TIBCO Foresight® Instream®

Document Splitter

Software Release 8.7
August 2017

Two-Second Advantage®

Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, Two-Second Advantage, TIBCO BusinessConnect, TIBCO Foresight EDISIM, TIBCO Foresight Instream, and TIBCO Foresight Translator are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java Platform Enterprise Edition (Java EE), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

The United States Postal Service holds the copyright in the USPS City State Zip Codes. (c) United States Postal Service 2017.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

General Contact Information
TIBCO Software Inc.
3303 Hillview Avenue
Palo Alto, CA 94304 USA
Tel: +1 650 846 1000
Fax: +1 650 846 1005

Technical Support
E-mail: support@tibco.com
Web: https://support.tibco.com
(Note: Entry to this site requires a username and password. If you do not have one, you can request one. You must have a valid maintenance or support contract to use this site.)

Contents

Introduction

Intended Audience

This document is intended for developers who are implementing Docment Spliter (Docsplitter).

Before using this document, familiarize yourself with TIBCO Foresight® Instream® validation, which is described in **TIB_fsp-instream_ <*n.n*>_usersguide.pdf**.

Capabilities

Docsplitter is the program in Instream® that lets you split these data types into valid and invalid data:

- HIPAA EDI
- X12 data
- EDIFACT data
- a flat file
- an XML file

For example, Docsplitter can sift out invalid claims in an 837 so that you can process the valid ones and notify the sender of the invalid ones.

You can call Docsplitter with a C++, C#, or Java API, with a command line, or with a script or batch file.

Data type	split valid/invalid	split content-based	split points	Split-point grouping
HIPAA	✓	✓	vary	✓
X12 non-HIPAA	✓		**ST** ISA,GS	

Data type	split valid/invalid	split content-based	split points	Split-point grouping
EDIFACT	✓	✓	**UNH** UNB, UNG	
Flat File	✓	✓	top of layout	
XML	✓		any repeating complex element	

For a list of where documents can split, please see Appendix B: Split Points on page 43.

Input and Output

Docsplitter runs after Instream validation and uses the validation results to separate the valid EDI from the invalid.

Docsplitter Input

- **The detail results file** created by Instream validation. It uses these records to build a splitting pattern for the EDI data.

  or

  A **997 or 999**. It splits the EDI based on the error reporting in a 997 or 999.

- **The data itself**. If it is EDI, it must have one or more complete interchanges per file. Each interchange can contain one or more functional groups, each with one or more transactions.

  It can be wrapped (no CR/LF) and/or folded (long lines chopped at certain number of bytes).

  (Instream comes with a command-line utility called EDIWRAP that manipulates wrapped and folded data. It is documented in **EDIwrap.pdf**.)

- A TIBCO Foresight guideline that generates these STRUS records in the Instream validation detail file:

  STRUS records with ISA, GS, and ST starting in the 16th position. Examples

```
STRUS          1|ISA|0|1|0
STRUS          2|GS|0|1|108
STRUS          3|ST|0|1|175
```

  SVALU records with ISA1, GSSG, and STST in the 16th position. Examples

```
SVALU          1|ISA1|0|ISA*00*          *00*  ...
SVALU          2|GSSG|0|GS*HC*901234572000*90  ...
SVALU          3|STST|1|ST*837*0386
```

TIBCO Foresight HIPAA guidelines that start with PD (example: PDSA837I) create these records. If you have merged your company guideline with one of these, the records will automatically be present.

For non-HIPAA data, add them yourself with TIBCO Foresight® EDISIM® Standards Editor as described in "Creating guidelines for Instream" in **TIB_fsp_edisim_<*n.n*>_fseditor.pdf**.

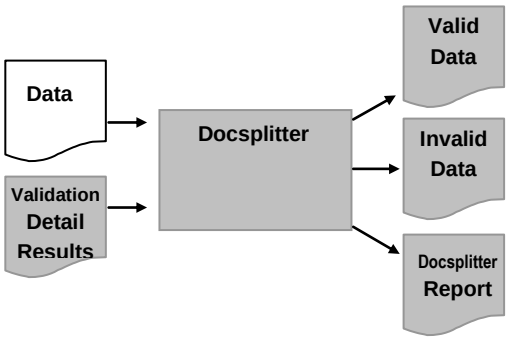
You can see a detail file containing these records by executing any of the Response Generator demos in Instream's Scripts directory.

Docsplitter Output

- A file containing the **valid** data.
- A file containing the **invalid** data.
- An XML or delimited report. This contains the status of each claim or other data being split, plus other information as described on page 38. You can use the report as input into notification systems including web applications.
- When using content-based splitting or split-point grouping, Docsplitter will create multiple split files.
- For EDI, output includes a CR/LF after the segment terminator unless:
  - The segment terminator in the input file is a LF.
  - You include the following section in a setup file (see Setup File on page 19):

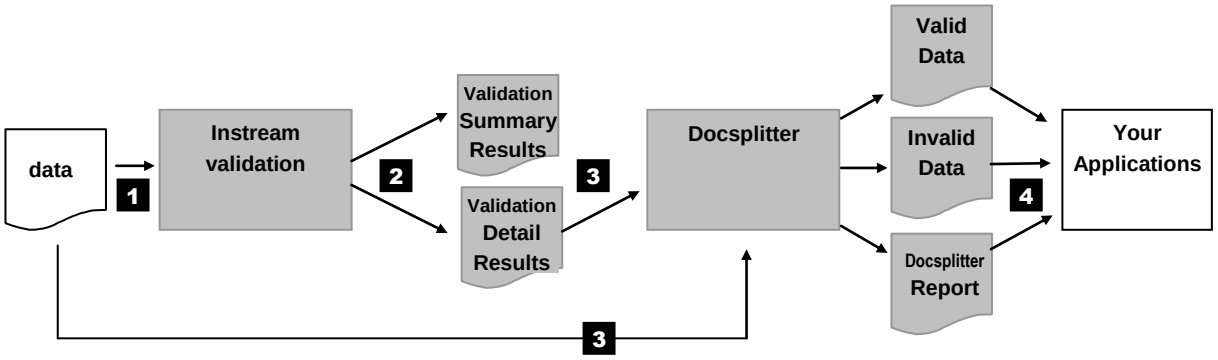
    ```
    [Options]
    OutputEDIWithCRLF=0
    ```
- Docsplitter fixes the output file's segment counts by default.

Input and Output



Gray objects are part of Instream or are created by Instream.

Example Simple Inbound Implementation



1. Instream reads and validates data.
2. Validation produces summary and detail results about the data's compliance.
3. Docsplitter reads the validation detail results and the data, and splits the data into valid and invalid files. It also creates an XML report.
4. Your applications act on the split data and the report.

What you need before using Docsplitter

Guideline

Guideline requirements:

Type of data	Guideline notes
HIPAA	Validation results must have STRUS records. Therefore, use a GuidelinePlus or a guideline merged with a GuidelinePlus. These guidelines ship with Instream and have names that start with **PD**. For example, the GuidelinePlus for an 835 is called **PDSA835**. GuidelinePlus names are listed in a separate document, **ForesightHIPAAguidelinelist.pdf**.
Non-HIPAA X12	Guideline can be used for splitting without any special modifications.
EDIFACT	Validation results must have STRUS records Therefore: 1. Edit the guideline with Standards Editor, right-click on the UNH segment, and choose **DSR Mark/Unmark**. 2. Open **Dictionary Object**s and then **Segments** and put a DSR mark on the **UNB** and **UNG**. 3. Save the guideline and copy it to Instream's **Database** directory. 4. Use this one for validation when you want to split data.
Flat File	Guideline can be used for splitting without any special modifications.
XML	Guideline can be used for splitting without any special modifications.

Example: Using a GuidelinePlus to validate

This Windows command line has Instream validate an 837I document using GuidelinePlus PDSA837I:

```
HVInStream -i"C:\Files\EDI_Input.txt" -o"C:\Files\Results.txt" -gPDSA837I
```

Profile settings

You can specify a profile on the validation command line or with an API call. If you don't specify one, Docsplitter uses the default profile **$fsdeflt.apf** in Instream's **Bin** directory.

Tip: Do not change $fsdeflt.apf, which is overwritten during updates. Instead, save it to a new name when making profile changes.

Be sure that your profile is generating the proper detail results for Docsplitter. To do this, open the profile file and be sure that WT_NonCritical, WT_Error and WT_Fatal are set to 1. This tells Instream validation to write DTL records for errors and fatal errors:

```
[Warning Allow]
WT_Message=1
WT_NonCritical=1
WT_Warning=1
WT_Error=1
WT_Fatal=1
```

These should be set to 1 so they will be included in the validation detail file.

In the [Types Allows] section, be sure Type0=1.

In the [Warning Levels] section, be sure the severity values for 31990 and 31991 are set to 1 (for informational):

```
31022=3,4,8,7,024,X7,
31990=1,1,,,,,
31991=1,1,,,,,
```

In the [Detail Record Output] section, these items must be set to 1 in your APF file if you are using Docsplitter:

```
STRUS=1
STRUE=1
SVALU=1
GEN=1
```

Windows Tutorial

Validate and Split the Tutorial File

1. Go to Instream's **DemoData** directory and look at the EDI data in **837I_4010_H_ErrorEvenClms.txt**.

   This contains one 837I transaction set with one provider, one subscriber, and 10 claims. The claims are numbered from 1 to 10 and the even-numbered claims have errors.

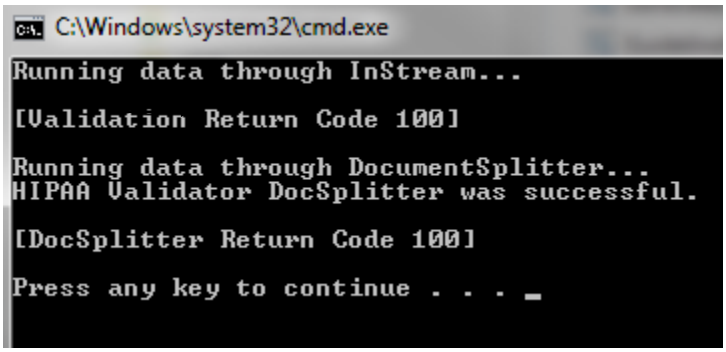
2. Validate and split the 837 data:

   Go to Instream's **Scripts** directory and double-click V_DS_837I_4010_EasyEdit.

   This batch first runs Instream validation with a GuidelinePlus, and then runs Docsplitter to separate the invalid data from the valid data.

   See page 35 for details about the script file format.

   When the batch file finishes, you should see the message **HIPAA Validator Docsplitter was successful**, plus return codes of 100 for validation and Docsplitter:

3. Using a text editor like **Notepad**, look at the files created in Instream's Output directory.

   Files created by the validation process (see **TIB_fsp-instream_*<n.n>*_usersguide.pdf** for file details):

   - **Summary_837I_4010_H_ErrorEvenClms_Results.txt**
     Validation summary report.
   - **837I_4010_H_ErrorEvenClms_Results.txt**
     Validation detail results file. Serves as input to Docsplitter.

Files created by Docsplitter:

- **837I_4010_H_ErrorEvenClms_Invalid.txt**
  EDI file containing the 5 invalid claims.
- **837I_4010_H_ErrorEvenClms_Report.xml**
  EDI report providing information about each claim, including whether it passed or failed.
- **837I_4010_H_ErrorEvenClms_Valid.txt**
  EDI file containing the 5 valid claims.

Unix Tutorial

Setting LIBPATH on Unix Operating Systems

AIX export LIBPATH=/HVInStream/bin:$LIBPATH

HP export SHLIB_PATH=/HVInStream/bin:$SHLIB_PATH

SUN LD_LIBRARY_PATH=/HVInStream/bin:$LD_LIBRARY_PATH;
export LD_LIBRARY_PATH

Validate and Split the Tutorial File

1. Go to Instream's **DemoData** directory and look at the EDI data in **837I_4010_H_ErrorEvenClms.txt**.

   This contains one 837I transaction set with one provider, one subscriber, and 10 claims. The claims are numbered from 1 to 10 and the even-numbered claims have errors.

2. Validate and split the data.

3. Go to Instream's **Scripts** directory and look at the contents of **V_DS_837I_4010_EasyEdit.sh**. Execute this script by typing its name.

   This script first runs Instream validation with GuidelinePlus PDSA837I, and then runs Docsplitter to separate the invalid data from the valid data.

   When the script finishes, you should see return codes and the message **Docsplitter was successful**.

   See pages 31 and 35 for details about the script file format.

   ---

4. Look at the following files created in Instream's **Output** directory.

   Files created by the validation process (see the **TIB_fsp-instream_*<n.n>*_usersguide.pdf** for file details):

   - **Summary_837I_4010_H_ErrorEvenClms_Results.txt**
     Validation summary report.
   - **837I_4010_H_ErrorEvenClms_Results.txt**
     Validation detail results file. Serves as input to Docsplitter.

Files created by Docsplitter:

- **837I_4010_H_ErrorEvenClms_Invalid.txt**
  EDI file containing the 5 invalid claims.
- **837I_4010_H_ErrorEvenClms_Report.xml**
  EDI report providing information about each claim, including whether it passed or failed. For XML report format, see page 38. For suggestions on how to view the Docsplitter XML report in Unix, see page 32.
- **837I_4010_H_ErrorEvenClms_Valid.txt**
  EDI file containing the 5 valid claims.

How Documents Split

Docsplitter creates a valid and an invalid EDI file, each with a complete EDI interchange and valid counters and totals.

The interchange can split at various places, depending on where errors are found. Split points are listed on page 43.

