used another name then the Destination name must be modified accordingly.

When you are finished, click **OK**.

The final process flow should appear similar to the following figure:

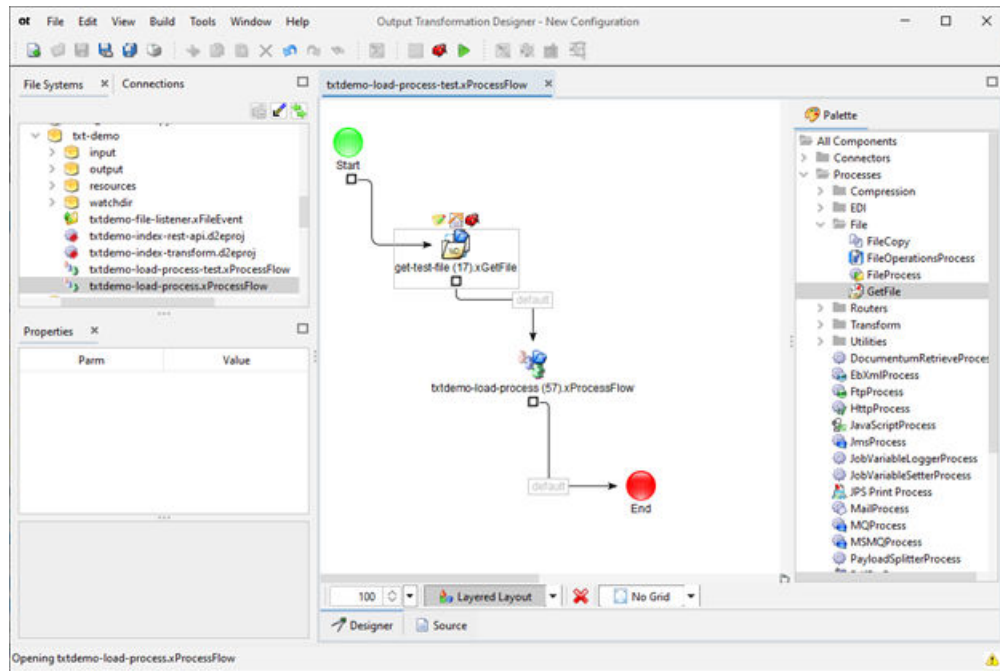


Figure 3-15: Sample test process flow with GetFile component and txtdemo-load-process process flow

Chapter 4

Setting up AppEnhancer

In AppEnhancer, you must configure the application using the following settings.

4.1 Required IIS settings

You must set certain properties for the AppEnhancer websites in IIS Manager to allow loading into AppEnhancer. Using the **Configuration Editor**, you must update the properties for each of the following websites in your instance:

- AppEnhancer
- AppEnhancerAdmin
- AppEnhancerReST

For each website, you must update various configuration properties with new values. For each of the following properties, the recommended values in kilobytes (kb) are shown in parenthesis:

- Under `system.web/httpRuntime`
 - `maxLength` (1048576)
- Under `system.webServer/serverRuntime`
 - `maxRequestEntityAllowed` (4294967295)
 - `uploadReadAheadSize` (1073741824)
- Under `system.webServer/security/requestFiltering`
 - `maxAllowedContentLength` (1073741824)



Tip: Alternatively, you can also make these changes directly in the `web.config` file.

4.2 Minimum required permissions for AppEnhancer users

To allow documents to be loaded into AppEnhancer from OpenText Output Transformation Server, an administrator must configure the privileges for the user performing the loading in the **Application Management** node in AppEnhancer Administrator.

In most situations, you should create a new AppEnhancer user for exclusive use by the AELoader component. A new user with only the minimum required privileges is highly recommended for loading since existing users may have too many permissions assigned to them, which may result in consumption of more of the limited amount of licenses than necessary.

The following list notes the minimum required privileges that must be assigned to the user:

- **Batch Scan**
- **Batch Index**
- **Add Page**
- **Multiple Logins**

The following additional privileges are required if the AELoader needs to assign multiple indexes to the same document:

- **Modify Index**
- **Display**

Chapter 5

Recommended Settings

Some recommended settings for the various applications are discussed in this chapter.

5.1 Optimizing performance in an Indexing project

In Output Transformation Engine, when creating a new XFT project the default behavior is to create a DOM structure in memory from the XFT results. The DOM properties are used in many use cases, such as during the creation of Accessible PDF (PDF/UA) and Data Transformation Engine component mappings.

When working with Output Transformation for AppEnhancer, DOM structures are not typically required and therefore, it is recommended users turn off this feature to eliminate the associated CPU and memory overhead.

To turn off DOM properties, edit your XFT Document parameters and make sure the **Enable DOM Properties** check box is not selected.

If DOM properties are enabled when working with Output Transformation for AppEnhancer, you may see the following message in the all-debug or Output Transformation Engine job logs:

```
WARNING - Dom Usage in Xft Document is enabled, Indexer Document Breaking is enabled:  
to avoid Out Of Memory issues Indexer must only use 'XftDocument change' type of  
document breaking
```

5.2 Performance tuning

For best performance, it is highly recommended you follow these general use case guidelines.

