XTranslator guide on complex flat text file format translation

Main website http://www.etasoft.com
XTranslator website http://www.xtranslator.com

Basic requirements

This document describes the process of mapping complex flat text file for translation. In order to do the mapping you should have:

1. If you want to produce some complex flat text file, then you need source file or database that will be used to produce flat text file. If you want to read complex flat text file then you need to have it to be able to run tests and make sure that translation you setup is working.

2. Documentation explaining complex flat file layout and structure. You should have some document that would list all the fields or elements that you need to create or read from the file. Contact your trading partner or supplier for simple documentation on all the fields.

Once mapping is done you do not have to recreate it again simply save it into the file with extension *.xmp. You can run map files using other utilities that come in the package (read User's Manual about other utility programs).

Software requirements

You will need to download and install XTranslator from the website http://www.xtranslator.com . Translator comes with number of templates accessible via Template Wizard. While you can create mapping templates and layouts manually adding them one item at a time, use of Template Wizard for standard formats like EDI X12, EDIFACT, etc. saves time.

Setup program asks if you want to install templates that come with the package. If you choose not to install them, Template Wizard will not work and you will not be able to follow some parts of this document.

Once it is done, start Map Editor tool.

This document comes with complete map in a file with extension xmp. Please open and examine it as you read this document.

What is complex flat file layout?

Complex flat text files usually have some common traits:

1. Each record is on separate line ending with carriage return and line feed.
2. Each record starts with some text identifier that tells us what kind of record it is. Record identifier defines each record as header, sub-header, detail, second detail, trailer, etc.
3. Each record contains fields that are either fixed length or variable length delimited with special character.

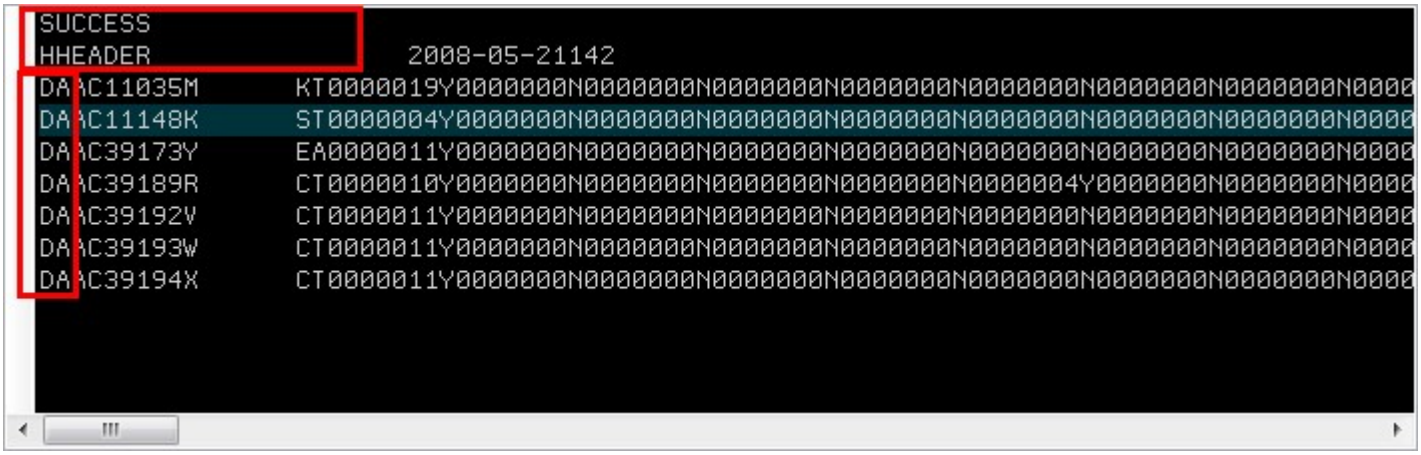
If you want to take in complex flat file and translate it translator map has to use all these traits to extract fields. If you are producing complex flat file translator needs to create the same structure with these traits.

There are many complex flat text files used in various computer systems and environments. Examples are numerous: NSF 2, NSF 3, flat file SAP IDoc, UB92 and countless more. They all have these basic three traits listed above and mapping steps for all of them are essentially the same explained in this document.

Since translator map slightly differs if you are reading or producing complex flat file we cover each scenario in separate chapter below. However initial steps are the same both for input and output layouts. You need to add message, add segments representing lines of the complex flat file and then add elements representing flat file fields. The only difference for the output is that sometimes you might want to hardcode certain constant values on the output side. So certain fields would contain constant values instead of being mapped. Other than that layout definitions in both cases are essentially the same.