Example EDI file before splitting:

Start of first 837 →

Start of second func. group containing an 837 →

```
1→ ISA
2→   GS for Functional Group 1
3→     837's Header (Table 1 including 1000A and 1000B loops)
4→     837's Detail (Table 2)
5→         loop 2000A for Provider A
6→            segments before 2000B
7→            loop 2000B for Subscriber A
8→                segments before claim
9→                loop 2300 Claim 1 - valid claim
10→               loop 2300 Claim 2 - invalid claim
11→               loop 2300 Claim 3 - valid claim
12→           loop 2000B for Subscriber B
13→               segments before claim - invalid
14→               loop 2300 Claim 4 - valid claim
15→               loop 2300 Claim 5 - valid claim
16→               loop 2300 Claim 6 - valid claim
17→        loop 2000A for Provider B
18→           segments before 2000B
19→           loop 2000B for Subscriber C
20→               segments before 2000C
21→               loop 2000C for Patient A
22→                   loop 2300 Claim 7 - valid claim
23→               loop 2300 Claim 8 - invalid claim
24→                   loop 2300 Claim 9 - valid claim
25→    837's Summary
26→  GE
27→  GS for Functional Group 2
28→        ST segment
     .
29→    . - all segments valid
     .
30→    SE
31→    GE
32→ IEA
```

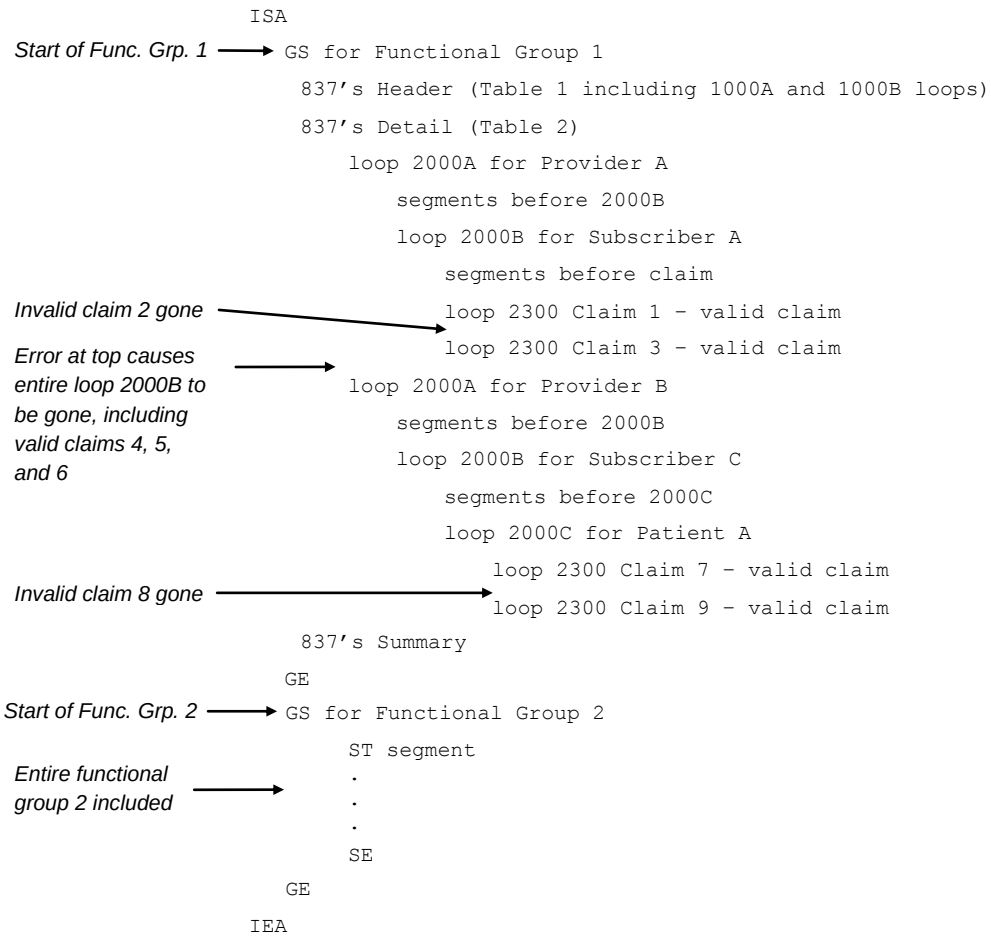
Explanation

Numbers correspond to pointers on page 11.

Number	Explanation
1, 32	ISA or IEA errors send the entire interchange to the invalid file. Some severe ISA errors cause validation to fail and so Docsplitter does not get a usable detail file to use for input.
2, 26, 27, 31	GS or GE errors send the entire functional group to the invalid file. Other error-free functional groups go to the valid file. In the example, an error at 26 would send 2 - 26 to the invalid file.
3, 25, 28, 30	Errors in Table 1 (ST, BHT, REF, Submitter name, Receiver name) or in Table 3 (SE) send the entire contents of that transaction set to the invalid file. Other error-free transaction sets go to the valid file. In the example, an error at 25 would send 3 – 25 to the invalid file.
6, 18	Errors in loop 2000A, before its first 2000B, send the entire contents of that iteration of loop 2000A to the invalid file. Other error-free 2000A loops in the same transaction go to the valid file.
13, 20	Errors in loop 2000B, before its first 2000C or claim, send the entire contents of that iteration of loop 2000B to the invalid file. Other error-free 2000B loops for the same provider go to the valid file.
9-11, 14-16, 22-24	Errors anywhere in a loop 2300 (claim loop) send the entire contents of that claim to the invalid file. Other error-free claims for the same dependent or subscriber go to the valid file. A claim loop is the smallest increment that can be split in an 837.

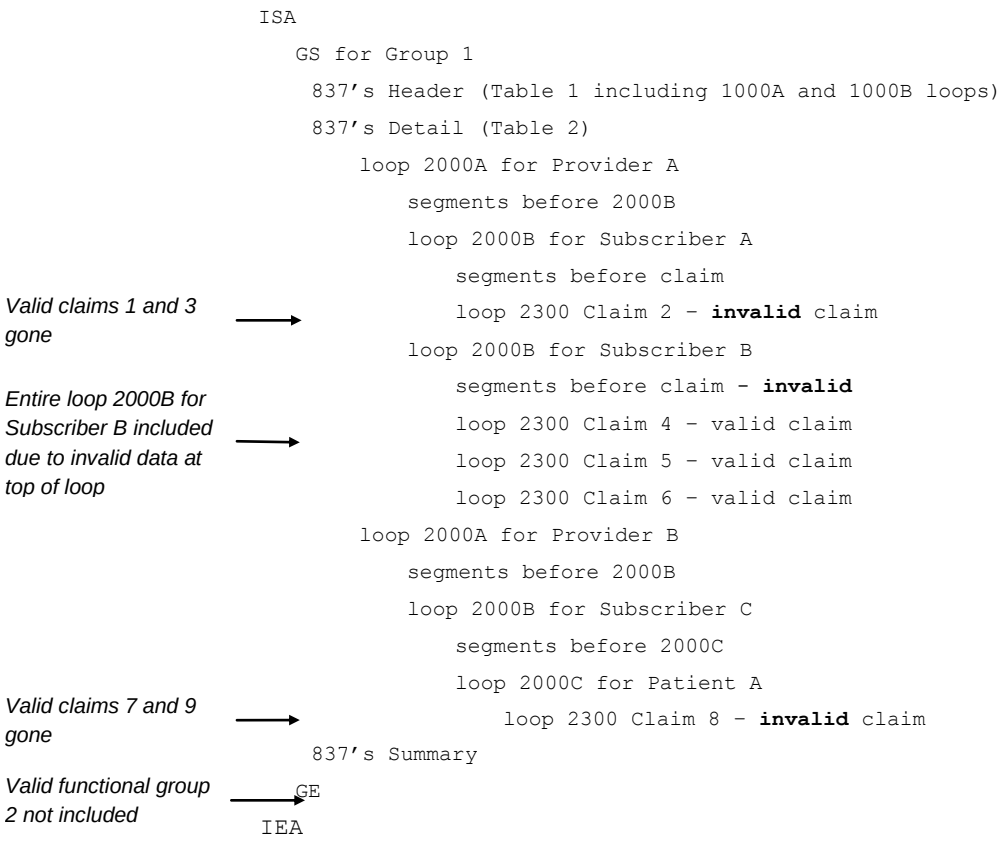
Assume that the only errors in the file are the ones circled at numbers 10, 13, and 23 on page 11. Docsplitter will create a valid EDI file like these:

Valid File



Assume that the only errors in the file are the ones circled at numbers 10, 13, and 23 on page 11. Docsplitter will create an invalid EDI file like these:

Invalid File



Executing Docsplitter

Command Line

Parameters

DocSplitter.exe is a standalone executable in Instream's **bin** directory. From the command line, it has the following case-sensitive parameters. Use quotation marks around paths or filenames that contain spaces.

Parameters	
-d	**d**ata file - path and filename of the corresponding data file. Required.
-i	**i**nput file - path and filename of an Instream validation detail results file. Required.
-oi	**o**utput invalid file - desired path and filename for the Docsplitter invalid EDI output. Directory must already exist. Required.
-ov	**o**utput valid file - desired path and filename for the Docsplitter valid EDI output. Directory must already exist. Required.
-r	**r**eport file - desired path and filename for the Docsplitter report file. Directory must already exist. Either -cd or -r is required.
-b	de**b**ug. Requests debugging information go to the specified file. Also requires DebugOutputToFile=1 in the setup file (see page 25). Example: `Docsplitter -b c:\logs\Rgdebug.log`

Parameters	
-cv **-ci** **-cb** **-cd**	**c**ontents of report: -cv Valid data -ci Invalid data -cb Both. Default -cd Disable. Do not create a report.
-fxp or **-fd**	**f**ormat for Docsplitter output report: -fxp XML (schema is DocSplitterReportPlus.xsd in Instream's Scripts and Output directories) -fd delimited by asterisks -fx old xml format that is no longer being updated (schema is DocSplitterReport.xsd in Instream's Scripts and Output directories) The XML report slows performance more than the delimited report. If omitted, no report is written.
-j	Archiver extraction file. Example: `-j"C:\Foresight\Instream\Bin\ArchiveExtract.txt"` This is only used by TI customers when they are retrieving archived files. Please see **TIB_fsp_archive_<*n.n*>_archiveadmin.pdf**.
-l	**l**og level for Docsplitter report file: 0 debug 1 no debug. Default.
-s	**s**etup file for Docsplitter (see Setup File on page 19). Example: `-s"C:\Foresight\Instream\Bin\DSsetup.ini"`
-TPA	Use trading partner automation to select a Docsplitter setup file. See **TIB_fsp-instream_<*n.n*>_tpa.pdf**. Example: `-TPA "C:\lookups\TPA_DS_RG.csv"`
-version	Displays Docsplitter's version (see Version Information below).

Parameters used when splitting with a 997 (see page 55)	
-e997	Error file; path and filename of a 997 or 999 (use -e997 for either)
-g	Guideline name without ".STD".

Parameters used when splitting and swapping (see page 81)	
-ai*"file"*	Specifies the invalid swap audit file. Do not use wildcards in the filename. If there are multiple invalid output files, this will report swapping in all of them. If -wi or -wb is not used, this is ignored.
-av*"file"*	Specifies the valid swap audit file. Do not use wildcards in the filename. If there are multiple valid output files, this will report swapping in all of them. If -wv or -wb is not used, this is ignored.
-wb	Swap data in both valid and invalid output file(s) according to the rules in the validation guideline.
-wi	Swap data in invalid output file(s) according to the rules in the validation guideline.
-wv	Swap data in valid output file(s) according to the rules in the validation guideline.
-z*"file"*	Identifies a *Dataswapper* INI file.

Version information

The **-version** parameter displays the Docsplitter version, as in this example:

```
"C:\Foresight\Instream\Bin\DocSplitter.exe" -version
```

To see the version and a list of command-line parameters, run Docsplitter without any parameters:

```
"C:\Foresight\Instream\Bin\DocSplitter.exe"
```

Execute the **Version** file in the Scripts directory to see versions for all Instream programs.

Command Line Examples

Assume these file names:

Validation detail results file:	HV_Results.txt
EDI input file:	EDI_Input.txt
Docsplitter XML report file:	Report.xml
Valid EDI file:	Valid_Output.txt
Invalid EDI file:	Invalid_Output.txt

The commands below create XML reports with information about valid and invalid claims and no debug logging.

Windows Example

All files are in the C:\Files directory.

```
"C:\Foresight\Instream\bin\DocSplitter.exe"
-i"C:\Files\HV_Results.txt" -d"C:\Files\EDI_Input.txt"
-r"C:\Files\Report.xml" -ov"C:\Files\Valid_Output.txt"
-oi"C:\Files\Invalid_Output.txt" -l1 -fxp -cb
```

Unix Example

All files are in the **/Files** directory.

```
export FSINSTREAMINI=/HVInStream/bin

export LIBPATH=/HVInStream/bin:$LIBPATH

/HVInStream/bin/DocSplitter -i"/Files/HV_Results.txt"
-d"/Files/EDI_Input.txt" -r"/Files/Report.xml"
-ov"/Files/Valid_Output.txt" -oi"/Files/Invalid_Output.txt"
-l1 -fxp -cb
```

Example using trading partner automation

See **TIB_fsp-instream_ <*n.n*>_tpa.pdf.**

Sample Batch or Script Files

Instream's Scripts folder has files that show Docsplitter and validation used together. See **Demo_Index.pdf** or look for scripts that start with **V_DS**.

Setup File

You can create a setup file to control Docsplitter behavior. It can be invoked with these command line parameters:

-s Points directly to a setup file. Settings in the [CommandLine Option] section are ignored. See Parameters on page 15.

-TPA Points to a trading partner automation CSV file that contains a pointer to a setup file. All sections in the setup file are processed. See **TIB_fsp-instream_*<n.n>*_.tpa.pdf**.

If there is a conflict between the command line and the settings in the setup file selected by TPA, the last setting processed will prevail.

By moving the -TPA to the end of the command line, you are assured of having its settings honored when there is a conflict:

The setup file can include any or all of these sections.