High frequency usage

If you are loading into AppEnhancer often with a low page count (1-50 pages) per document, then:

- Install AppEnhancer and OpenText Output Transformation Server on the same machine
- Recommend running on a system with minimum 4 GB of RAM and 2 cores, over the standard minimum system requirements
- For OpenText Output Transformation Server system configuration:
 - Total number of MaxConcurrent jobs: 15 (5 jobs for indexing and 10 jobs for loading)

If you are loading into AppEnhancer often with a high page count (51 to >1000 pages) per document, then:

- Install AppEnhancer and OpenText Output Transformation Server on separate machines
- Recommend running with a minimum 8 GB of RAM on each system, over the standard minimum system requirements
- For OpenText Output Transformation Server system configuration:
 - Total number of MaxConcurrent jobs: 20 (10 jobs each for indexing and loading)

Low frequency usage

If you are loading into AppEnhancer sporadically with a low page count (1-50 pages) per document, then:

- Install AppEnhancer and OpenText Output Transformation Server on the same machine
- Recommend running on a system with minimum 4 GB of RAM and 2 cores, over the standard minimum system requirements
- For OpenText Output Transformation Server system configuration:
 - Total number of MaxConcurrent jobs: 15 (5 jobs for indexing and 10 jobs for loading)

If you are loading into AppEnhancer sporadically with a high page count (51 to >1000 pages) per document, then:

- You can choose from either High frequency usage scenario. Note that if the low page count requirements are used, then expect the AppEnhancer load times to be 2-3 times longer than a typical ERMX load. If the high page count requirements are used, then expect the AppEnhancer load times to be relatively similar to an ERMX load.

If you are loading into AppEnhancer frequently with a low page count per document, but occasionally execute a large load with high volume (for instance, a monthly run with a large number of documents and a high page count per document), then:

- It is recommended you use the High frequency usage with high page count per document scenario to ensure that performance is not significantly compromised when executing large loading jobs.

Chapter 6

Special implementations

The recommended techniques for some atypical, but not uncommon, implementations are outlined in this chapter.

6.1 Skipping pages in Linedata Parser projects when loading into AppEnhancer

If you are designing a project where some pages from the original document must be excluded when loading into AppEnhancer, using the Skip Page component logically makes sense since the component can eliminate pages from the workflow based on user configuration.

When configuring the Output Transformation Engine project, the Skip Page component must be used in conjunction with a generator component, such as the PDF Generator, to produce the new output. In the project, each page is sent downstream to the next component in the workflow until it reaches the generator.

However, in an Output Transformation for AppEnhancer project workflow, the Skip Page component is typically positioned after the XFT component and before the Indexer and generator components. Consequently, the pages are removed before reaching the Indexer and generator components so they are not aware that the pages ever existed.

In projects using the Linedata Parser to load normalized ASCII or UTF-8 data into AppEnhancer, there is no generator component. Instead, the Linedata Parser writes out the normalized ASCII or UTF-8 data based on the Linedata Parser's FdAsciiOutput parameter. Therefore, an alternative strategy is needed to exclude Linedata pages from loading into AppEnhancer.

To exclude pages in the index file coming from the Linedata Parser, it is recommended you configure an Indexer component to eliminate pages from some XFT documents from the index file. Then, the AELoader component will process the index file and detect the missing page ranges to exclude them when it sends the FdAsciiOutput data to AppEnhancer. In summary, you must complete two parts:

1. Setting up XFT documents to separate the excluded content from the input
2. Configuring Indexer to filter out the excluded XFT document(s)

Step 1: Setting up XFT documents to separate the excluded content from the input

To set up XFT documents to separate the excluded content from the input, you can review the following use case example and follow the general principles when designing your own project:

1. In Output Transformation Designer, create at least two documents in XFT to separate the excluded pages from the pages to preserve.

In the XFT window, shown in [Figure 6-1](#), the MainDocument and ExcludedPages XFT Documents have been created.

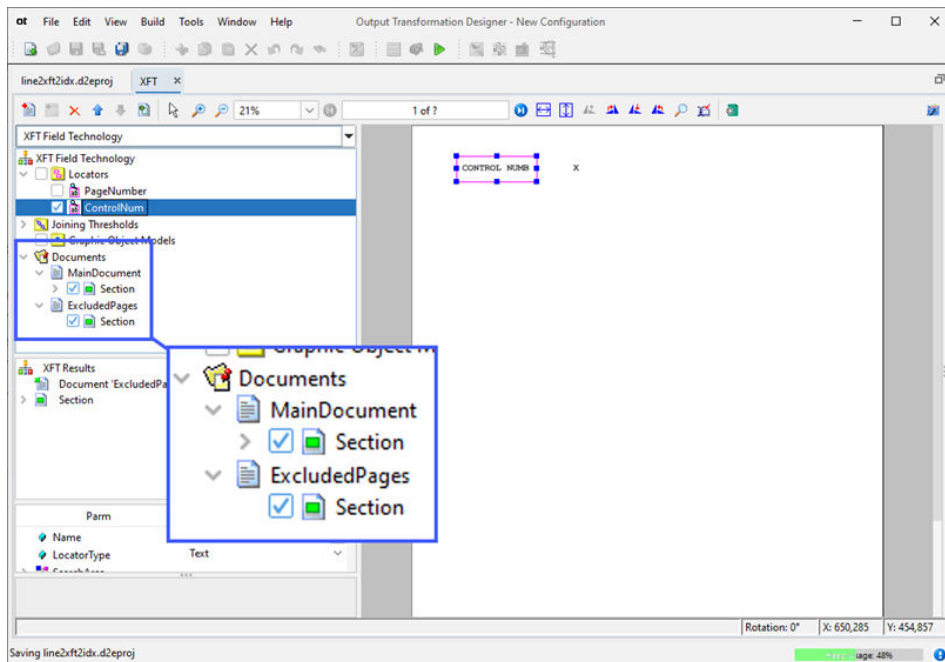


Figure 6-1: XFT window with two XFT Documents created

2. Create two XFT Locators. One locator will be used to identify pages to exclude while the other locator will be used to identify text that occurs on the first page of the main document to preserve.