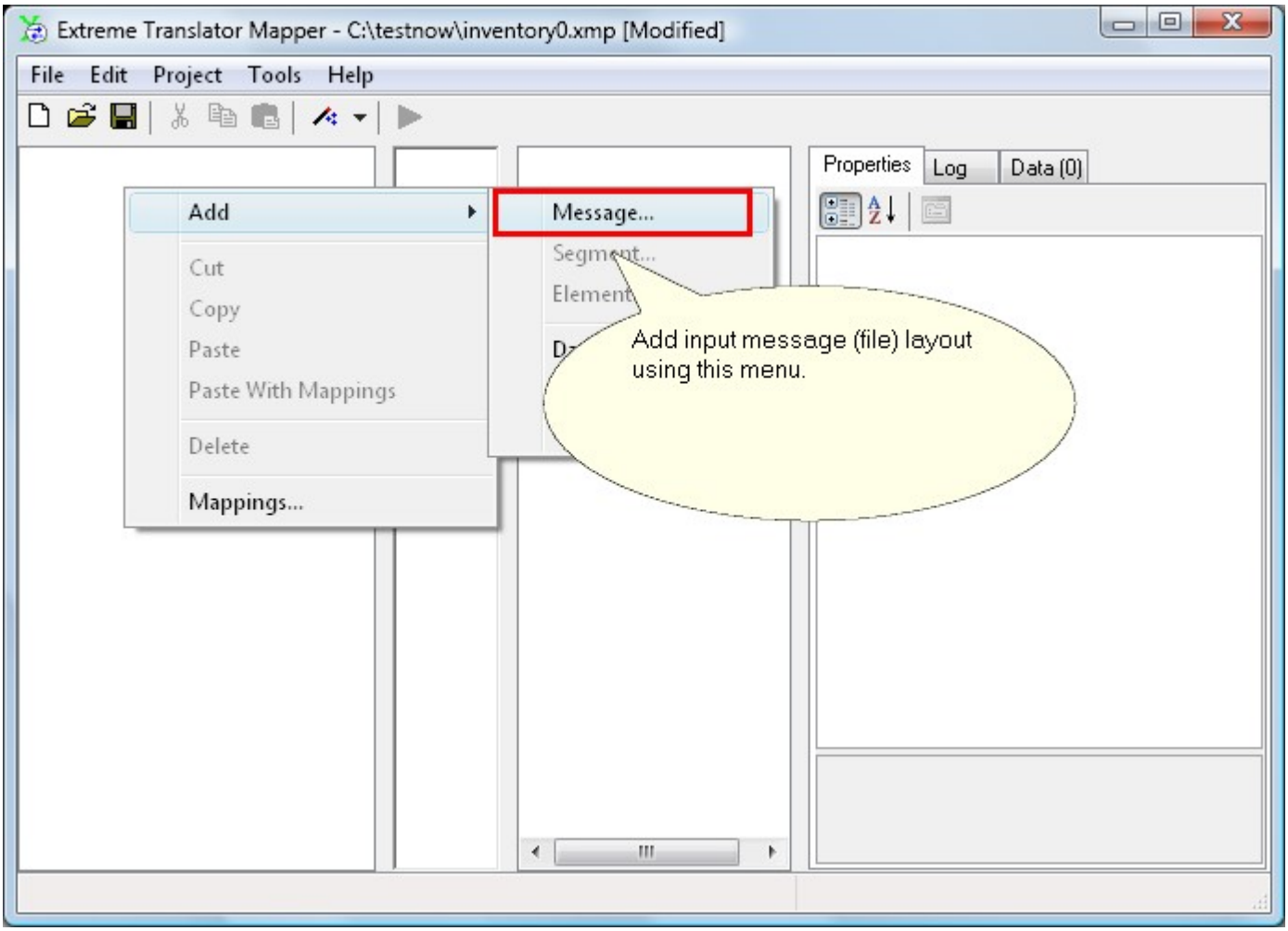
Complex flat text file as input

We are going to map inventory file with few header lines and detail lines containing fixed length fields to CSV output file. Certain fields have to be mapped to comma separated (CSV) output file other fields are not transferred to the output but they still have to be setup in the map in order to read the whole file structure.