This section is processed only when you use the -TPA command line option	
[CommandLine Option]	**Effect on output**
ValidEdiOutputPathName=*file*	Name of the output file containing valid EDI. (Command line option -ov) Example (asterisk stands for the EDI filename): `ValidEdiOutputPathName=C:\out\*_good.txt`
InValidEdiOutputPathName=*file*	Name of the output file containing invalid EDI. (Command line option -oi) Example (asterisk stands for the EDI filename): `InValidEdiOutputPathName=C:\out\*_bad.txt`
LogLevel=*n*	Logging level for Docsplitter report file. (Command line option -l) 0 = debug 1 = no debug Example: `LogLevel=0`
ReportFormat=*x*	Report format (Command line option -f) x = xml d = Delimited xp = Generic XML Example: `ReportFormat=d`

This section is processed only when you use the -TPA command line option	
[CommandLine Option]	**Effect on output**
ReportFilePathName=*file*	Report file path and filename. Directory must already exist. (Command line option -r) Example (asterisk stands for the EDI filename): `ReportFilePathName=C:\out\*_Report.xml`
PerformDataSwapping=*setting*	Controls whether swapping will take place (see Appendix H: Split-and-Swap on page 81). *Setting* can be: v Swap data in valid file i Swap data in invalid file b Swap data in both This is equivalent to Docsplitter's -wv, wi, and -wb command line parameters. Example: `PerformDataSwapping=b`
ValidSwapAuditFile=*file*	Specifies the valid swap audit file. Do not use wildcards in the filename. This is equivalent to Docsplitter's -av command line parameter. Example: `ValidSwapAuditFile=c:\output\auditv.txt`
InvalidSwapAuditFile=*file*	Specifies the invalid swap audit file. Do not use wildcards in the filename. This is equivalent to Docsplitter's -a1 command line parameter. Example: `InvalidSwapAuditFile=c:\output\auditi.txt`
SwapSetupFile=*file*	Specifies the DataSwapper.ini file. This is equivalent to Docsplitter's -z command line parameter. Example: `InvalidSwapAuditFile=c:\setupDswap.ini`
ContentsOfReport	Type of data in the report. V valid I invalid B both D disable (suppress creation of report) This is equivalent to Docsplitter's -c command line parameter.

This section is processed when you use either -s or -TPA command line options	
[Options]	**Effect on output**

AddST02ToFilename =*n*	Do you want to add the first ST02 (Transaction Set Control Number) value encountered in the file to the output filename? This is useful if you wish to identify good/bad files by their associated ST02.
	0 = no (default)
	1 = yes
	Example 1:
	`AddST02ToFilename =0`
	results in a file named `TestFile.good`
	Example 2:
	`AddST02ToFilename =1`
	results in a file named `TestFile_0001.good`
	Where "`_0001`" refers to the first ST02 value encountered in the file.

CalcValidBPR02=*n* CalcInValidBPR02=*n*	(820s and 835s only). Do you want to recalculate the value in the BPR02 after splitting? These two options let you recalculate the valid file and/or the invalid file. 0 = no 1 = yes (default) If you split your 835s below the ST, you should consider recalculating the valid file's BPR02 to create a valid 835. If you split at the ST or above, you should consider not recalculating the invalid file's BPR02.
ConcatenateValidOutputFiles=*n* ConcatenateInValidOutputFiles=*n*	Do you want multiple valid and/or invalid files to be concatenated? 0 = no (default) 1 = yes

This section is processed when you use either -s or -TPA command line options	
[Options]	**Effect on output**
FilesPerRun=*n*	This option lets you reduce resource use on the Docsplitter machine. Docsplitter knows how many files its split will generate before it actually does the split. FilesPerRun determines how many of them will be open at once while it actually generates the files. A maximum of *n* valid and invalid files can be open at once. After that, they are processed and closed and then the next *n* valid and invalid files are processed. If you're doing large numbers of files per split, bigger numbers will be a little faster, but use more resources. FilesPerRun also lets you limit the number of open files per pass if your system will only let you open a relatively small number of files (which can happen on Unix). The end result will be the same regardless of the number you specify. If FilesPerRun is omitted, 100 is assumed. This affects content-based splitting and split-point grouping. It does not affect standard error-based splitting, which generates just one valid and one invalid file.
OutputEDIWithCRLF=*n*	Do you want a CR/LF to follow each segment terminator? 0 = no (wrap the output data) 1 = yes (default) See Input and Output on page 2.
CalcValidSE01 CalcInValidSE01	Do you want Docsplitter to recalculate the SE01 in the valid or invalid output file? 0 = no 1 = yes (default: recalculate)
CalcValidGE01 CalcInValidGE01	Do you want Docsplitter to recalculate the GE01 in the valid or invalid output file? 0 = no 1 = yes (default: recalculate)
CalcValidIEA01 CalcInValidIEA01	Do you want Docsplitter to recalculate the IEA01 in the valid or invalid output file? 0 = no 1 = yes (default: recalculate)
CalcValidHL CalcInValidHL	Do you want Docsplitter to recalculate the HL element values in the valid or invalid output file? 0 = no 1 = yes (default: recalculate)

This section is processed when you use either -s or -TPA command line options	
[Options]	**Effect on output**
CalcValidTS2 CalcInValidTS2	Do you want Docsplitter to recalculate the TS2 element values in the valid or invalid output file? 0 = no 1 = yes (default: recalculate)
CalcValidTS3 CalcInValidTS3	Do you want Docsplitter to recalculate the TS3 element values in the valid or invalid output file? 0 = no 1 = yes (default: recalculate)
RemoveZeroByteEDIFiles=*n*	Do you want to remove all empty files created by Docsplitter: 0 = no (default; leave empty files) 1 = yes
SaveDTLRecords=*n*	Do you want to include information about errors (from the detail file's DTL records) in the XML or delimited report? If this flag is on, Docsplitter's report file will contain error messages as well as values encountered. 0 = no (saves memory) 1 = yes (default)
SaveGENRecords=*n*	(TIBCO Foresight® Transaction Insight® internal use only)
SaveSVALURecords=*n*	Do you want to include EDI data (from the detail file's SVALU records) in the XML or delimited report? 0 = no (saves significant memory) 1 = yes (default)
SeparateAtSplitPoint=*n*	If this option appears in the setup file, Docsplitter uses split-point grouping, which sets a maximum number of split units per file. See Appendix F: Split-Point Grouping on page 73. *n* is the maximum number of units per file. This option cannot be used with [Content Splitting Map].

This section is processed when you use either -s or -TPA command line options	
[Options]	**Effect on output**
UseDocSplitterPlus =*n*	**Note:** High-Performance Parsing should be used for HIPAA EDI documents ONLY. When set to 1 (yes), this option specifies the use of High-Performance Parsing, which improves processing time on large files. Reporting information is not available with High-Performance Parsing. An empty report file is generated. 0 = no, use standard parsing (default) 1 = yes This option cannot be used with the command line parameter **–e997**. If **–e997** is specified, DocSplitter reverts to standard parsing.

This section is processed when you use either -s or -TPA command line options	
[Debugging]	**Displays debugging information while running from the command line.**
DebugOutputToFile=*n*	Do you want to redirect output to the file specified with the -b command line parameter? 0 = no (default) 1 = yes
TreeFinal=*n*	Do you want to display splitting details in a hierarchical format? 0 = no (default) 1 = yes Do not set this to 1 when executing Docsplitter in a workflow run by Automator. See Appendix G: Debugging on page 77.
Content=*n*	Do you want to display EDI values used for content-based splitting? 0 = no (default) 1 = yes See Appendix G: Debugging on page 77.
VariableSplitPoints=*n*	Do you want to display which data is splitting to each file for your variable split points? 0 = no (default) 1 = yes
ProcessNode=*n*	Do you want to display loop levels as they are processed? 0 = no (default) 1 = yes
ProcessRecord=*n*	Do you want to display detail file record IDs as they are processed? 0 = no (default) 1 = yes
ProgressStatus=*n*	Do you want to display checkpoint messages as they are processed? 0 = no (default) 1 = yes
ControlNumberStatus=*n*	Displays the control numbers of ISA, GS and ST segments after n segments have been processed. Example: ControlNumberStatus=5000 means that after segment 5000, any control numbers encountered in ISA, GS and ST segments will be displayed.

This section is processed when you use either -s or -TPA command line options	
[Split Point]	**Sets customized split points. See Changing the Split Point on page 49.**
All=ISA All=ISA/GS } One of these … *trans#=splitpoint* *trans#=splitpoint* } Or one or more of these	See Sample Setup File on page 30. Docsplitter finds the most specific match in the list, and splits other transactions at the location designated with ALL. The order in which these entries appear does not matter. Split points are listed on page 43.
ProcessBelowSplitPoint=*n*	Do you want to rebalance any of these in the split files: CalcValidSE01 CalcInValidSE01 CalcValidGE01 CalcInValidGE01 CalcValidIEA01 CalcInValidIEA01 CalcValidHL CalcInValidHL CalcValidBPR02 CalcInValidBPR02 CalcValidTS2 CalcInValidTS2 0 = no 1 = yes (default) – This causes Docsplitter to check the individual Calc settings. There is no corresponding command-line parameter.
Xml=*rootelement*|*splitpath*	Specifies the split point for XML data. See **XMLatForesight.pdf**. For an example, see DocSplitter_XML.ini in Instream's DemoData directory
This section is processed when you use either -s or -TPA command line options	
[Summary Point]	**Specifies the split summary element for XML data. See XMLatForesight.pdf**

XML	An element where Docsplitter is to put the number of splits in each output file. For an example, run V_DS_XML_split_PO in Instream's Scripts directory and look at the TOTAL element at the bottom of the valid output file. This one uses an element called TOTAL and shows that the file contains 9 splits:
	`- <TOTALLINES>`
	`  <TOTAL>9</TOTAL>`
	`  <TOTALAMOUNT>186</TOTALAMOUNT>`
	`  </TOTALLINES>`
	It uses DocSplitter_XML.ini, which contains this:
	`Summary Point]`
	`XML="<TOTAL>"`

This section is processed when you use either -s or -TPA command line options	
[Content Splitting Options]	**Determine which data is split when using content-based splitting. See page 61.**
AutoCreateCBSSplitFiles=*n*	Do you want Docsplitter to automatically create split files when ZCBS records are found? 0 = no – use the Content Splitting Map to decide how to split. 1 = yes This eliminates the need to specify particular values. See Appendix E: Content-Based Splitting on page 61 for details. This option cannot be used with [Content Splitting Map].
MergeValidOutput=*n*	When using content-based splitting, do you want all valid EDI data to go to one file? 0 = no – use content-based splitting on valid data too. 1 = yes
MergeInValidOutput=*n*	When using content-based splitting, do you want all invalid EDI data to go to one file? 0 = no – use content-based splitting on invalid data too. 1 = yes
ValidationSplitting=*n*	When using content-based splitting, do you want to split valid from invalid data also? 0 = no – just split based on content. 1 = yes

This section is processed when you use either -s or -TPA command line options	
[Content Splitting Map]	**Determine which data is split when using content-based splitting, and names its output file. See page 61.**

filename-end=|*content*|*content*|
filename-end.ext=|*content*|
Examples:
`HOSPITALA=|UN02 |`
`2GRANTS=|GRANTE |GRANTW |`
`HOSPITALB.OH=|OHIOGEN|`
	Which content causes a split and what file name ending shall it split to?
	In the examples:
	Content containing "`UN02 `" goes to a file ending with HOSPITALA.
	Content containing "`GRANTE `" or "`GRANTW `" goes to a file ending with 2GRANTS.
	Content containing "`OHIOGEN`" goes to a file ending with HOSPITALB.OH. File extensions specified on the –oi and –ov parameters are replaced with OH.
	Notice the vertical bars and trailing spaces. The *content* must be the fixed length defined by the DefineCustomRec business rule that wrote the ZCBS record in the detail results file. If in doubt, find any ZCBS record in a detail file produced by that guideline, and count the data plus trailing spaces.

This section is processed when you use either -s or -TPA command line options	
[AK3_Allow]	**Specify which error codes in the AK304 are considered warnings when doing 997-based splitting (see Customizing Docsplitter's 997 or** 999 Splitting **on page 59)**
code=0 *this code is considered a warning* *code*=0 *this code is considered a warning* or AK3=0 *all AK3 codes are considered warnings* Example: `6=0`	In this example, code 6 in the AK3 is considered a warning.

This section is processed when you use either -s or -TPA command line options	
[AK5_Allow]	**Specify which error codes in the AK502-AK506 are considered warnings when doing 997-based splitting.**
code=0 *code*=0 . . .	Similar to the AK3_Allow section.
This section is processed when you use either -s or -TPA command line options	
[AK9_Allow]	**Specify which error codes in the AK905-AK909 are considered warnings when doing 997-based splitting.**
code=0 *code*=0 :	Similar to the AK3_Allow section.

This section is processed when you use either -s or -TPA command line options	
[TA1_Allow]	**Specify which error codes in the TA105 are considered warnings when doing 997-based splitting. See Appendix C: Docsplitter Return Codes on page 55.**

code=0 (*this code is considered a warning*)
code=0 (*this code is considered a warning too*)
or
TA1=0 (*all TA1 codes are considered warnings*)
..
Example:
`010=0`
`011=0`
	In this example, codes 010 and 011 in the TA1 are considered warnings.

Sample Setup File

```
[Options]
OutputEDIWithCRLF=0

[Debugging]
TreeFinal=1
Content=1
VariableSplitPoints=1

[Split Point]
All=ISA/GS
835=ISA/GS/ST
837=ISA/GS/ST

[Content Splitting Options]
MergeValidOutput=0
MergeInValidOutput=0
ValidationSplitting=1

[Content Splitting Map]
HOSPITALA=RIVERSIDE01
HOSPITALB=UNIVERISTY02
HOSPITALC=GRANT03
HOSPITALD=DOCTORS04
HOSPITALE=MEMORIAL05

[AK3_Allow]
6=0

[AK5_Allow]
AK5=0

[TA1_Allow]
011=0
```

In this example, 835s and 837s will split at the ST and other transactions will split at the GS.

In this example, 835s and 837s will split at the ST and other transactions will split at the GS.