[Figure 6-1](#) shows two XFT Locators, PageNumber and ControlNum. The PageNumber locator will be used to identify the text that occurs on the first page of the main document to load into AppEnhancer.

As shown in the figure, the ControlNum locator is highlighted and is configured to look for the CONTROL NUMB string that appears on the pages you want to exclude from loading into AppEnhancer.

3. Set up the starting identifiers on the respective **Start Identification** tabs for the XFT documents.

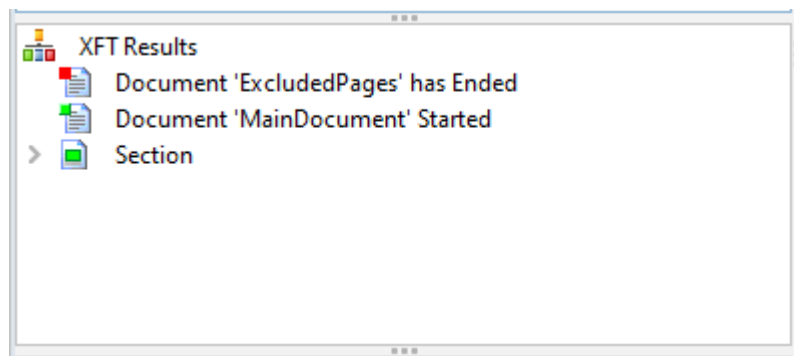
Configure the MainDocument XFT Document with the PageNumber locator as the start identifier. This locator is used to identify text that occurs on the first page of the main document that you want to load into AppEnhancer.


Configure the ExcludedPages XFT Document with the ControlNum locator as the start identifier.

This locator is used to identify the `CONTROL NUMB` string and when found on a page, will ensure the particular page gets captured in an XFT Document instance named `ExcludedPages{}`.

4. When the documents are configured correctly, you can step through the pages in the document and observe the XFT Results tab to verify the correct document switching.

For example, assume that the first two pages in a document should be excluded during processing and the third page is a regular page that should be loaded. When viewing the XFT Results tab for page 3, it should match the following figure, which indicates that the `ExcludedPages` document has ended on the previous page (page 2) and the current page (page 3) begins the `MainDocument`.



 **Note:** Be aware that there can be any number of document changes throughout the file since there can be more instances of a filter condition similar to `ExcludedPages` in the middle of a file.

Step 2: Configuring Indexer to filter out the excluded XFT document(s)

To configure the Indexer component to filter the XFT documents:

1. In Output Transformation Designer, open the Indexer component and on the **Indexing** tab, select the **Enable XFT Document filtering** check box to enable the **XFT Doc Index Filter** tab.
2. On the **XFT Doc Index Filter** tab, in the **Conditions** pane click **Add** to create a new filter entry.
3. In the **Condition Definition** pane, fill the **Skip Filter Entry Mode**, **String Kind**, and **String Value** boxes with your filtering criteria.

For our use case example, you will configure the Skip Filter Entry Mode to **Include** the **MainDocument** String Value in our index filter.

Alternatively, configuring the Skip Filter Entry Mode to **Exclude** and String Value as `ExcludedPages` would also provide the same result in the index file.

4. When you are finished, click **Apply**.

When the Indexer component configuration is complete, you can run the project and review the index file to verify that the page ranges to exclude are not present in the index file and the remaining pages will load into AppEnhancer as expected. In the following figure, the log file shows that the initial GROUP_OFFSET is page 3, which is as expected because pages 1 and 2 are omitted in our use case example, and its inclusion indicates that it will be loaded into AppEnhancer.

```

1 COMMENT: Generated By OpenText Output Transformation
2 COMMENT: Started 16-Oct-2023 10:51:37 AM
3 COMMENT:
4 CODEPAGE:500
5
6 GROUP_FIELD_NAME:PageNum
7 GROUP_FIELD_VALUE:000001
8 GROUP_FIELD_NAME:StatementNumber
9 GROUP_FIELD_VALUE:12345
10 GROUP_FIELD_NAME:Address
11 GROUP_FIELD_VALUE:Abc Company, 1234 Main St.
12 GROUP_OFFSET:3
13 GROUP_LENGTH:1
14 GROUP_FILENAME:c:\output\myproject\myFdAsciiOutputFile.txt
15
16 GROUP_FIELD_NAME:PageNum
17 GROUP_FIELD_VALUE:000004
18 GROUP_FIELD_NAME:StatementNumber
19 GROUP_FIELD_VALUE:678910
20 GROUP_FIELD_NAME:Address
21 GROUP_FIELD_VALUE:Maple Syrup Co. 678 Maple Drive
22 GROUP_OFFSET:4
23 GROUP_LENGTH:1
24 GROUP_FILENAME:c:\output\myproject\myFdAsciiOutputFile.txt

```



Note: For more information about the Indexer component and the XFT Doc Index Filter tab, see Indexer component *OpenText Embedded Output Transformation Engine - User Guide (VDTOTS-H-UTE)* in Output Transformation Engine User Guide.