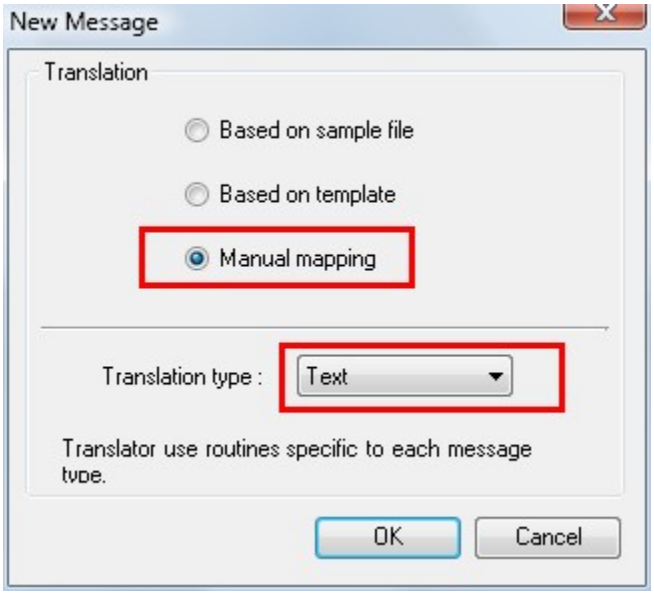


This inventory file contains two separate header lines that start with text SUCCESS or HHEADER. It also contains detail lines that start with text DA. Each line ends with carriage return and line feed. Detail line fields have fixed length.
Our setup will have two segments representing header lines, one segment representing detail lines and detail segment will contain fields with fixed length to read all the values from the flat file.

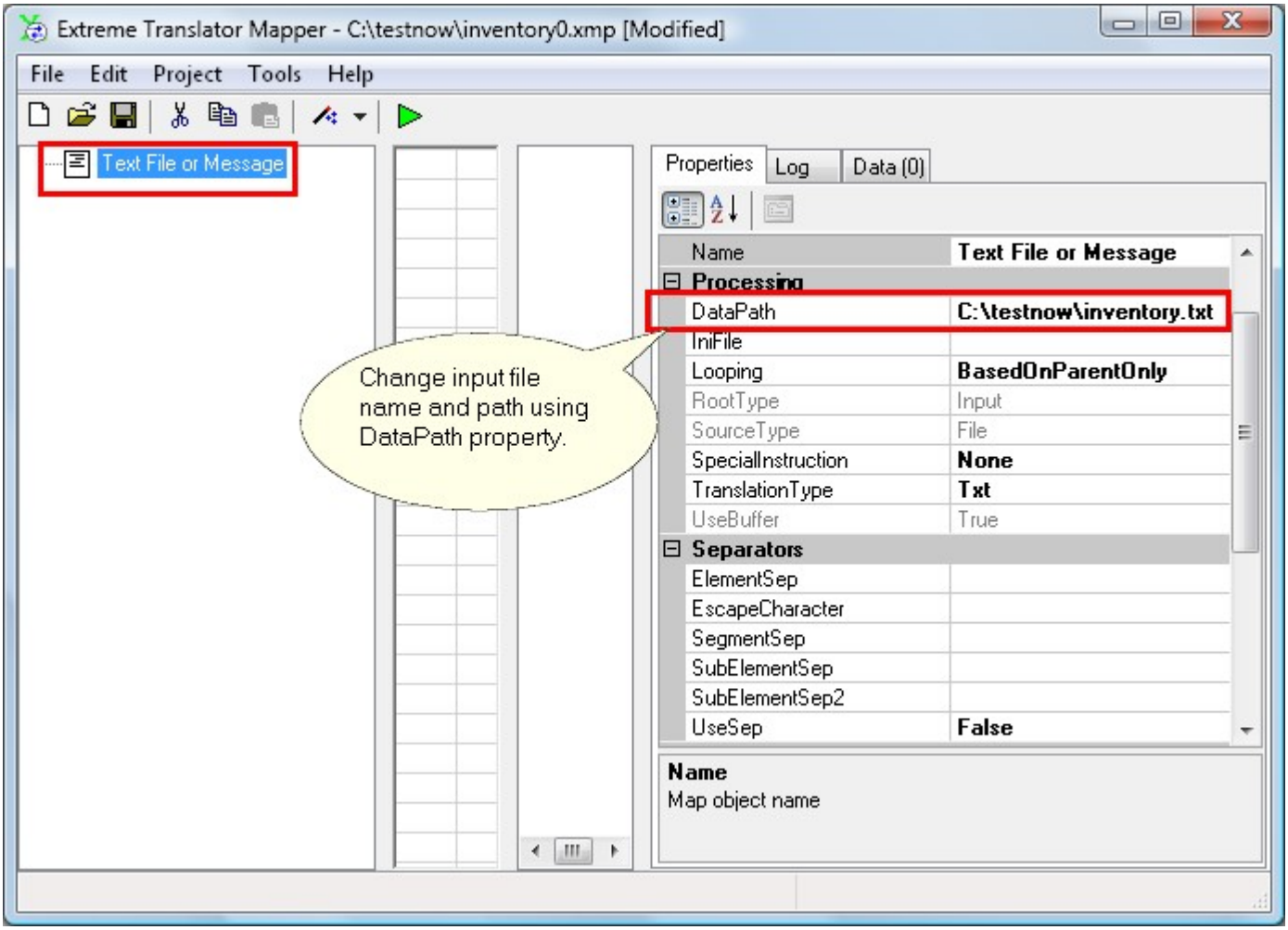
Map Editor main screen contains 4 panes going left to right. First left most pane is for input layout, second displays mappings, third is for output layout and last one shows Properties/Log/Data for the item selected in first or third pane.