C++, C# and Java Application Program Interface

APIs for validation, Docsplitter, and Response Generator are described in **TIB_fsp-instream_<*n.n*>_api.pdf**.

Reporting

Generating a Docsplitter XML report

To generate a report from the command line, use Docsplitter's **-r**, **-f**, and **-c** parameters as described on page 15.

To generate a report from an API call, see **TIB_fsp-instream_<n.n>_api.pdf**.

Docsplitter XML reports are readable and easily parsed. Use your own XSLT processor to prepare it for browser display. The report is not configurable by you. For details, see page 35.

The data in the XML report is collected from the Instream results file.

Viewing the XML Report in Windows

See page 38 for information about interpreting XML reports.

For Windows, you can display the report hierarchically on the screen by double-clicking on the file in Windows Explorer. If this does not work for you, upgrade your Internet Explorer. You can display the report with a text editor like Notepad, but it will be on one line.

When displaying an XML report in Windows Explorer, you can collapse or expand the hierarchy by clicking on the minus signs in front of some tags:

```
-<Subscriber>
  <LastName>SMITH</LastName>
  <FirstName>MUFFY</FirstName>
  <MiddleName>M</MiddleName>
  <IDNumber>111222333</IDNumber>
  <PayerName>JONES</PayerName>
  <PayerID>43202</PayerID>
```

Viewing the XML Report in Unix

Since Docsplitter produces XML that is all on one very long line, you cannot display the report with **vi**. One way to view it is to use a tool like WC3's free Amaya, which lets you view XML.

Interpreting XML error and fatal error elements

Document Splitter XML reports show two different types of errors:

- Errors found during validation
- Fatal errors

These are described in the next two sections.

Errors Detected in Validation

See page 38 for a description of Error element attributes (like line, severity, loop_id, etc.).

An error reported by Instream validation shows up as an <Error> element in Docsplitter XML reports. It usually contains two child elements:

Message

Data

Example

```
<Error line="59" severity="3" loop_id="2300" seg_id="DTP"
elem_id="1250" seg_pos="56" elem_pos="2">
  <Message>Code Value "DD" was marked not used in code set
for DTP02 (D.E. 1250) at col. 9</Message>
  <Data>DD</Data>
  </Error>
```

Fatal Errors

In addition reporting on errors detected during validation, Docsplitter reports may include information about fatal errors, which are:

- Something unexpected or unrecoverable.
- Errors in trailer segments (SE, GE, or IEA). A trailer segment error invalidates the entire block of data to which it refers, regardless of what was already reported within the block.

A fatal error shows up as a Fatal element to the XML report. It usually contains child elements:

Message

BeginLine

EndLine

Example

```
<Fatal number="10917" line="239" severity="3">
  <Message>Transaction Set segment total 236 Incorrect - should be
237</Message>
  <BeginLine>9</BeginLine>
  <EndLine>238</EndLine>
  </Fatal>
```

See page 40 for details.

Appendix A: Docsplitter File Formats

Annotated Example Script Files

Example 1 – Windows Batch File

```
@echo off
set BinDir=C:\Foresight\Instream\Bin\
```
Must be the BIN directory under Instream's directory

```
set InputDir=C:\Foresight\Instream\DemoData\
```
Directory containing the EDI input text file

```
set OutputDir=C:\Foresight\Instream\DemoData\Output\
```
Directory where results will be written

```
set Guideline=PDSA835
```
GuidelinePlus for Instream to use

```
set EDIInput=%InputDir%835_Data.txt
```
Path and filename to the EDI Input file

```
set HVResults=%OutputDir%835_Results.txt
```
Path and filename to the validation detail results file

```
set Valid=%OutputDir%835_Valid.txt
```
Path and filename to the valid EDI output file

```
set Invalid=%OutputDir%835_Invalid.txt
```
Path and filename to the invalid EDI output file

```
set Report=%OutputDir%835_Report.xml
```
Path and filename to the Docsplitter report file

```
cd "%BinDir%"
```
Go to the Bin directory before executing

```
@echo Running data through Instream...
HVInStream.exe -i"%EDIInput%" -o"%HVResults%" -g%Guideline%
```
Execute Instream

```
@echo Running data through Docsplitter...

DocSplitter.exe -i"%HVResults%" -d"%EDIInput%" -r"%Report%" -
ov"%Valid%" -oi"%Invalid%" -l0 -fxp -cb
```

Execute Docsplitter, report valid and invalid claims, with debug logging, to an XML report

```
@echo Finished

pause
```

Example 2 – Windows Batch File

This file is designed so that you can make most changes at the top and not have to edit the entire file. Frequently changed places are circled below.

```
@echo off
rem InStreamRoot is set to root Instream dir during install
set InStreamRoot=E:\Foresight\Instream
set BinDir=%InStreamRoot%\Bin
set InputDir=%InStreamRoot%\DemoData
set OutputDir=%InStreamRoot%\Output
rem Input File assumed to have .txt extension
set RootFile=Tutorial837IA
set Guideline=PDSA837I
set EDIInput=%InputDir%\%RootFile%
set HVResults=%OutputDir%\%RootFile%_Results.txt
set Valid=%OutputDir%\%RootFile%_Valid.txt
set Invalid=%OutputDir%\%RootFile%_Invalid.txt
set Report=%OutputDir%\%RootFile%_Report.xml

@echo Running data through Instream...
"%BinDir%\HVInStream.exe" -i"%EDIInput%" -o"%HVResults%" -g%Guideline%

rem Test for Instream error (an ERRORLEVEL > 100 is an error)
if NOT ERRORLEVEL 101 goto noiserror
@echo.
@echo [ERROR %ERRORLEVEL%] There was a problem running HVInStream.exe.
@echo See the file %HVResults% for details.
@echo.
goto end

:noiserror
@echo Running data through DocumentSplitter...
"%BinDir%\DocSplitter.exe" -i"%HVResults%" -d"%EDIInput%" -r"%Report%"
-ov"%Valid%" -oi"%Invalid%" -l0 -fxp -cb

rem Test for Docsplitter error (an ERRORLEVEL > 100 is an error)
if NOT ERRORLEVEL 101 goto nodserror
@echo.
@echo [ERROR %ERRORLEVEL%] There was a problem running
DocSplitter.exe.
@echo See the file %Report% for details.
@echo.
goto end

:nodserror
@echo Finished - Results can be found in %OutputDir%

:end
pause
```

validation →

Docsplitter →

Example 3 – Windows Batch File

validation (all on one line)

```
"C:\Foresight\Instream\Bin\HVInStream.exe"
-i"C:\Foresight\Instream\DemoData\Tutorial837IA.txt"
-o"C:\Foresight\Instream\Output\Tutorial837IA_Results.txt"
-gPDSA837I
```

←①

Docsplitter (all on one line)

```
"C:\Foresight\Instream\Bin\DocSplitter.exe"                ←②
-i"C:\Foresight\Instream\Output\Tutorial837IA_Results.txt"←③
-d"C:\Foresight\Instream\DemoData\Tutorial837IA.txt"       ←④
-r"C:\Foresight\Instream\Output\Tutorial837IA_Report.xml"  ←⑤
-ov"C:\Foresight\Instream\Output\Tutorial837IA_Valid.txt"  ←⑥
-oi"C:\Foresight\Instream\Output\Tutorial837IA_Invalid.txt←⑦
-fxp -l1←⑧
pause
```

① Command to run validation. Put paths and filenames in quotation marks if they contain spaces.

② Command to run Docsplitter.

③ The detail results file written by validation now becomes input to Docsplitter.

④ The same EDI file used as input by validation is also input to Docsplitter.

⑤ The report file to be written by Docsplitter.

⑥ The valid EDI file to be written by Docsplitter.

⑦ The invalid EDI file to be written by Docsplitter.

⑧ **-fxp** means the report format is to be XML. The **-c**b means the report is to contain information about valid *and* invalid EDI. The **-l1** means no logging information should be placed in the report.

Example 4 – Unix Batch File

specify locations

validation (one line)

Docsplitter (one line)

```
#!/usr/bin/sh
export FSINSTREAMINI=/HVInStream/bin
export LIBPATH=/HVInStream/bin:$LIBPATH

/HVInStream/bin/HVInStream -i"/HVInStream/DemoData/Tutorial837IA.txt"
-o"/HVInStream/output/Tutorial837IA_Results.txt" -gPDSA837I

echo "validation return code = " $?

/HVInStream/bin/DocSplitter -
i"/HVInStream/output/Tutorial837IA_Results.txt"
-d"/HVInStream/DemoData/Tutorial837IA.txt"
-r"/HVInStream/output/Tutorial837IA_Report.xml"
-ov"/HVInStream/output/Tutorial837IA_Valid.txt"
-oi"/HVInStream/output/Tutorial837IA_Invalid.txt" -fxp -cb -l1

echo "Docsplitter return code = " $?
```

XML Report File Formats

Schemas

Docsplitter has two schemas that describe the XML reports that it can create.

Schema file	Location	Used with
DocSplitterReport.xsd	Instream's Scripts and Output directories **Deprecated**. This is no longer being updated but is still available for legacy purposes.	-fx command line parameter
DocSplitterReportPlus.xsd	Instream's Scripts and Output directories	-fxp command line parameter

Binary Data and Special Characters

Binary data and special characters in EDI files will appear in the Instream detail results file and will be read in by Docsplitter. Before being written to the XML file, they are converted so that the XML is correct.

Character	Converted to …
&	&amp;
\	&quot;
<	&lt;
>	&gt;
Hex	preceded with **&#x** and followed by **;** Example: Hex **1A** is converted to **&#x1A;**

Error Elements

See page 32 for more information about Error elements

Example error element

```
<Error number="10613" line="6" severity="3" type="1"
loop_id="1000A" seg_id="NM1" elem_id="1035" seg_pos="3"
elem_pos="3">
  <Message>Element NM103 (D.E. 1035) at col. 10 is missing, though
marked "Must Be Used"</Message>
  </Error>
```

Missing NM103 that caused the error

```
ISA*00*          *00*          *01*9012345720000 . . .
GS*HC*901234572000*908887732000*20030212*1615*386. . .
ST*837*0386!
BHT*0019*00*3920394930203*20030212*1615*CH!
REF*87*004010X096DA1!
NM1*41*1**BARBARA*T***46*9012345918341!
```

Error Attribute	Meaning
number	The Instream error number as listed in FSANErrs.txt, FSBRErrs.txt, or the customer error message file.

line	The number of the physical line in the EDI input file where the error was detected. It matches the DTL number in the validation detail results file. If data is wrapped, folded, etc., this may not correspond to segments. The line number does not start over anywhere in the EDI file.
	Example:
	XML file contains:
	`<Error line="4" …`
	Validation detail results file contains:
	`DTL 4 BHT1005 ...`
	EDI contains:
	`ISA*00* *00* ...`
	`GS*HC*901234572000* ...`
	`ST*837*0386!`
	`BHT**00*3920394930203* ...` ← *4th line has the error*

severity	The Instream severity number (1 - 7). See **TIB_fsp-instream_<*n.n*>_usersguide.pdf** for details about severity numbers. **Example:** `severity="3"` for a warning.
type	The WEDI/SNIP Type for the error. **Example**: `type="1"` for an X12 syntax integrity error.
loop_id	The identifier of the loop in effect when the error was detected. **Example**: `loop_id="1000A"`
seg_id	The identifier of the segment in effect when the error was detected. **Example**: `seg_id="NM1"`
elem_id	The element identifier corresponding to the error message. **Example**: `elem_id="1035"`
comp_id	The composite identifier corresponding to the error message.

Error Attribute	Meaning

seg_pos	The sequential number of the segment where the error was detected. Segment numbering starts at zero on the ST segment, and starts over with each transaction set.
	Example:
	XML file contains:
	`<Error line="4" ... seg_pos="1" ...`
	EDI input file contains:
	`ISA*00* *00* ...`
	`GS*HC*901234572000* ...`
	`ST*837*0386!`
	`BHT**00*3920394930203* ...`
	`REF*87*004010X096DA1!`
	segment with error

elem_pos	The position number of the element that caused the error. In the following example, elem_pos="3" if the error is in the circled element. NM1*41*1**BARBARA*T** . . .
comp_pos	The position number of the composite element that caused the error.
Child Element	**Meaning**
Message	Describes the error.
Data	Data that caused the error. May not be included for missing segments and other cases where data's absence caused the error.

Fatal Error Elements

Example Fatal Error Element

This fatal error was caused by a missing SE segment:

```
FATAL>
  <Message>Error occurred while creating Node Tree. Error
occurred while processing Record DTL 24 SE 21 0 11208 4 941-
102848 DocSplitter encountered an Envelope Level Error with
a Severity of Fatal. (SplitPattern.cpp, line 799).
(SplitPattern.cpp, line 278). (DocSplitter.cpp, line
287).</Message>
  </FATAL>
```

The bold text in the example corresponds to information in the DTL record, which is described in **TIB_fsp-instream_ <*n.n*>_usersguide.pdf**.

Delimited report files

To get an asterisk-delimited report instead of an XML report, use **–fd** on the Docsplitter command line (see page 15).

The report includes key information about the EDI document that was split. You can parse it with one of your application programs or bring it into a desktop application like a spreadsheet.

If an error is detected during validation, the delimited report will contain a record that starts with the word ERROR.

The next line is often an EMSG record that describes the error. If specific erroneous data is available, there will also be an EDAT record that shows the invalid data. These two records are simply echoed from validation's detail file.