6.2 Creating multiple AppEnhancer documents that include the same page

Some projects will require that one or more pages in an input document being loaded in AppEnhancer as more than one document.

For example, consider a document that has a single “Customer Info” section at the top of each page, followed by one or more “Invoice Info” sections for that customer on the same page. The Customer Info section could contain fields for a CustomerID and CustomerName. Then the Invoice Info section could contain fields for InvoiceID and TotalAmount. The result of loading this set of documents into AppEnhancer may appear similar to the following:

CustomerID	CustomerName	InvoiceID	TotalAmount	Document page
1234	FirstName1 LastName1	4444	\$123.45	10
1234	FirstName1 LastName1	5555	\$1023.33	10
1234	FirstName1 LastName1	6666	\$67.87	10
2335	FirstName2 LastName2	7777	\$2876.45	11
2335	FirstName2 LastName2	8888	\$5678.11	12

As shown, page 10 of the document has content on three different invoices for FirstName1 LastName1, but FirstName2 LastName2 has two invoices on different pages, 11 and 12. This indicates that when one of FirstName1 LastName1's documents are viewed in AppEnhancer, page 10 is always shown since it contains all three of their invoices.

To create multiple AppEnhancer documents that include the same page or page ranges, you must complete the following steps:

1. Configuring XFT to locate all unique field instances
2. Configuring the Indexer to recognize multiple indexes per document

Step 1: Configuring XFT to locate all unique field instances

You must configure the XFT Field Technology component to locate multiple instances of the same XFT Section or Text Field. To locate all instances in the input document, there are two separate parts you must configure:

1. Setting up dynamic positioning
2. Allowing multiple sections per page

As illustrated in [Figure 6-2](#), you can see that in our sample document a CustomerInfo section has been located three times on the same page in the XFT Results. The corresponding CustomerInfo sections in the document for each XFT result is shown by matching the numbers.

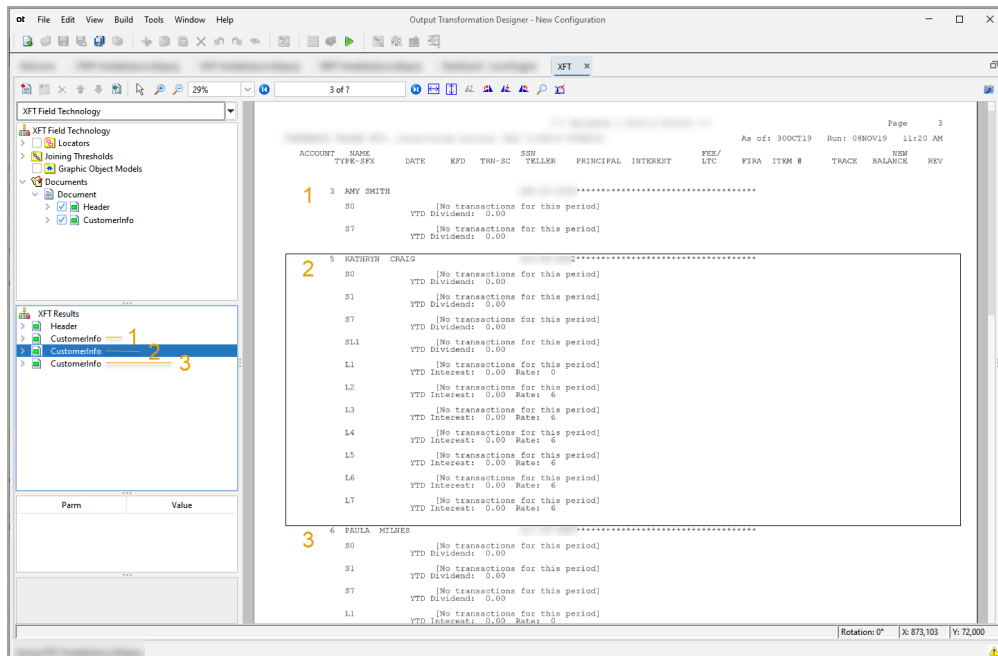


Figure 6-2: XFT window showing CustomerInfo locators and the corresponding sections

Setting up dynamic positioning

To begin, in the XFT Section Properties dialog box, on the **General** tab, select **Enable Dynamic Positioning** to enable the **Positioning** tab options where you will dynamically define the position of each instance of the section with bounding boxes.

In [Figure 6-2](#), you can see the leftmost column contains the account numbers for each customer: 3, 5, and 6. For this sample project, an XFT Locator named DocId was created to search for and detect every account number.

Logically, you would set up the top of the CustomerInfo Section to start at the line of text where the corresponding DocId value is located, and then the bottom of the section area would be the line above where the next DocId value is located. The concept is to define the default size of the XFT section (that is, where the initial bounding box is drawn when creating the Section) to cover the maximum boundaries of the first and last sections on the page. This sets some default values, which are important for the final section on the page that does not have any additional sections following it.