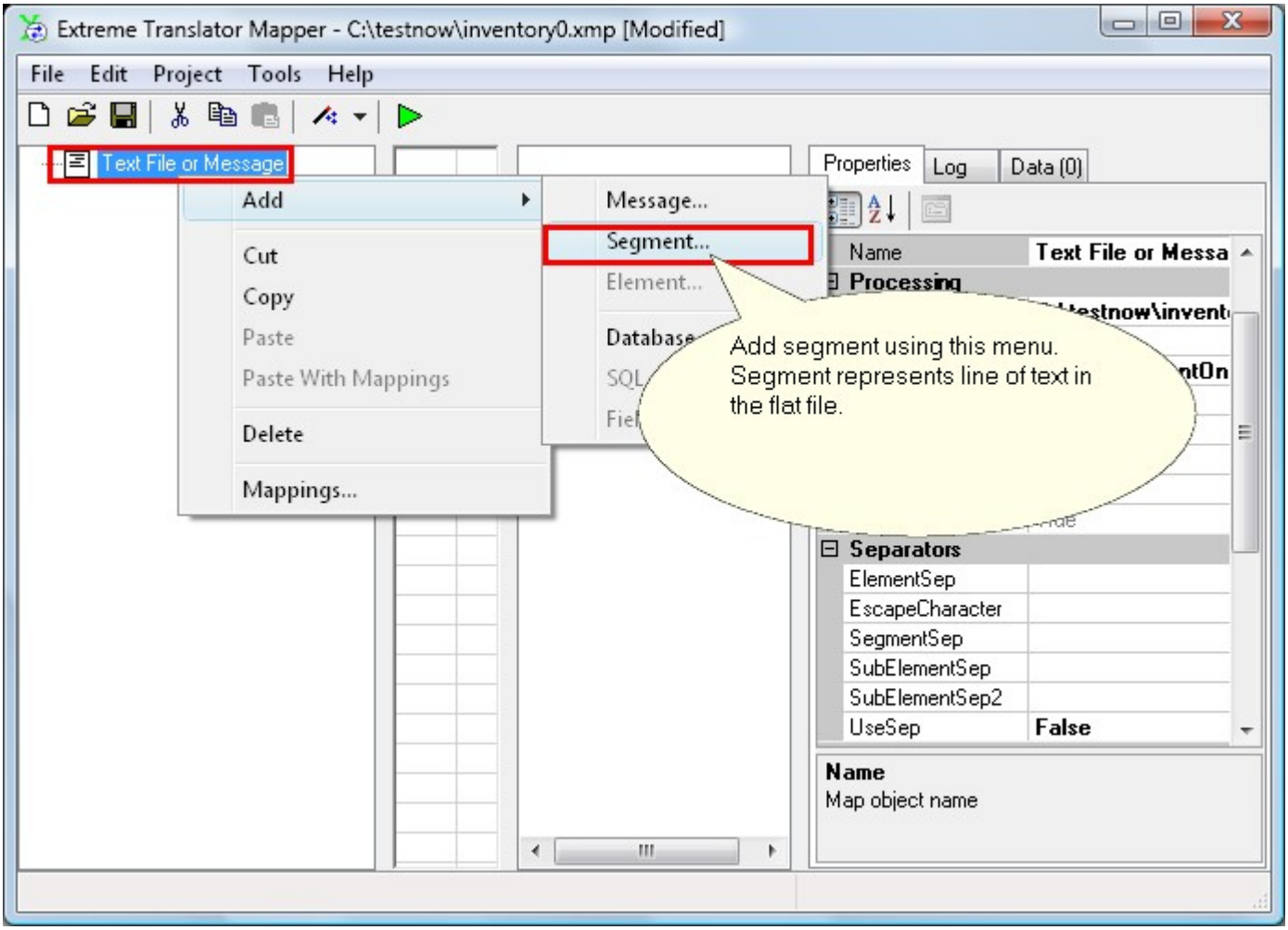
We will setup flat file layout manually and add all the items to it one by one. Mapping starts with the new message. Message is your file.