For example, this segment in the EDI input file has an error:

```
CN1~01~HDG50.00~.10~12345678900987654321768958473~.25~VERSIO
N IDENTIFIER!
```

It might prompt these lines in a Docsplitter delimited report:

```
ERROR*10626*190*3*1*2300*CN1*782**187*2*
EMSG*Syntax Error for CN102 (D.E. 782) at col. 8 (R 1/10)
"HDG50.00": Real value required
EDAT*HDG50.00
```

The error record contains some or all of the attributes listed below, with delimiters maintaining the positions:

```
ERROR*10626*190*3*1*2300*CN1*782**187*2*
   ↑       ↑     ↑   ↑ ↑   ↑     ↑    ↑  ↑ ↑  ↑   ↑ ↑
   1       2     3   4 5   6     7    8  9 10 11 12
```

Where:

1	record tag	*5*	type	*9*	comp_id
2	number	*6*	loop_id	*10*	seg_pos
3	line number	*7*	seg_id	*11*	elem_pos
4	severity	*8*	elem_id	*12*	comp_pos

Delimited Fatal Records

A fatal error will look like this:

```
FATAL*Error occurred while creating Node Tree. Error occurred while processing Record
DTL          28        GS 480            2 8          0
10614 4    01 8 1007     122
DocSplitter encountered an Envelope Level Error with a Severity of Fatal.
```

The DTL record format is documented in **TIB_fsp-instream_<*n.n*>_usersguide.pdf**.

Appendix B: Split Points

Please see page 11 for an annotated example of how Docsplitter splits an 837I that has three errors.

Split Points for XML Files

Windows only

You can split XML data at any repeating complex element. The split point is controlled by the [Split Point] section of the Docsplitter INI file.

Please see Splitting XML data in **XMLatForesight.pdf** for details.

Split Points for Flat Files

Please see **V_DS_Ffdelim_vet** in Instream's Scripts directory for an example of flat file splitting.

The APF file used for validation must have these lines set to 1:

STRT=1
END=1

Flat files can split into valid and invalid files. The split point is at the top of the layout. In this example, the split point will be the NAME record:

This document will split at the bold lines:

```
HEAD*1*20070110*1412!
NAME*RITA*O'NEILL!
ADDR*115 CENTRAL AVE.**HILLSDALE#MN#12345!
PETS*SKYLER*DOG*YELLOW*2001!
PETS*JENNY*PARAKEET*BLUE!
PETS*CELESTE*CAT*BLACK WHITE*2002!
NAME*JAMES*WILSON!
ADDR*682 AUTUMN WOODS PLACE**HILLSDALE#MN#12345!
PETS*KAVER*DOG*BLACK*1998!
PETS*POPCORN*DOG*WHITE!
PETS*PIERRE*CAT*BEIGE*2005!
TRLR*1!
```

Assuming that the PETS record containing PARAKEET is in error, the resulting valid and invalid files would look like these:

Valid

```
HEAD*1*20070110*1412!
NAME*JAMES*WILSON!
ADDR*682 AUTUMN WOODS PLACE**HILLSDALE#MN#12345!
PETS*KAVER*DOG*BLACK*1998!
PETS*POPCORN*DOG*WHITE!
PETS*PIERRE*CAT*BEIGE*2005!
TRLR*1!
```

Invalid

```
HEAD*1*20070110*1412!
NAME*RITA*O'NEILL!
ADDR*115 CENTRAL AVE.**HILLSDALE#MN#12345!
PETS*SKYLER*DOG*YELLOW*2001!
PETS*JENNY*PARAKEET*BLUE!
PETS*CELESTE*CAT*BLACK WHITE*2002!
TRLR*1!
```

Split Points for X12 EDI

X12 EDI can split into valid and invalid data at the ST segment. By using a setup file (see page 19), you can change the split point to the GS or ISA.

Please see **V_DS_810_5040** in Instream's Scripts directory for an example.

Split Points for EDIFACT EDI

The APF file used for validation must have these lines set to 1:

STRT=1
END=1

EDIFACT data can be split into valid and invalid data or on the basis of content (see Appendix E: Content-Based Splitting on page 61).

Default split point is at the UNH. By using a setup file (see page 19), you can split at the UNB or UNG.

Please see these demo files in Instream's Scripts directory:

- **V_DS_CUSCAR_D93A_UNB**
- **V_DS_CUSCAR_D93A_UNH**

Split Points for HIPAA EDI

All Documents

For all transactions, Docsplitter can split at these points:

Interchanges

Functional Groups

Transaction Sets

270-271 Eligibility, Coverage or Benefit Inquiry

Loop or Element	Description
2000A	Information Source Level (HL)
2000B	Information Receiver Level (HL)
2000C	Subscriber Level (HL)
2110C	Subscriber Eligibility or Benefit [Inquiry] Information (EQ/EB)
2000D	Dependent Level (HL)
2110D	Dependent Eligibility or Benefit [Inquiry] Information (EQ/EB)

275 Health Care Claim or Encounter

Loop or Element	Description
ST	(Default split point for 275)

276-277 Health Care Claim Status Request/Notification

Loop or Element	Description
2000A	Information Source Level (HL)
2000B	Information Receiver Level (HL)
2000C	Service Provider Level (HL
2000D	Subscriber Level (HL)
2200D	Claim Submitter Trace Number (Sub. Patient) (TRN)
2000E	Dependent Level (HL)
2200E	Claim Submitter Trace Number (Dep. Patient) (TRN)

278 Health Care Services Request for Review

Loop or Element	Description
ST	(Default split point for 278)

820 Payment Order/Remittance Advice

Loop or Element	Description
2000A	Organization Summary Remittance (ENT)
2300A	Organization Summary Remittance Detail (RMR)
2000B	Individual Remittance (ENT)
2300B	Individual Premium Remittance Detail (RMR)

820-X306 Health Insurance Exchange: Related Payment

Loop or Element	Description
2000	Remittance Information
2300	Remittance Detail

834 Benefit Enrollment and Maintenance

Dependent information is not nested within a subscriber loop in the 834 guidelines. This means that there is the possibility of splitting subscriber from dependent if both are sent in the same 834. However, splitting an 834 could still create valid EDI because you are allowed to have dependent information without a subscriber if the subscriber information was sent in a previous transmission.

Loop or Element	Description
2000	Member Level Detail (INS)
2300	Health Coverage (HD)

834-X307 Health Insurance Exchange: Enrollment

Loop or Element	Description
2000	Member Level Detail
2300	Health Coverage

835 Health Care Claim Payment/Advice

Loop or Element	Description
2000	Header Number (LX)
2100	Claim Payment Information (CLP)

837D, 837I, 837P Health Care Claim

837 documents will split as low as the 2300 loop. They do not split at the 2400 loop.

Subscriber as Patient	
Loop or Element	**Description**
2000A	Billing/Pay-to Provider Hierarchical Level (HL)
2000B	Subscriber Hierarchical Level (HL)
2300	Claim Information (CLM)

Dependent as Patient	
Loop or Element	**Description**
2000A	Billing/Pay-to Provider Hierarchical Level (HL)
2000B	Subscriber Hierarchical Level (HL)
2000C	Patient Hierarchical Level (HL)
2300	Claim Information (CLM)

Changing the Split Point

If you don't want to use the default split points, you can change the lowest split point.

Example

The default split points for an 835 are loops 2000 and 2100. If the EDI contains this:

ISA
  GS
    ST
      2000 (1st)
        2100 – no errors
        2100 – errors
      2000 (2nd)
        2100 – no errors

With default split points, you will get these two files:

Valid EDI File

ISA
  GS
    ST
      2000 (1st)
        2100 – no errors
      2000 (2nd)
        2100 – no errors

Invalid EDI File

ISA
  GS
    ST
      2000 (1st)
        2100 – errors

If you want to reject an entire 2000 loop if it contains errors anywhere in it, including in the 2100 loops that it contains, you can set the split point at loop 2000. You will get these two files:

Valid EDI File

ISA
  GS
    ST
      2000 (2nd)
        2100 – no errors

Invalid EDI File

ISA
  GS
    ST
      2000 (1st)
        2100 – no errors
        2100 – errors

How to Change Split Points

Create a custom split point file

Create a Docsplitter setup text file that contains the changes in split points, like this:

```
[Split Point]
835=ISA/GS/ST/2000
837=ISA/GS/ST/2000A
834=ISA/GS/ST
```

Trans. set *split point path*

Where:

`[Split Point]`	Literal text.
`835`	Transaction set with changed split point. For 837s, this should be `837`. Do not use `837I`, `837P`, or `837D`. Within a single INI file, all 837s must have the same split points.
=	Literal text.
`ISA/GS/ST/2000`	Entire path to the new lowest split point for 835 guidelines (see List of Possible Split Points on page 52).
`834`	Another transaction set to have changed split point.
`ISA/GS/ST`	New split point path for 834 guidelines. This line says to send the whole transaction set to the invalid file if it contains an error anywhere.

Save the file to the name, type, and path of your choice – somewhere reachable by Docsplitter.

Splitting at the ISA or GS

To split at the ISA or GS, use **All** instead of the transaction set number:

All=ISA *All ISA's go to separate files*

or

All=ISA/GS *All GS's go to separate files*

Guidelines with multiple lowest split points

Certain guidelines might have two paths to the split point. For example, an 837 normally splits as low as the 2300 loop at either the subscriber level:

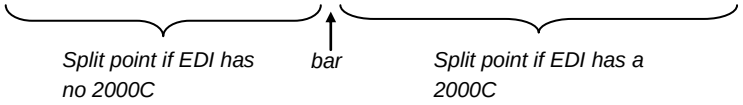
ISA/GS/ST/2000A/2000B/2300

…or the dependent level:

ISA/GS/ST/2000A/2000B/2000C/2300

If you want to split one level above the 2300 loop (2000C if present, or else the 2000B) put both split points in your split point file with a vertical bar between them:

837=ISA/GS/ST/2000A/2000B|ISA/GS/ST/2000A/2000B/2000C



The split points must be parallel structures like the 837 example above. The following is INVALID because it says to split at the ISA or the ST, which is within the ISA. Docsplitter would use the lower of the split points (ST).

270=ISA|ISA/GS/ST

Run Docsplitter with the -s parameter

Use the -s parameter on the command-line:

-s"*file*"

Where:

-s	Literal text (lower case)
"*file*"	Path and filename to the Docsplitter setup file that you just created. Surround the path with double quotes.

Example:

```
-s"C:\Foresight\Instream\Bin \DSsetup.ini"
```

The entire command-line might be:

```
"C:\Foresight\Instream\Bin\DocSplitter.exe"
- i"C:\Output\270Errors1A_Results.txt"
-d"C:\EDIdata\270Errors1A.txt"
-r"C:\Output\270Errors1A_ReportXML_Both.xml"
-ov"C:\Output\270Errors1A_EDIValid.txt"
-oi"C:\Output\270Errors1A_EDIInvalid.txt"
-s"C:\Foresight\Instream\Bin\DSsetup.ini" -l1 -fxp -cb
```

For more information about the setup file, see Setup File on page 19.

List of Possible Split Points

You can select from the following list of split levels to define the lowest point at which you want a certain type of document to split when errors are found.

All GuidelinePlus

ISA

ISA/GS

ISA/GS/ST

PDSA837I, PDSA837P, PDSA837D

ISA/GS/ST/2000A

ISA/GS/ST/2000A/2000B

ISA/GS/ST/2000A/2000B/2300

ISA/GS/ST/2000A/2000B/2000C

ISA/GS/ST/2000A/2000B/2000C/2300

PDSA270, PDSA271

ISA/GS/ST/2000A

ISA/GS/ST/2000A/2000B

ISA/GS/ST/2000A/2000B/2000C

ISA/GS/ST/2000A/2000B/2000C/2100C/2110C

ISA/GS/ST/2000A/2000B/2000C/2000D

ISA/GS/ST/2000A/2000B/2000C/2000D/2100D/2110D

PDSA276, PDSA277

ISA/GS/ST/2000A

ISA/GS/ST/2000A/2000B

ISA/GS/ST/2000A/2000B/2000C

ISA/GS/ST/2000A/2000B/2000C/2000D

ISA/GS/ST/2000A/2000B/2000C/2000D/2200D

ISA/GS/ST/2000A/2000B/2000C/2000D/2000E

ISA/GS/ST/2000A/2000B/2000C/2000D/2000E/2200E

PDSA820

ISA/GS/ST/2000A

ISA/GS/ST/2000A/2300A

ISA/GS/ST/2000B

ISA/GS/ST/2000B/2300B

PDSA834

ISA/GS/ST/2000

ISA/GS/ST/2000/2300

PDSA835

ISA/GS/ST/2000

ISA/GS/ST/2000/2100

Appendix C: Docsplitter Return Codes

Return Codes

Return Code		Meaning
FSDSSUCCESS	100	Docsplitter ran successfully.
FSDSFAILED	110	Docsplitter did not run successfully.
FSDSLIBFAILED	150	Docsplitter did not run successfully because it could not load one of its internal libraries. This is probably an installation problem. Contact TIBCO Foresight Technical Support.
FSDSINITFAILED	180	Docsplitter did not run successfully because of a problem other than those listed below. Contact TIBCO Foresight Technical Support.
DOCSP_FAILED	185	When running validation, Docsplitter, and Response Generator together in the API, Docsplitter failed.
RESGEN_FAILED	186	When running validation, Docsplitter, and Response Generator together in the API, Response Generator failed.
DOCSP_RESGEN_FAILED	187	When running validation, Docsplitter, and Response Generator together in the API, Docsplitter and Response Generator both failed.
FSDSEDIINPUT	200	Docsplitter could not access the EDI input file. Binary data was encountered in the EDI input file. Bad segment tag or other document structure problem.
FSDSEDIOUTPUT	201	Docsplitter could not open the EDI output file.
FSDSINSTREAMINPUT	202	Docsplitter could not open the Instream® detail file.
FSDSREPORT	203	Docsplitter could not open the report file.
FSDS997INPUT	204	Docsplitter could not open the input 997 file.
FSDSINIINPUT	205	Docsplitter could not open the INI config file.

Troubleshooting information	Notes
Debug file	Use -b command-line parameter to specify a debug report. Use DebugOutputtoFile=1 setup file parameter.
Report file	Use -r command-line parameter to specify the report.
Content splitting map	Use this in the setup file to see what values caused splitting: [Debugging] Content=1
Splitting tree	Use this in the setup file to see what content went to each output file: [Debugging] TreeFinal =1

Displaying Return Codes

To display return codes when you run a script, put this line similar to this in the script right after running the program:

UNIX echo "return code = " $?