The final configuration on the Positioning tab should appear similar to the following:

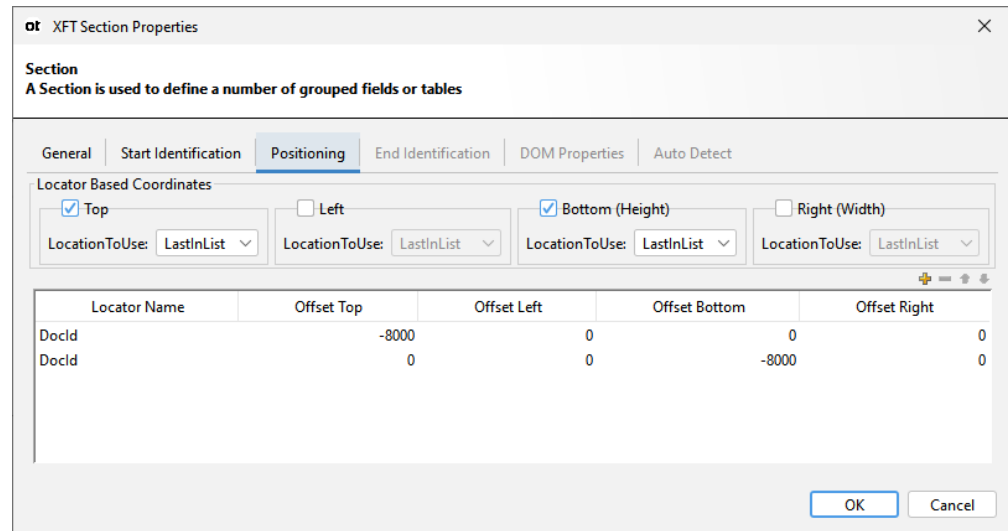


Figure 6-3: XFT Section Properties dialog box Positioning tab sample

Two entries for the same locator name are defined so that XFT will locate all of the DocId's on the page. In the **Locators** list, the first DocId entry represents the DocId for the next section that was found, which is based on the **Offset Top** value (top of the bounding box) relative to where the application found the locator. The -8000 value is small in the context of XFT 72000 DPI units and indicates a shift of approximately 1/10th of an inch above the locator's found position. The second DocId entry represents the next DocId locator instance that will be found further down the page, which is the DocId for the next section. The section for this DocId is configured to end above this locator by using a -8000 value for the **Offset Bottom** for the section.

When processing the third and final Customer Info section on the page, there is no additional DocId locator instance to use to position the bottom of the bounding box. In this case, the bottom position of the section uses the default values set for the maximum boundaries, as previously described.

Allowing multiple sections per page

Be aware that XFT offers plenty of flexibility so this example is not the only method to capture multiple values, and some other user cases may need to cover multiple pages in the same document. For instance, you may have a single section that spans multiple pages and locates the same fields on each page.

The only requirement is to make sure you have multiple XFT/Index fields with the same name in the same logical document. When this occurs and multiple sections per page is turned on, the Indexer will detect the duplicate XFT fields in each document for a single document.

To turn on multiple sections per page, in the XFT Section Properties dialog box, on the **General** tab select **Enable Multiple Sections Per Page**.



Tip: In the context of Indexing, an XFT Document and an Indexer Document are referring to different entities. If the Indexer's Document Breaking parameters are using the **XFT Document Change** mode, there will be a one-to-one correlation between every XFT Document and Indexer Document. If another Document Breaking mode is used, there may be a different ratio. For example, you may use a single XFT Document for all pages and set the Indexer's Document Breaking parameter to use **XFT Field Change** when the value of CustomerID changes. The index file produced by the Index Writer describes a set of **Indexer Documents**.

Step 2: Configuring the Indexer to recognize multiple indexes per document

In the Indexer component, on the **Indexing** tab, select **Enable multiple indexes per document** to turn on the ability to create multiple Index Documents for the same document. Once enabled, multiple entries in the index file for the same page range will be automatically created.

In the following figure, you can see that from our input document there are three documents pointing to page 3 (`GROUP_OFFSET:3`), with different DocId values (3, 5, and 6) as indicated by `GROUP_FIELD_VALUE`.

```

35 GROUP_FIELD_NAME:ClientName
36 GROUP_FIELD_VALUE:
37 GROUP_FIELD_NAME:DocumentType
38 GROUP_FIELD_VALUE:
39 GROUP_FIELD_NAME:AsOfDate
40 GROUP_FIELD_VALUE:30OCT19
41 GROUP_FIELD_NAME:RunDate
42 GROUP_FIELD_VALUE:08NOV19
43 GROUP_FIELD_NAME:DocID
44 GROUP_FIELD_VALUE:3
45 GROUP_FIELD_NAME:CustomerName
46 GROUP_FIELD_VALUE:AMY SMITH
47 GROUP_OFFSET:3
48 GROUP_LENGTH:1
49 GROUP_FILENAME:C:\Users\
50 GROUP_FIELD_NAME:ClientName
51 GROUP_FIELD_VALUE:
52 GROUP_FIELD_NAME:DocumentType
53 GROUP_FIELD_VALUE:
54 GROUP_FIELD_NAME:AsOfDate
55 GROUP_FIELD_VALUE:30OCT19
56 GROUP_FIELD_NAME:RunDate
57 GROUP_FIELD_VALUE:08NOV19
58 GROUP_FIELD_NAME:DocID
59 GROUP_FIELD_VALUE:5
60 GROUP_FIELD_NAME:CustomerName
61 GROUP_FIELD_VALUE:KATHRYN CRAIG
62 GROUP_OFFSET:3
63 GROUP_LENGTH:1
64 GROUP_FILENAME:C:\Users\
65 GROUP_FIELD_NAME:ClientName
66 GROUP_FIELD_VALUE:
67 GROUP_FIELD_NAME:DocumentType
68 GROUP_FIELD_VALUE:
69 GROUP_FIELD_NAME:AsOfDate
70 GROUP_FIELD_VALUE:30OCT19
71 GROUP_FIELD_NAME:RunDate
72 GROUP_FIELD_VALUE:08NOV19
73 GROUP_FIELD_NAME:DocID
74 GROUP_FIELD_VALUE:6
75 GROUP_FIELD_NAME:CustomerName
76 GROUP_FIELD_VALUE:PAULA MILNES
77 GROUP_OFFSET:3
78 GROUP_LENGTH:1

```

Figure 6-4: Different DocId values retrieved from the sample page sample

6.3 Using scripting for complex document breaking rules

Sometimes a project will require complex rules to define documents and the standard options in the Indexer's Input Breaking tab are unable to handle the rules. In some cases, using JavaScript can provide the custom functionality needed for your project.