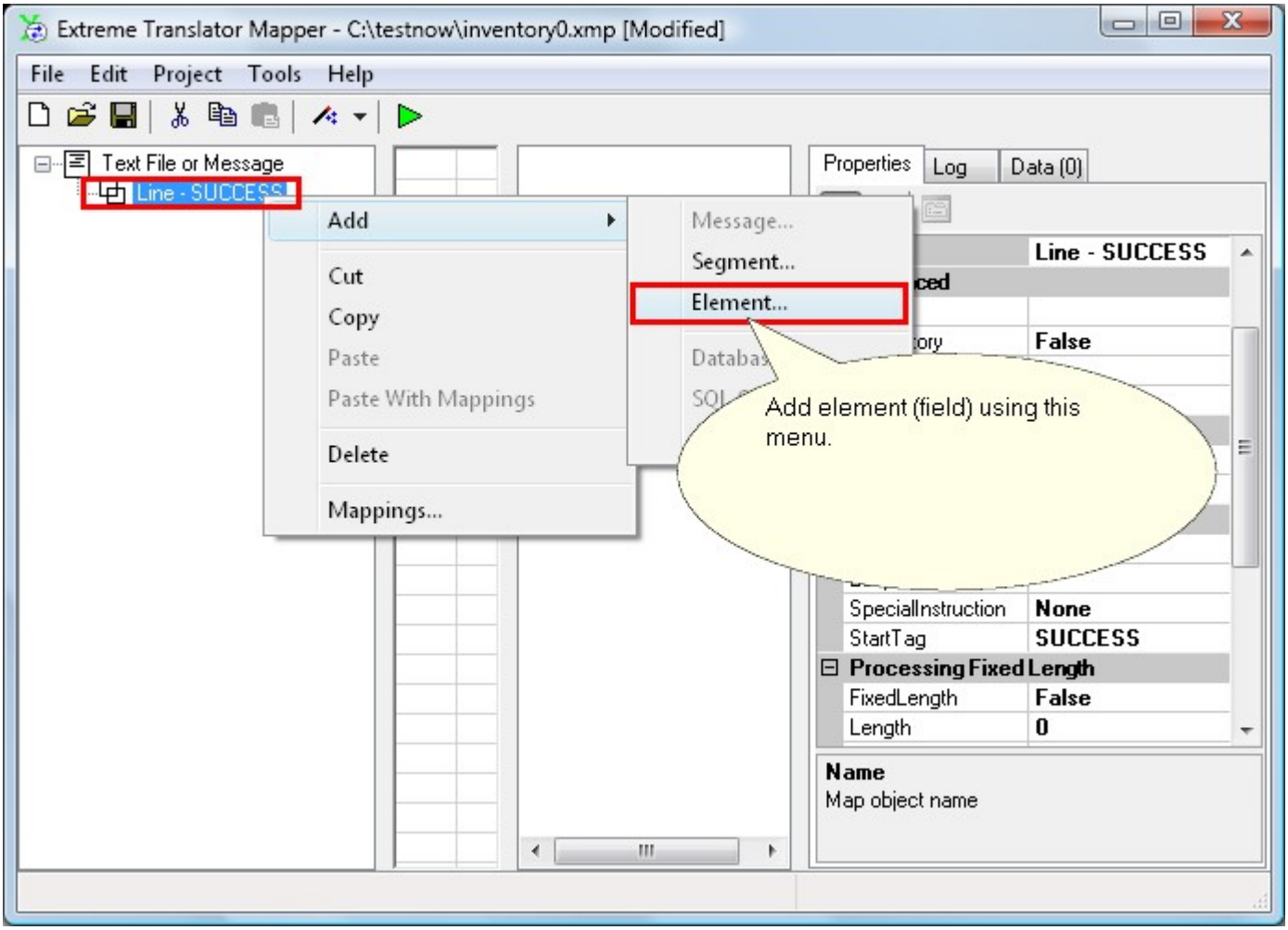
If file has custom non-standard format you can not import its template using Template Wizard or use By-Example wizard to read its structure. For non-standard formats manual mapping is the only choice.