Windows @echo [Return Code = %ERRORLEVEL%]

This returns something like: `[Return Code=100]`

Virus Checking and Foresight Products

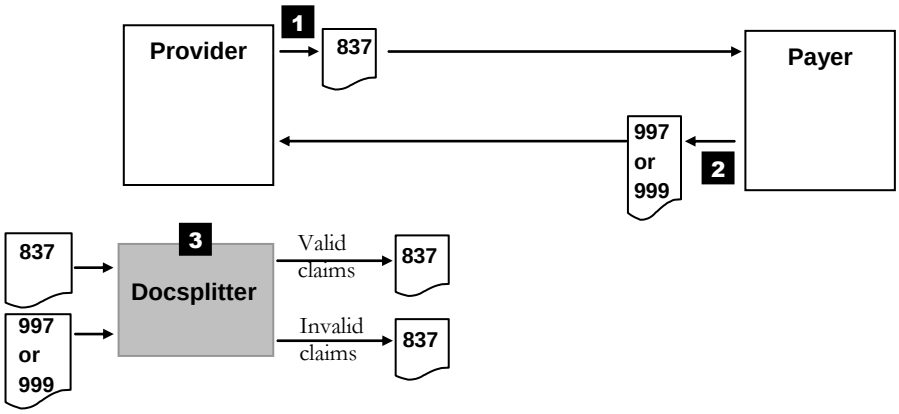
Exclude all TIBCO Foresight workflow subdirectories from virus checking.

Appendix D: Splitting from a 997 or 999

Overview

Docsplitter normally splits your EDI into valid and invalid files by using the errors found in the detail results file from Instream validation.

Instead, you can split EDI based on the error reporting in a 997 or a 999, as in this scenario:



1 The provider sends an 837 containing many claims to the payer.

2 The payer sends back a 997 or 999 describing errors in some of the claims.

3 The provider then uses the incoming 997 or 999 file to describe the errors when splitting the original 837 into valid and invalid claims.

Performing the Split using a 997 or 999

Command-line qualifiers for running Docsplitter with a 997 for error information:

-e997	**e**rrors are contained in the 997 or 999 file. Include the path and filename of a 997 or 999 file. (Use -e997 for either 997 or 999.) Required for splitting with a 997 or 999; not used otherwise.
-g	**g**uideline for the original data. Must be a GuidelinePlus (one that starts with PD) without the ".STD." Either **–g** or **–i** must be included when splitting with 997s or 999s.
-i	**i**nput file. The path and filename of an Instream validation detail results file. Either **–g** or **–i** must be included when splitting with 997s or 999s.

You **must** use these qualifiers as usual for Docsplitter: **-d**, **-r**, **-ov**, **-o***i*.

You **may** use these qualifiers as usual for Docsplitter: **-l**, **- fxp** or **-fd**, **-c, -s**

See page 15 for details about these qualifiers.

Examples

Assume these file names:

997 file	Their997.txt
EDI input file	Our837i.txt
Guideline	PDSA837I
Docsplitter XML report file	Our837i_Report.xml
Valid EDI file	Our837i_Valid.txt
Invalid EDI file	Our837i_Invalid.txt

The commands below split the claims into valid and invalid files based on the contents of the 997.

Windows Example

All files are in the **C:\Files** directory.

```
"C:\Foresight\Instream\Bin\DocSplitter.exe"
-e997"C:\files\Their997.txt" -d"C:files\Our837i.txt" -gPDSA837I
-r"C:\files\Our837i_Report.xml" -ov"C:\files\Our837i_Valid.txt"
-oi"C:\files\Our837i_Invalid.txt" -fxp -cb -l1
```

Demos of 997 and 999 Splitting

Demo in Instream's Scripts directory	Description
V_RG_DS_837I_5010_999split	Demonstrates Docsplitter 999 splitting.
V_RG_DS_837P_4010_997split	Demonstrates Docsplitter using 997 splitting.

Customizing Docsplitter's 997 or 999 Splitting

You can customize Docsplitter's 997- or 999-based splitting behavior by creating a Docsplitter setup file (see page 19). This file can override the default behavior for AK3, AK5, AK9, and TA1 segments and can specify what error codes in the 997 or 999 and TA1 are treated as warnings for Docsplitter.

Defaults:

- If the AK501 contains an R, the interchange is marked as an error.
- If the TA104 contains an R, the interchange is marked as an error unless the TA105 contains 000.

You can override these by specifying that certain codes are to be treated as warnings. These include codes in the:

AK304	List them in the [AK3_Allow] section of the Docsplitter setup file.
AK502	List them in the [AK5_Allow] section of the Docsplitter setup file.
AK905-AK909	List them in the [AK9_Allow] section of the Docsplitter setup file.
TA105	List them in the [TA1_Allow] section of the Docsplitter setup file.

Example setup file

0 means warning and 1 means error, as shown below.

```
[AK3_Allow]
6=0                <- in the AK3, error code 6 is a warning
[AK5_Allow]
5=0                <- in the AK5, error code 5 is a warning
8=0                <- and so is error code 8
[AK9_Allow]
AK9=0              <- in the AK9, all error codes are warnings
[TA1_Allow]
010=0              <- in the TA1, error code 010 is a warning
011=0              <- and so is error code 011
```

How Docsplitter uses the 997 or 999

The source of errors is the 997 or the 999 rather than the detail results file.

Docsplitter does not check the validity of the 997 or 999. It is assumed to be valid.

Docsplitter treats certain Error Codes in the AK5 and AK9 as warnings rather than errors so that it can split the valid application documents from the invalid ones. Otherwise, Docsplitter would have to handle the transaction sets and groups as unsplittable units.

Seg.	With this ACK code	AND this Error code	Docsplitter action
AK5	M, R, W, X	5 or *empty* (with no other codes)	Treats it as a warning; splits within transaction set; records WARN tag in output report
	A, E		Not an error; nothing in output report
AK9	M, R, W, X	*empty*	Considers it a warning; splits the functional group
	A, E, P		Not an error; nothing in output report

By treating the AK5 and AK9 error codes as warnings, Docsplitter can look at the AK3s to determine which chunks of data go to the valid file and which go to the invalid file.

Examples

Seg.	Docsplitter treatment
AK5*R~	Treated as warning, since the Acknowledgement code is R and the error code is empty. Docsplitter looks at the AK3s for this transaction set to determine what data goes to the valid file and what data goes to the invalid file.
AK5*R*5~	Treated as warning, since the Acknowledgement code is R and the error code is 5.
AK5*E*5~	Treated as warning, since the Acknowledgement code is E and the error code is 5.
AK5*R*6~	Treated as an error, since the error code is 6. The entire transaction set goes to the invalid file. Error codes other than 5 or empty indicate errors to the transaction set itself (for example, to the ST or SE segments or to security). It would be impossible for Docsplitter to write any valid EDI from this transaction set.
AK9*R*2*2*0~	Treated as warning, since the Acknowledgement code is R and there are no error codes. An error code in the AK9 would indicate an error in the GS or GE segments, meaning that no valid EDI data could be written.

Appendix E: Content-Based Splitting

Content-Based Splitting Demo

To see a demonstration of content-based splitting, go to Instream's **Scripts** directory and:

- Run **V_DS_837I_4010_ContentSplit1** to see an example of content-based splitting that uses a split map (see Creating a Splitting Map on page 66). Output will appear in Instream's **Output** directory.
- Run **V_DS_837I_4010_ContentSplit** for a demo of automatic content-based splitting (see Automatic Content-Based Splitting on page 66).
- Run **V_DS_837I_4010_NoContentSplit** to see how the same data splits without content-based splitting.

These examples split **837I_4010_H_5provider.txt** in Instream's **DemoData** directory. It contains five providers. Each provider has one subscriber with no dependent and one subscriber with a dependent. PROVIDER TWO has an error. Names are in alphabetic order and IDs are in numeric order so that you can easily check output.

The guideline used in the demos, PCBS, generates ZCBS records in the validation detail file. These records contain the first 9 characters of the provider IDs, and that value is used for the content-based splitting demo.

Overview

You will need EDISIM® to use Content-Based Splitting.

Docsplitter normally splits your EDI into valid and invalid files by using the errors found in the validation detail results file.

Content-based splitting lets you separate data into **multiple** valid and invalid files based on values in the input EDI file. The data splits at the same split points used for error-based splitting (see Appendix B: Split Points on page 43).

The following scenarios show splitting based on Provider ID, but you can flag any value from the EDI file for splitting. These are just a few of many possible scenarios.

Scenario 1: Separating provider A data

Docsplitter can separate Provider A data from other providers, so that you get these output files:

Validfile0	All valid data except for Provider A
Invalidfile0	All invalid data except for Provider A
Validfile1	All valid Provider A data
Invalidfile1	All invalid Provider A data

Scenario 2: Separating provider A data and provider B data

If you split both Provider A and Provider B into separate files, you get:

Validfile0	All valid data except for Providers A and B
Invalidfile0	All invalid data except for Providers A and B
Validfile1	All valid Provider A data
Invalidfile1	All invalid Provider A data
Validfile2	All valid Provider B data
Invalidfile2	All invalid Provider B data

Scenario 3: Separating provider A and B and combining in one file

You can also combine some providers. If you split both Provider A and Provider B into the same valid and invalid files, you get:

Validfile0	All valid data except for Providers A and B
Invalidfile0	All invalid data except for Providers A and B
Validfile1	All valid Provider A and B data
Invalidfile1	All invalid Provider A and B data

Setting up Content-Based Splitting

Configuring your APF file

Open the APF file you will be using and set these lines to 1:

In the Detail Record Output section

```
STRUS=1
STRUE=1
SVALU=1
```

In the Warning Allow section

```
WT_NonCritical=1
WT_Error=1
WT_Fatal=1
```

In the Types Allow section

```
Type0=1
```

The default APF file is **$fsdeflt.apf** in Instream's Bin directory. For more information, see **APF.pdf**.

Customizing a Guideline to write ZCBS Records

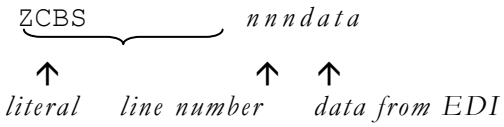
The purpose of this step is to create a guideline to be used for validation. During validation, the guideline will write ZCBS records to the detail results file. Each ZCBS record will contain the value from the EDI data. Docsplitter will look at this value to determine if it is one that causes a split.

This is a job for an experienced EDISIM user who can:

1. Add a custom record to a guideline with the BusinessRules.CustRec function. The record must be called **ZCBS**.
2. If this is a HIPAA guideline, merge it with a TIBCO Foresight-supplied PD guideline.

BusinessRules.pdf contains directions for creating business rules and merging the guidelines into one that Instream can use for validation.

The record will have this format when it is created in the validation results file:



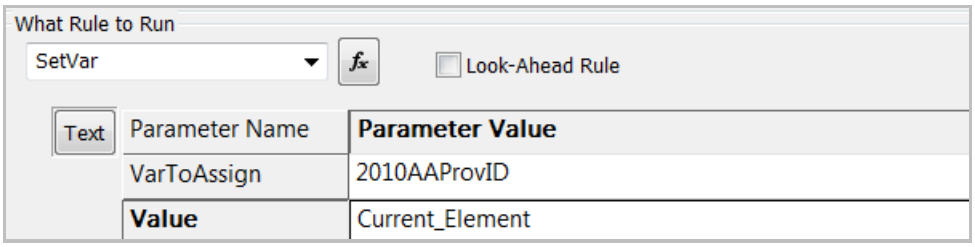
Example

This example creates a guideline that will place a ZCBS record in the Instream Validation Results file. The record will contain the value of the Provider ID (Loop 2010AA, NM1-09), for 837I data:

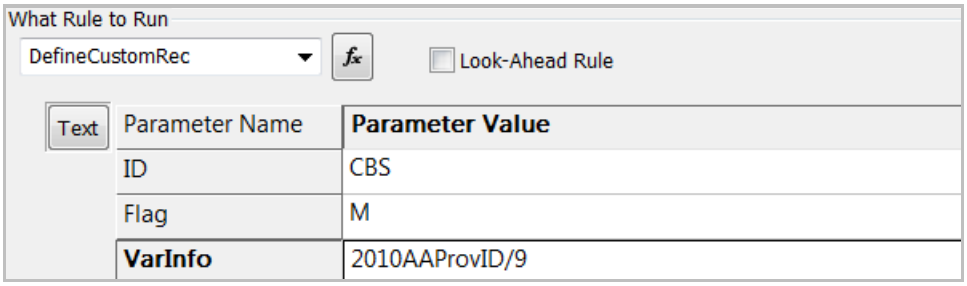
```
ZCBS          nnnProviderID
```

In EDISIM Standards Editor, create these business rules:

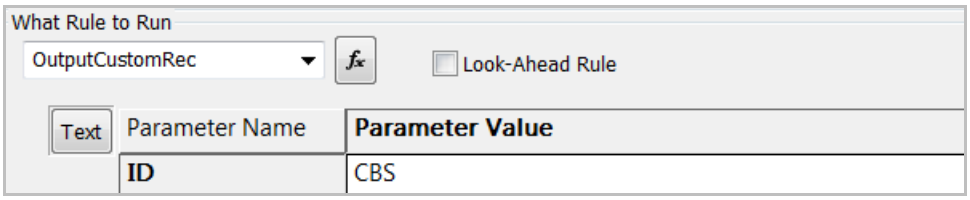
1. In a guideline based on a HIPAA 4010 837I addenda, go to 2010AA, NM1-09 and assign a variable for the Provider ID:



2. Go to the ST segment and define the ZCBS record (notice the leading Z is always omitted when defining the rule):

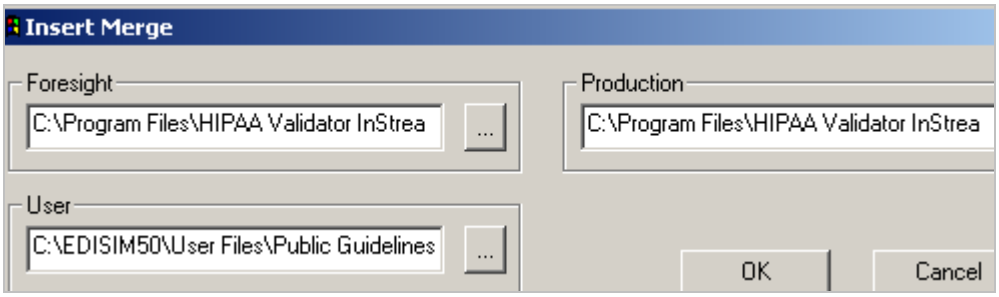


3. Go to the 2010AA NM1-09 element and output the record:



4. Save the guideline.

5. Merge it with PDSA837I to create a new production guideline for use with Instream. See **GuideMerge.pdf** for details.