In the following example, the project will be able to:

1. Define XFT fields to inspect to determine document breaking boundaries
2. Use a Script component to inspect XFT field value(s) and set job variables that can be used for document breaking

3. Use an Indexer component to create a custom index field that uses a job variable from the Script component
4. Configure the Indexer's Input Breaking to break on Index Entry Change and configure it to change on the custom index field

The following sections discuss the key points of this project flow.

```
1 var currentDoc = jobState.getXifDocument();
2 var currentPage = jobState.getXifPage();
3 var filePageNumber = 0;
4
5
6 if (currentPage) {
7
8     var documentType = jobState.getXftFieldOnPage("Doctype");
9     var formOverlay = '';
10    var documentName = '';
11
12    switch(documentType){
13        case "POSINSTR":
14            documentName='ASSEMBLY';
15            formOverlay = documentName;
16            break;
17        case "POSTRVLR":
18            documentName='TRAVELR';
19            formOverlay = documentName;
20            break;
21        case "POSCRT02":
22            documentName='CERT';
23            formOverlay = documentName;
24            break;
25        case "POS1STPG":
26            documentName='POS';
27            formOverlay = documentName;
28            break;
29        case "POSCONTD":
30            documentName='POS';
31            formOverlay = "POS_2";
32            break;
33        default:
34            documentName='POS';
35    }
36
37    jobState.setJobVar("DOCUMENT_TYPE", documentName);
38    lib.logInfo("DOCUMENT TYPE IS " + documentName);
39
40    jobState.setJobVar("DOCUMENT_FORM", formOverlay);
41    lib.logInfo("FORM OVERLAY IS " + formOverlay);
42 }
```

Figure 6-5: Code snippet from Script component sample

As shown on line 8 of the previous code snippet from the Script component, it has been configured to retrieve an XFT field value called `DocType` and then inspect the value of the field to determine what the document type is.



Note: The complicated rules in this example are that if the `DocType` field is “POS1STPG” (first page) or “POSCONTD” (continued), then they apply to the same document. These pages appear sequentially in the document. Furthermore, there can be a sequence of “POSINSTR” pages in a document.

After setting the JavaScript variable `documentName`, it is populated in a job variable on line 37.

Then the Indexer component must be configured on the **Custom Properties** tab with a custom Index field that uses `@jobVar("<jobVarName>")` as the **Value** of the **Property Definition**, where `<jobVarName>` is the job variable set in the Script component. The job variable can also be expressed as:

```
DOC_TYPE = @jobVar("DOCUMENT_TYPE_VAR")
```

Continuing to the **Document Breaking** tab of the Indexer component, select **Indexer Entry Change**. Then configure the **Index Entry Change** to **Any Field Changes** and the **Selected Field** to `DOC_TYPE` so that document breaking occurs whenever the `DOC_TYPE` field is found.

Chapter 7

Troubleshooting

This chapter contains information that can help you evaluate your output and review log files to solve the most common problems or questions related to difficulties encountered during processing.

7.1 When loading input, I am getting a “Request Entity Too Large” error message.

To troubleshoot this issue, open the audit log file to verify the exact error that caused the load failure. If your audit log contains the following line:

```
<title>IIS 10.0 Detailed Error – 413.1 – Request Entity Too Large</title>
```

Then it is confirmed to be an IIS-related issue.

To resolve this issue, you must go to the IIS Manager and verify your settings under the **Sites** node on the **Connections** pane for the AppEnhancer websites in your instance. For more information on the configuration settings, see [“Required IIS settings” on page 57](#).

7.2 After loading is finished, I don’t see my overlays.

To troubleshoot this issue, you can check the audit log that is produced by the AELoader process. The following snippet shows the beginning of a sample AELoader log file:

```
ApplicationXtender Loader Log File
AX Loader Parms...
  AuthType = Basic
  Username = OTSWRITER
  Password= ??-?-?
  HostURL = http://someserver/AppXtenderReST
  DatasourceName = AX_PROD
  ApplicationName = TEST_INVOICE1
  IndexFile = ${d2eAppPath}..\Customer_Invoices\Ouput\Indexes\${inputFileName}_$
  {JobTicket.id}.ind -> E:\OTSWorkspace\ParserProcesses\..\Customer_Invoices\Ouput\Indexes
  \OUTPUT_20211208_0135_002.ind
  AuditFileName = E:\OTSWorkspace\Customer_Invoices\Ouput\Logs\${inputFileName}_$
  {JobTicket.id}.audit -> E:\OTSWorkspace\Customer_Invoices\Ouput\Logs
  \OUTPUT_20211208_0135_002.audit
  DocumentOptions...
    OnError = FAIL
    OnDuplicateIndexEntry = NEW_DOCUMENT
    KeyReferenceUpdate = false
    IgnoreDlsViolation = false
    COLDFormName = BUSOFF
    SubmitFullText = false
  BatchOptions...
    BatchIDProperty = OTSLOAD_ID
    BatchIDFormat = OtsLoad_CustomerInvoices: ~${AXBatchUuid}
  DateFormat...
    SourceDateFormat =
```

```
TargetDateFormat =  
ReliableTransfer...  
PerformCleanup = true    Retries = 0  
TimeBetweenRetries = 5000  
OnError = ThrowException
```

Within the log file, check the following settings:

- Verify that the **DocumentOptions > COLDFormName** parameter has a value.
- Verify where the configuration was executed because running from Output Transformation Designer or the server results in separate locations.
- Verify that there is a valid form name found in the **_FORMS** application in AppEnhancer.
- Verify that the audit log was produced from the specific load.