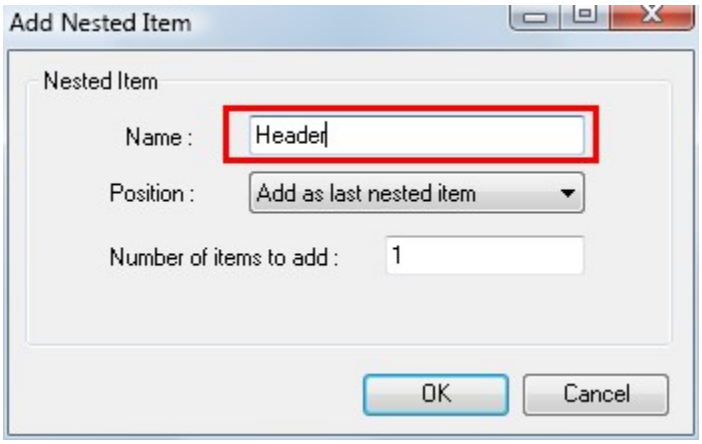
You can set file name and path using DataPath property. Depending on how you are going to use the map, this value can be changed via Developer SDK, command line or GUI based map runner applications.



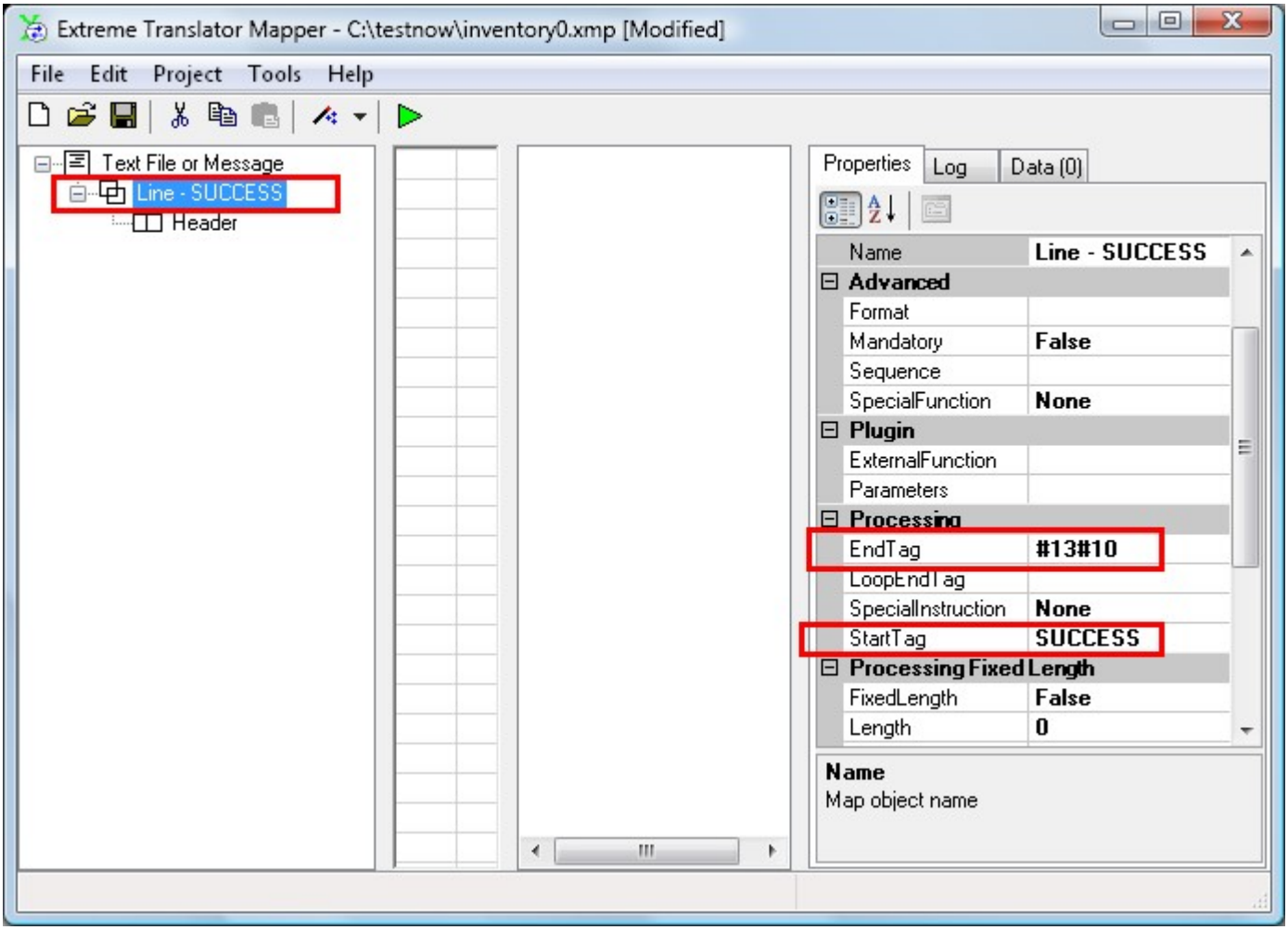
Add as many segments as many unique lines you might have in the input file. Unique lines are header, sub-header, detail, and trailer. In reality your file might contain one header, one sub-header, ten detail lines and one trailer line but you only need four segments to represent all these lines. So even detail line is in the file ten times, you only need one segment to represent it in the map.