6. Place the new production guideline in Instream's Database directory.

Editing the Setup File

The setup file (see page 19) controls what data causes a split and where the split data goes. It also allows options for displaying additional information during splitting and for suppressing splitting because of errors.

Automatic Content-Based Splitting

Demo:

V_DS_837I_4010_ContentSplit in Instream's Scripts directory
Content_Based_Split_Auto_Setup.ini in Instream's DemoData directory

A simple way to do content-based splitting is to let Docsplitter split at every unique value conveyed in the ZCBS records in the validation detail results file.

With this method, you will not have to create a split map and keep it updated (see next section).

This feature does not allow specific values to be mapped to specific output files; for that, you will need a split map.

To use automatic content-based splitting, use this in the setup file:

```
[Content Splitting Options]
AutoCreateCBSSplitFiles=1
```

This cannot be used with [Content Splitting Map].

Creating a Splitting Map

Demo:

V_DS_837I_4010_ContentSplit1 in Instream's Scripts directory
Content_Based_Split_Setup.ini in Instream's DemoData directory

The purpose of this step is to specify the literal values that cause a split and to tell Docsplitter the filename for the split data.

Add a [Content Splitting Map] section to a setup file (see Setup File on page 19). List each custom split point and a corresponding filename:

FilenameEnd=Splitdata|Splitdata|…

You can define any number of files, each corresponding to one line in the [Content Splitting Map] section of the setup file. Fewer lines in this section will result in faster performance.

Example Content Splitting Map

```
[Content Splitting Map]
Anderson=111111111
BTownDamascus=222222222|444444444
Edwards.EDI=555555555
```

`Anderson=111111111` specifies:

- When `111111111` appears in a ZCBS record, split there.
- Place the split data in files that ends with **`Anderson`**. The full filenames will start with the **–ov** and **–oi** parameters passed to Docsplitter:

With this command-line parameter …	Output files for data 111111111 will be …
-oi specified Provider_Invalid.txt	Provider_InvalidAnderson.txt
-ov specified Provider_Valid.txt	Provider_ValidAnderson.txt

`BTownDamascus=222222222|444444444` specifies:

- When either **`222222222`** or **`444444444`** appears in a ZCBS record, split there.
- Place the data for both values in files that ends with **`BTownDamascus`**. The full filenames will start with the **–ov** and **–oi** parameters passed to Docsplitter:

With this command-line parameter …	Output files for data 222222222 and 444444444 will be …
-oi specified Provider_Invalid.txt	Provider_InvalidBTownDamascus.txt
-ov specified Provider_Valid.txt	Provider_ValidBTownDamascus.txt

`Edwards.EDI=555555555` specifies:

- When **`555555555`** appears in a ZCBS record, split there.
- Place the data in files that ends with **Edwards.EDI**. The file extension will be EDI, regardless of what extension is given on the **–ov** and **–oi** parameters passed to Docsplitter:

With this command-line parameter …	Output files for data 555555555 will be …
-oi specified Provider_Invalid.txt	Provider_InvalidEdwards.EDI
-ov specified Provider_Valid.txt	Provider_InvalidEdwards.EDI

Spaces in the Content Splitting Map

If your content data contains leading or trailing spaces, set up the lines this way in the content splitting map:

`Anderson=111111111 |`	Vertical bar after the trailing spaces
`BTownDamascus=|  222222222|444444444`	Vertical bar before the leading spaces

Merging Output

These setup file options let you place all valid or all invalid data in one file when using content-based splitting.

```
[Content Splitting Options]
MergeValidOutput=n          If n=1, all valid data goes to the –ov filename
MergeInValidOutput=n        If n=1, all invalid data goes to the –oi filename
```

Example:

```
MergeValidOutput=1
MergeInValidOutput=0
```

This causes all valid output to go to one file. Invalid data is split into separate files as listed in the [Content Splitting Map] section.

Suppressing Splitting on Errors

This setup file option lets you turn off splitting by errors when you are splitting by content:

```
[Content Splitting Options]
ValidationSplitting=0
```

In this example, errors do not cause a split, and filenames start with the file name passed with the **–ov** parameter. The **–oi** parameter is ignored.

Validating and Splitting

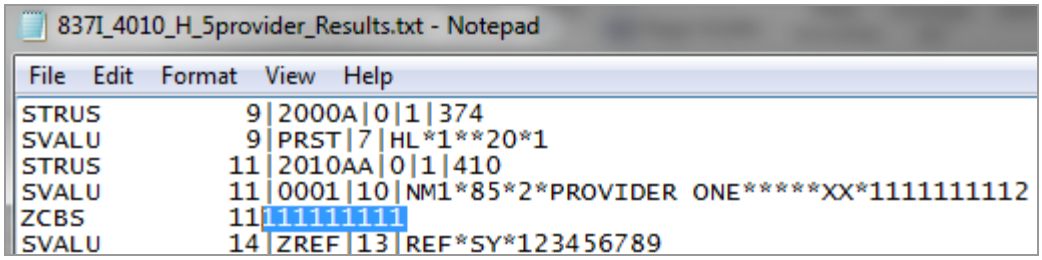
Validating for Content-Based Splitting

Validate with the guideline that will create the ZCBS records. The results file should contain ZCBS records with values from the EDI file.

Example command line

```
"C:\Foresight\Instream\Bin\HVInStream.exe"
 -i"C:\Foresight\Instream\DemoData\837I_4010_H_5provider.txt"
-o"C:\Foresight\Instream\Output\837I_4010_H_5provider_Results.txt"
-gPCBS
```

Example ZCBS record in the results file



The highlighted data is the value from the EDI file. It was found on line 11.

Running Docsplitter for Content-Based Splitting

For content-based splitting, use the **-s** parameter to point to the setup file that contains your splitting map.

Example:

```
"C:\Foresight\Instream\Bin\DocSplitter.exe"
-i"C:\Foresight\Instream\Output\Provider_Results.txt"
-d"C:\Foresight\Instream\DemoData\837I_4010_H_5provider.txt"
-r"C:\Foresight\Instream\Output\Provider_Report.xml"
-ov"C:\Foresight\Instream\Output\Provider_Valid.txt"
-oi"C:\Foresight\Instream\Output\Provider_Invalid.txt"
-s"C:\Foresight\Instream\DemoData\Content_Based_Split_Auto_Setup.ini"
-fxp -cb -l1
```

Complex Content-Based Splitting

You can have more than one ZCBS record in a guideline, thus requesting splitting on more than one element. For example, you can split on both the provider's state and the diagnosis code.

Use caution if you do this. You are likely to end up with some information duplicated in multiple files.

Debugging your Content-Based Splitting

These lines in the Docsplitter setup file (see page 19) display splitting values found in the EDI:

```
[Debugging]
Content=1
```

If set to 1	When Docsplitter runs, it displays each value it finds that causes a split, where the value was found, and what output file will get it. See the example below. (One common problem: the value in the ZCBS record has trailing spaces. You must account for these in the splitting map as shown in Spaces in the Content Splitting Map on page 68.)
If set to 0 or omitted	This information is not displayed.

Example

```
C:\Foresight\Instream\Scripts>"C:\Foresight\Instream\Bin\DocSplitter.exe"
-i"C:\Foresight\InStream\Output\Provider_Results.txt"
-d"C:\Foresight\Instream\DemoData\837I_4010_H_5provider.txt"
-r"C:\Foresight\Instream\Output\Proviider_Report.xml"
-ov"C:\Foresight\Instream\Output\Provider_Report_Valid.txt"
-oi"C:\Foresight\Instream\Output\Provider_Invalid.txt"
-s"C:\Foresight\Instream\DemoData\Content_Based_Split_Auto_Setup.ini "
-fxp -l1

Content Splitting Map
Mapping >111111111< to File1
Found Content >111111111< at Level 2000A Start Line 9 Outputting to File1
Found Content >222222222< at Level 2000A Start Line 57 Outputting to File0
Found Content >333333333< at Level 2000A Start Line 105 Outputting to File0
Found Content >444444444< at Level 2000A Start Line 153 Outputting to File0
Found Content >555555555< at Level 2000A Start Line 201 Outputting to File0
```

Value in content map, found in EDI data

Value was found in this line in the EDI file

Splits to this logical output file – see below

This Docsplitter output mentions the logical filenames **File1** and **File0**:

```
Content Splitting Map
Mapping >111111111< to File1
Found Content >111111111< at Level … Outputting to File1
Found Content >222222222< at Level … Outputting to File0
Found Content >333333333< at Level … Outputting to File0
```

The actual filenames depend on the [Content Splitting Map] section of the setup file. For these entries in the setup file

```
[Content Splitting Map]
Anderson=111111111                  ← "file1"
BTownDamascus=222222222|444444444   ← "file2"
Edwards.EDI=555555555               ← "file3"
                                      "file0" – everything else
```

file0 The logical name for the file containing the leftover data – the data that was not selected based on the content-based splitting map. It has the filename given in Docsplitter's **–oi** and **–ov** parameters.

file1 The logical name for the file created for data that splits due to the first value in the [Content Splitting Map] section of the setup file. In the example above, it will have **Anderson** at the end of the filename and contain data split because the value 111111111 was found in the EDI data.

file2 The logical name for the file created for data that splits due to the second value in the [Content Splitting Map] section of the setup file. In the example above, it will have **BTownDamascus** at the end of the filename and contain data split because the value 222222222 or 444444444 was found in the EDI data.

You can define any number of files, each corresponding to one line in the [Content Splitting Map] section of the setup file.

Appendix F: Split-Point Grouping

Setting a Maximum Number of Split Units per File

Demo

V_DS_837I_4010_SetupFile_DelimReport in Instream's Scripts directory demonstrates split-point splitting.

The original EDI file, **837I_4010_H_6claims.txt**, has the structure shown in Split-Point Grouping Examples on page 75.

The setup file used, **SampleSetupDS1.ini**, is in Instream's Bin directory.

You can specify a maximum number of interchanges, transactions, or parts of transactions (split points) that can go in Docsplitter valid and invalid files.

Examples:

- You may have an EDI file with 5000 claims and the maximum that you can handle is 100 per file. This allows you to create separate files of 100 claims each.
- You may have an EDI file containing multiple transactions. This allows you to create a separate file for each transaction.

Setting up Split-Point Grouping

Note: SeparateAtSplitPoint cannot be used with [Content Splitting Map].

Put this in your Docsplitter setup file (see page 19).

```
[Options]
SeparateAtSplitPoint=n
```

Where:

n Maximum number of split data units to put in a single file. This number does not change the enveloping structure. Each output claim will be in an envelope that matches its original envelope.

Output Filenames

The original EDI file splits into separate files with names in this format:

*filename*_Valid_File*n*.txt
*filename*_Invalid_File*n*.txt

Where:

filename the original EDI filename without the file type.

n an incrementing number.

Location of Split

If you specify a split point in the [Split Point] section of the Docsplitter setup file (see page 19), that would be where the splitting occurs.

Otherwise, it would take place at the default split point for the document. These are listed in the split point tables on page 43.

Setup Options frequently used with Split-Point Splitting

These can go in the setup file's [Options] section:

`ConcatenateValidOutputFiles=1`	After creating all of the individual valid files, Docsplitter concatenates them. This will give multiple ISAs in the same file.
`ConcatenateInValidOutputFiles=1`	After creating all of the individual invalid files, Docsplitter concatenates them.
`RemoveZeroByteEDIFiles=1`	This automatically deletes empty files created by Docsplitter.

Split-Point Grouping Examples

Assume that you want to split up the data at the claim level and create EDI files with a specific number of claims per file.

Since the default split point in an 837 is the claim loop, you do not have to specify the split point in the Docsplitter setup file.

Your EDI data file has this structure:

```
ISA1
    GS
        ST
            Claim 1 - valid
            Claim 2 - valid
            Claim 3 - invalid
            Claim 4 - invalid
            Claim 5 - invalid
        SE
    GE
IEA
ISA2
    GS
        ST
            Claim 6 - invalid
        SE
    GE
IEA
```

Example 1: One Claim per File

Assume that you want to split up the data at the claim level and create EDI files with just 1 claim per file.

In your Docsplitter setup file, you include this:

```
[Options]
SeparateAtSplitPoint=1
```

In our example, claims would be separated into these files:

Claim 1	(valid)	*filename*_Valid_File1.txt	*wrapped in ISA1*
Claim 2	(valid)	*filename*_Valid_File2.txt	*wrapped in ISA1*
Claim 3	(invalid)	*filename*_Invalid_File1.txt	*wrapped in ISA1*
Claim 4	(invalid)	*filename*_Invalid_File2.txt	*wrapped in ISA1*
Claim 5	(invalid)	*filename*_Invalid_File3.txt	*wrapped in ISA1*
Claim 6	(invalid)	*filename*_Invalid_File4.txt	*wrapped in ISA2*

Example 2: Two claims per file

Assume that you want to split up the data at the claim level and create EDI files with 2 claims per file.