7.3 An error message stating “Expected next offset (page) at >=2 but found 1” was shown.

This message indicates that the page offsets in the index file are not incrementing sequentially. In normal circumstances, the index file contains the following pattern:

```
GROUP_FIELD_NAME:Department  
GROUP_FIELD_VALUE:A  
GROUP_OFFSET:1  
GROUP_LENGTH:1  
GROUP_FILENAME:somefile.txt  
GROUP_FIELD_NAME:Department  
GROUP_FIELD_VALUE:B  
GROUP_OFFSET:2  
GROUP_LENGTH:1  
GROUP_FILENAME:somefile.txt
```

In the previous example, the group offsets are incrementing. If you want to set up multiple indexes per document, then the page offsets should not be incremented because you want to apply a different set of indexes to the same document pages. In this scenario, the index file appears similar to the following example:

```
GROUP_FIELD_NAME:Department  
GROUP_FIELD_VALUE:A  
GROUP_OFFSET:1  
GROUP_LENGTH:1  
GROUP_FILENAME:somefile.txt  
GROUP_FIELD_NAME:Department  
GROUP_FIELD_VALUE:B  
GROUP_OFFSET:1  
GROUP_LENGTH:1  
GROUP_FILENAME:somefile.txt
```

When an index file is configured for multiple indexes per document and the application in AppEnhancer is not set up for multiple indexes, then the “Expected next offset (page) at >=2 but found 1” message is issued.

To resolve the issue, you must enable the **Multiple indexes referencing single document** option in the **Additional Settings** section on the Applications tab for the Application in AppEnhancer Administrator. However, if the application is not supposed to support multiple indexes, then the indexing is incorrect and you must

return to the Output Transformation Engine project to ensure the index file is generated properly.

7.4 The audit log does not provide any error message details.

When an error occurs during loading and it is caused by a server side operation, usually an error message is provided and displayed in the audit logs. However, some scenarios do not generate a log message. For example, error code 115 is triggered when performing multi-index loading by page and the user does not have the correct permissions. As an example of an audit log that does not display message details, the log file may appear similar to the following example:

```
AxApplicationId: 178
Input index file: D:\OTS\Output\Multi-Org_inv-20230720_1226_002.idx
2023-07-20 12:27:28,-1,eeekmpli,,STR,-1,-1,-1,
2023-07-20 12:27:28,-1,poacbild,,INFO,-1,-1,-1,Index Fields=[INV-
CR #:,DOC_DATE:,TOTAL:,CUST_NAME:,CUST_ORD_NUM:,CUST_ACCT #:,PKSHT-DEBIT #:,DOC_TYPE:]
2023-07-20 12:27:28,484,hcmfmplic,,UBE,-1,131,-1,OtsUuid=OTSLOAD:
E703BA65-1113-453D-9E22-935620B333D4:1|F=D:\OTS\Output\Multi-
Org_doc-20230720_1226_002.pdf
2023-07-20 12:27:29,662,ejffagck,1:0:1:23,CDE,11,131,65,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]
2023-07-20 12:27:29,31,cknlfkig,1:1:1:42,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
2023-07-20 12:27:29,16,akhbjoed,1:2:1:61,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
2023-07-20 12:27:29,15,ojbhmcnp,1:3:1:80,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
2023-07-20 12:27:29,16,fmceeicp,1:4:1:99,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
2023-07-20 12:27:29,16,kbnbched,1:5:1:118,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
2023-07-20 12:27:29,16,micikloo,1:6:1:137,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
2023-07-20 12:27:29,15,ophocajk,1:7:1:156,IDS,-1,131,-1,
I=[1234567890,01/01/2000,$999.99,COMPANYNAME,1234567890S,1234567,123456789,DOCTYPE]::
Err[]
```

As a general rule, if the audit log does not provide enough information, you can inspect the AppEnhancer Server logs (that is, Windows Event Logs) and in most cases, this file will contain further information.

7.5 The audit log shows an “Err[Entity input stream has already been closed.]” message.

You may encounter this type of message in the audit log during loading. This is a general error and in most cases, it indicates that you are trying to connect to AppEnhancer using HTTP protocol when AppEnhancer requires using HTTPS.

To resolve this issue, verify that the **Host** parameter in the AELoader is set to a URL that uses HTTPS protocols. For example, if the Host value is set to `http://<AE_Host>/` then you must modify the value to be `https://<AE_Host>/` instead.

7.6 The audit log shows an “Err[org.glassfish.jersey.client.InboundJaxrsResponse cannot be cast to com.xenos.applicationxtender.api.rest.BatchesResponse]” message.

You may encounter this type of message when you are trying to connect to AppEnhancer using HTTP protocol when AppEnhancer requires using HTTPS.