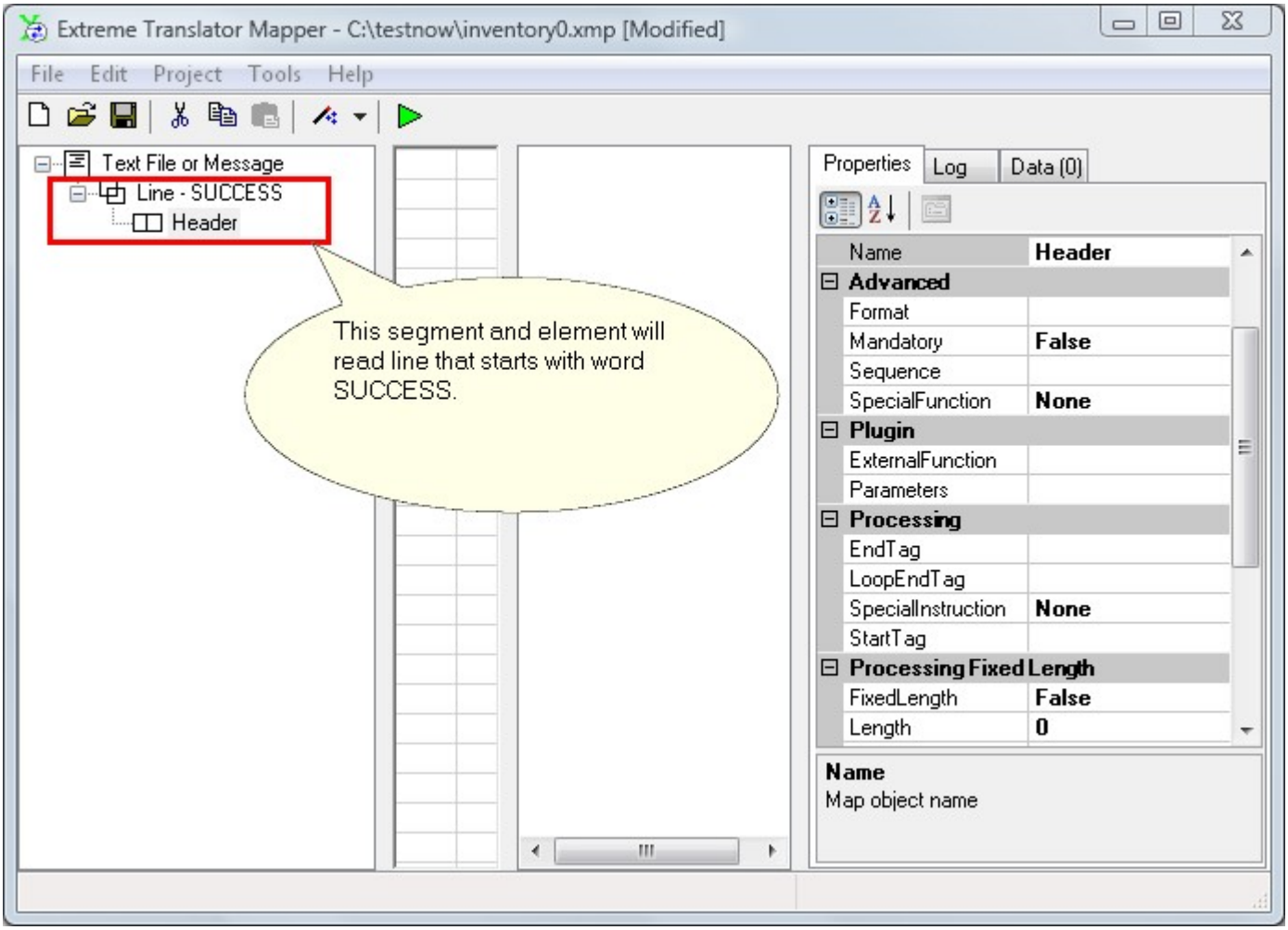
Each line of complex flat file may contain number of fields. Add all those fields one by one and set specific length for each of them using FixedLength=True and Length properties. If your fields do not have specific length but are separated by some separator, example comma then set EndTag property to comma for each field.