In your Docsplitter setup file, you include this:

```
[Options]
SeparateAtSplitPoint=2
```

In our example, claims would be separated into these files:

Claim 1 (valid) Claim 2 (valid)	*filename*_Valid_File1.txt	*Both wrapped in ISA1*
Claim 3 (invalid) Claim 4 (invalid)	*filename*_Invalid_File1.txt	*Both wrapped in ISA1*
Claim 5 (invalid) Claim 6 (invalid)	*filename*_Invalid_File2.txt	*Claim 5 wrapped in ISA1* *Claim 6 wrapped in ISA2*

Appendix G: Debugging

Debugging Overview

You can display debugging information in the command window while executing Docsplitter:

A

```
Content Splitting Map
Mapping >FGem       < to File1
Mapping >FCarets    < to File2
Mapping >AAMHS      < to File3
Mapping >CCCarets   < to File4
Mapping >Other      < to File5
Found Content >AAMHS      < at Level 2300 Start Line 30 Outputting to File3
Found Content >FGem       < at Level 2300 Start Line 95 Outputting to File1
Found Content >Skyler     < at Level 2300 Start Line 162 Outputting to File0
Found Content >Other      < at Level 2300 Start Line 213 Outputting to File5
Found Content >CCCarets   < at Level 2300 Start Line 265 Outputting to File4
Found Content >FCarets    < at Level 2300 Start Line 310 Outputting to File2
Found Content >AAMHS      < at Level 2300 Start Line 363 Outputting to File3
Unique Content 1=|AAMHS      |
Unique Content 2=|CCCarets   |
Unique Content 3=|FCarets    |
Unique Content 4=|FGem       |
Unique Content 5=|Other      |
Unique Content 6=|Skyler     |
```

B

```
Start Node Tree FINAL
C:\Foresight\Instream\DemoData\Training-multi.txt

[Content Splitting Options]
ValidationSplitting=ON

   Level               Start -   End           0  1  2  3  4  5
+<   ISA>             (    1 -   420) I       I- I- I- I- I- I-
 +<    GS>             (    2 -   419) I       I- I- I- I- I- I-
  +<    ST>            (    3 -   418) I       I- I- I- I- I- I-
   +< 1000A>           (    6 -     7) I       I- I- I- I- I- I-
   +< 1000B>           (    8 -     8) I       I- I- I- I- I- I-
   +< 2000A>           (    9 -   417) I   E   I- I- I- I- I- I-
    +< 2000B>          (   20 -    84) I       -- ** ** I- ** **
     +<  2300>         (   30 -    84) I  S  C -- ** ** I- ** **
    +< 2000B>          (   85 -   151) I       -- I- ** ** ** **
     +<  2300>         (   95 -   151) I  S  C -- I- ** ** ** **
```

A To display information like **A** above, showing what values in the ZCBS record caused a split to what file, see Displaying Splitting Content on page 78.

B To display a map like **B** above, showing what levels of EDI split to what files, see Displaying Tree Information during Splitting on page 79.

Displaying Splitting Content

These lines in the setup file (see page 19) display values in the ZCBS record that caused splits:

[Debugging]
Content=1

Example

```
Content Splitting Map
Mapping >FGem      < to File1
Mapping >FCarets   < to File2
Mapping >AAMHS     < to File3
Mapping >CCCarets  < to File4
Mapping >Other     < to File5
Found Content >AAMHS     < at Level 2300 Start Line 30 Outputting to File3
Found Content >FGem      < at Level 2300 Start Line 95 Outputting to File1
Found Content >Skyler    < at Level 2300 Start Line 162 Outputting to File0
Found Content >Other     < at Level 2300 Start Line 213 Outputting to File5
Found Content >CCCarets  < at Level 2300 Start Line 265 Outputting to File4
Found Content >FCarets   < at Level 2300 Start Line 310 Outputting to File2
Found Content >AAMHS     < at Level 2300 Start Line 363 Outputting to File3
Unique Content 1=|AAMHS     |
Unique Content 2=|CCCarets  |
Unique Content 3=|FCarets   |
Unique Content 4=|FGem      |
Unique Content 5=|Other     |
Unique Content 6=|Skyler    |
```

The lines that start with **Mapping** show the values that will cause a split, according to the [Content Splitting Map] section of the setup file. Note the trailing spaces. In the first Mapping line above, the content splitting map is looking in the validation detail file for a ZCBS record that contains the characters `FGem` followed by exactly six spaces.

The lines that start with **Found Content** show where the content was found and what files will get the split EDI. In the example above:

- Two 2300 loops were found with a ZCBS record containing `AAMHS` followed by five spaces; these two loops were split to File3.
- One 2300 loop was found with a ZCBS record containing `FGem` followed by six spaces and sent to File1, and so on.
- Skyler was sent to File0 since the content splitting map did not contain an entry for Skyler.

The lines that start with **Unique Content** contain a compact list of the content found in ZCBS records for this EDI file.

When having problems with content-based splitting, be sure that the values in the Mapping lines exactly match those in Found Content, including the trailing spaces.

In this example, FGem has no trailing spaces in the content splitting map, but it does when found in the ZCBS record:

```
Mapping >FGem< to File1
Mapping >AAMHS     < to File3
Found Content >AAMHS     < at Level 2300 Start Line 30 Outputting to File3
Found Content >FGem      <at Level 2300 Start Line 95 Outputting to File1
```

Displaying Tree Information during Splitting

This feature is good for debugging but not for production workflows using Automator.

These lines in the setup file (see page 19) display a chart that shows what files received data from each split level:

```
[Debugging]
TreeFinal=1
```

Example

```
C:\Foresight\Instream\Scripts>C:\Foresight\InStream\Bin\DocSplitter.exe
-i"C:\Foresight\Instream\Output\Provider_Results.txt"
-d"C:\Foresight\Instream\DemoData\MultProvider837I.txt"
-r"C:\Foresight\Instream\Output\MultProvider_Report.xml"
-ov"C:\Foresight\Instream\Output\MultProvider_Valid.txt"
-oi"C:\Foresight\Instream\Output\MultProvider_Invalid.txt"
-s"C:\Foresight\Instream\Bin\DSSetup.ini -l1
```

```
Start Node Tree FINAL
C:\Foresight\Instream\DemoData\M
   Level               Start -   End           0  1
+<   ISA>             (    1 -   251) IV      IV -V
 +<    GS>             (    2 -   250) IV      IV -V
  +<    ST>            (    3 -   249) IV      IV -V
   +< 1000A>           (    6 -     7) IV      IV -V
   +< 1000B>           (    8 -     8) IV      IV -V
   +< 2000A>           (    9 -    56)  V    C -- -V
    +< 2000B>          (   15 -    32)  V      -- -V
     +<  2300>         (   22 -    26)  V S    -- -V
   +< 2000A>           (   57 -   104) IV    C IV --
    +< 2000B>          (   63 -    80)  V      -V --
     +<  2300>         (   70 -    74)  V S    -V --
    +< 2000B>          (   81 -   104) I       I- --
     +< 2000C>         (   88 -   104) I   E   I- --
      +<  2300>        (   94 -    98) I  S    I- --
      +<  2300>        (   99 -   104) I  S    I- --
        ↑                    ↑         ↑       ↑
        ①                    ②         ③       ④
```

Understanding Tree Information

This Docsplitter output shows which files get data from each possible split level

① ID of each loop where a split can occur.

② Lines in the EDI file that are included if a split occurs at that point.

③ **I** = This data goes to an invalid file (without considering content-based splitting).

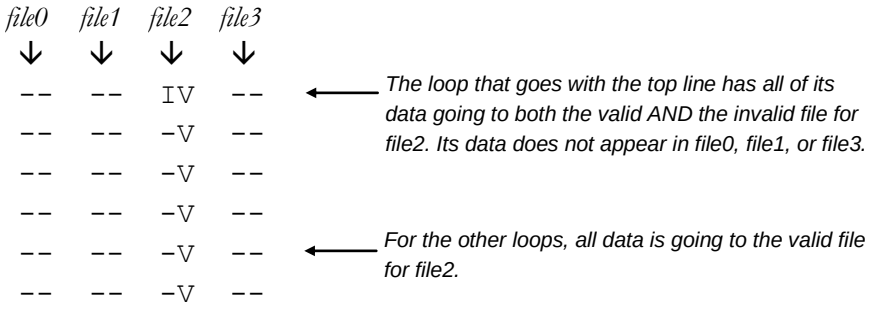
V = This data goes to a valid file (without considering content-based splitting).

S = Actual split point.

E = Error found at this level.

C = For content-based splitting only. Content was found at this level. See Appendix E: Content-Based Splitting on page 61.

④ For content-based splitting only. This section has a column for each logical file in this particular instance of content-based splitting. This example has four columns to show information for file0, file1, file2, and file3 (see Understanding filenames in the Content Section on page 71).



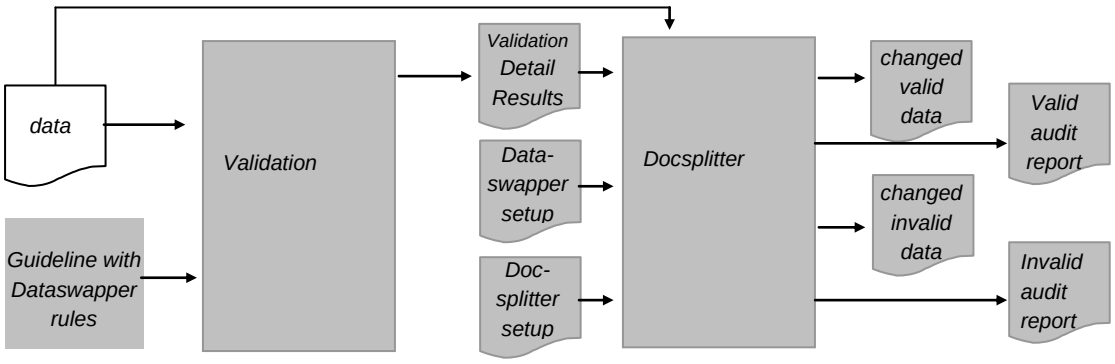
Appendix H: Split-and-Swap

Split-and-Swap Overview

You can use Docsplitter to:

- Split data.
- Then automatically run Dataswapper to change the output data.

Any type of split can have a simultaneous swap in it.



This is a convenient alternative to running Docsplitter, revalidating its output files to get new detail files that correspond to the split data, and feeding the output files and their new detail files into Dataswapper.

See the demo V_DS_Swap_837I in Instream's Scripts directory.

Docsplitter Command Line to Split and Swap

Command-line parameters specifically for split-and-swap		
Param.	**Description**	**Corresponding Setup file entry (see page 19)**
-wv	Swap data in valid output file(s) according to the rules in the validation guideline.	PerformDataSwapping=v
-wi	Swap data in invalid output file(s) according to the rules in the validation guideline.	PerformDataSwapping=i
-wb	Swap data in both valid and invalid output file(s) according to the rules in the validation guideline.	PerformDataSwapping=b
-av*"file"*	Specifies the valid swap audit file. Do not use wildcards in the filename. If there are multiple valid output files, this will report swapping in all of them. If -wv or -wb is not used, this is ignored.	ValidSwapAuditFile
-ai*"file"*	Specifies the invalid swap audit file. Do not use wildcards in the filename. If there are multiple invalid output files, this will report swapping in all of them. If -wi or -wb is not used, this is ignored.	InvalidSwapAuditFile
-z*"file"*	Identifies a Dataswapper INI file.	SwapSetupFile

Example

This takes in data and its validation results file, splits the data into valid and invalid data, changes the data according to the rules in the validation guideline (which are reflected in the validation results file), and creates two audit files to show what was changed. It uses an INI file to control the swapping.

```
C:\Foresight\Instream\Bin\DocSplitter.exe" -i"C:\results\File1_Results.txt"
-d"C:\in\File1.txt" -r"%InStreamRoot%\Output\DS_Report.xml"
-ov"C:\out\File1_Valid.txt" -oi" C:\out\File1_Invalid.txt" -l0 -fxp -cb -wb
-av" C:\out\File1_ValidAudit.txt" -ai" C:\out\File1_InvalidAudit.txt"
-z"C\INI\DataSwap.ini"
```

Docsplitter Setup Files for Split and Swap

The -TPA command line parameter identifies a trading partner automation CSV file.

This CSV file can point to a Docsplitter setup file that contains split and swap settings plus other settings. These are alternatives to the command-line parameters listed on page 82.

Docsplitter setup files are explained on page 19.

Dataswapper Setup Files for Split and Swap

With the Docsplitter -z parameter, you can use any Dataswapper setup file during a split-and-swap (see **TIB_fsp-instream_ <*n.n*>_dataswapper.pdf**). Docsplitter ignores everything in the file except these:

Dataswapper INI file settings used with split-and-swap	
InsertSegmentsAfter	(in the [Options] section) If **1**, insert new segments *after* the triggering segment. If **0** (the default), insert new segments *before* the triggering segment. Corresponds to Dataswapper command line option **-a** Example: InsertSegmentsAfter=1
MissingReplaceValue=0 MissingReplaceValue=1	(in the [Suppress] section) Do you want to continue processing if a no value is found to use for the replace? This can happen if Dataswapper was told to replace an element but was never given a value to use instead. The SubstituteReplace rule may be on an element that was not included in the EDI data. 0 = no (default). Stop processing the file and give a return code of 206 and an explanatory message. 1 = yes, continue processing the file.

Appendix I: Large Files

Large File Problems

The validation detail file can be many times the size of the original data. For example, a 300 MB data file might produce a 1 GB validation results file. A 1 GB validation results file might require 12-16 GB of memory to be available to Docsplitter at the time it processes the file.

These numbers vary, depending on the number of errors detected in the original file and the looping structure of the transaction.

Large File Solutions

- Contact your system administrator to add more memory or allocate more memory to Docsplitter. Please see the recommendations in the readme file provided with Instream.
- Restructure your workflows to break apart your large files. A recommended solution is to set up a separate workflow for large files. TIBCO's TPARouter has the ability to route based on size. This allows you to process the large files where and when you have enough resources.
- In Docsplitter's setup file, set SaveDTLRecords and SaveSVALURecords to 0 if you don't use the Docsplitter report.