To resolve this issue, verify that the **Host** parameter in the AELoader is set to a URL that uses HTTPS protocols. For example, if the Host value is set to `http://<AE_Host>/` then you must modify the value to be `https://<AE_Host>/` instead.

7.7 There is a discrepancy in the number of documents loaded using Output Transformation Server versus the number loaded by ERMX.

Some users are discovering that the number of logical documents loaded by ERMX does not match the number loaded by OpenText Output Transformation Server. This issue is commonly encountered during project migration testing.

In both applications, a single large input source file is split into many individual documents during the load process. The rules used to break the documents are part of the project configuration in both cases. Therefore, when the document counts do not match, this indicates that there is a discrepancy in the user configured rules used to split documents in ERMX versus OpenText Output Transformation Server.

In most ERMX to OpenText Output Transformation Server migration projects, the ERMX output will be considered the correct behavior. Therefore, to troubleshoot the mismatch in document counts, users must identify where document boundaries are different in their OpenText Output Transformation Server project.

For example, a document with 70 pages creates 12 documents when loaded via ERMX, however, only 9 documents are created when loaded via Output

7.7. There is a discrepancy in the number of documents loaded using Output Transformation Server versus the number loaded by ERMX.

Transformation Server. The following table compares how ERMX and OpenText Output Transformation Server handle document splitting:

Table 7-1: Comparison of document splitting page ranges between ERMX and OpenText Output Transformation Server

Document number	ERMX page range	OpenText Output Transformation Server page range
1	1–5	1–5
2	5–8	5–8
3	9–13	9–13
4	14–17	14–17
5	18–20	18–22
6	20–22	23–26
7	23–26	27–38
8	27–38	39–59
9	39–50	60–70
10	51–59	Not applicable
11	60–65	Not applicable
12	66–70	Not applicable

There are several page ranges that are different in the ERMX and OpenText Output Transformation Server projects. For instance, you can see that Document 5 was broken into two separate documents in the ERMX project (page ranges 18-20 and 20-22), but it is only one document in the OpenText Output Transformation Server project (page range 18-22). This pattern continues as the page ranges climb, with the ERMX project occasionally producing more documents and at other times, the OpenText Output Transformation Server project producing more.

To correct the unequal document counts between applications, you must review XFT or other rules used to break the document at those page ranges in OpenText Output Transformation Server and modify the rules accordingly to match the rules used in ERMX.

Discrepancies in document counts within larger data sets

The simplified example above used 70 pages and 12 documents and identifying discrepancies between ERMX and OpenText Output Transformation Server is relatively straightforward in small datasets like this. In larger datasets, there may be 1,000 or more documents to review and therefore discrepancies cannot easily be identified through manual inspection. In these situations, the following method is recommended:

1. Import a single file into AppEnhancer using ERMX.
2. View the result set for the import, including the column showing page counts for each logical document.
3. Using the Export feature, export the result set to a CSV file.
4. Repeat steps 1-3 using OpenText Output Transformation Server to load the file.
5. Prepare the CSV files for comparison.
 - a. If the columns of the CSV file are not equivalent, you can use Excel or another tool to remove the columns that are not required.
 - b. Keep the page count and document number/identifier.
6. Compare the two CSV files using your preferred text file comparison tool, such as WinMerge.

This technique should provide a way to identify the page ranges that cause the discrepancies. Furthermore, you can use OpenText Output Transformation Server features like XFT Results or custom script logging to help identify what adjustments need to be made to the OpenText Output Transformation Server project to reproduce the same rules used in the ERMX line and page procedure scripts.

7.8 After upgrading AppXtender to AppEnhancer, I am getting a 404 error.

A potential issue may occur after upgrading from AppXtender to AppEnhancer, which may result in the following error shown in this excerpt from the audit log:

```
AxApplicationId: -1 <AppId lookup failed>
Input index file: ...
2024-02-01 14:25:56,-1,eeekmpli,,STR,-1,-1,-1,
2024-02-01 14:25:56,186,hkjiaild,,TCO,-1,-1,-1,Err[<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"/>
<title>404 - File or directory not found.</title>
<style type="text/css">
```

This issue may occur after an upgrade if REST services were installed to listen at a different URL context path than what is configured in AppEnhancer Loader. During installation of AppEnhancer REST Services, you may encounter the default AppEnhancerReST name when setting up the **Application name**.

To correct this issue, you must make sure that the loader configuration is consistent with the URL for AppEnhancer REST services.

Historically, `https://<ae.server.host>/AppXtenderREST` was the AppEnhancer Loader **HostUrl** to use. Newer installs of AppEnhancer use a **HostUrl** value of `https://<ae.server.host>/AppEnhancerREST` by default.

7.9 I received an “The index could not be saved because a duplicate index already exists” error message.

This issue occurs when loading and one or more index fields are marked as part of a unique index. To resolve this issue, you can configure the **DocumentOptions** > **OnUniqueKeyViolation** parameter in the AELoader. For **OnUniqueKeyViolation**, there are two options:

- Skipping the page. Indicates the error should be ignored and to not load any of the pages. To enable this setting, you must set **OnUniqueKeyViolation** to **SKIP**.
- Appending the page. Indicates that the page(s) should be appended to the existing document. To enable this setting, you must set **OnUniqueKeyViolation** to **FAIL** and **OnDuplicateIndexEntry** to **APPEND_EXISTING**. Furthermore, to successfully append pages, you must make sure that all index values exactly match the existing values.