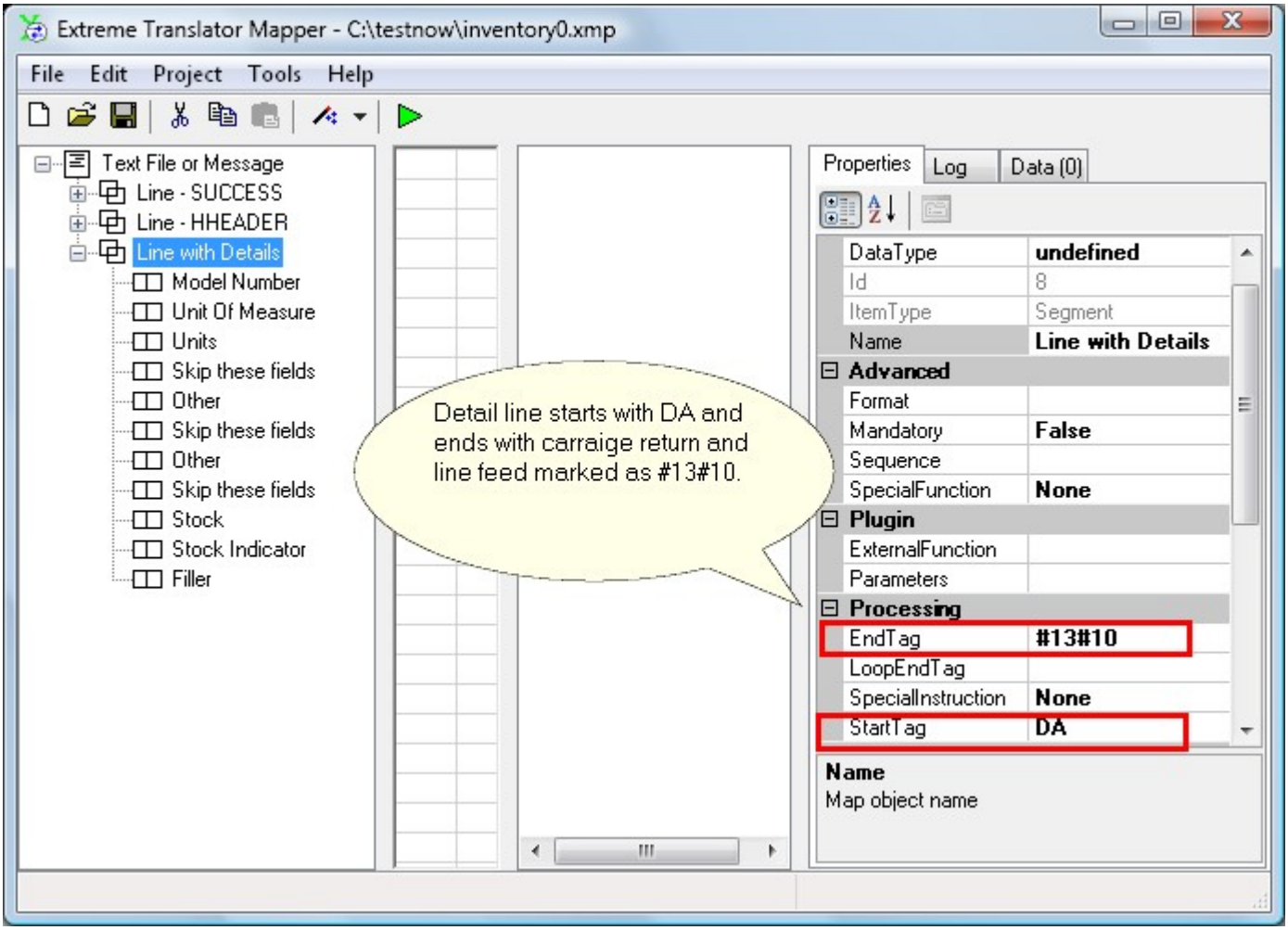
You can change field name using this screen or change it later via property called Name.



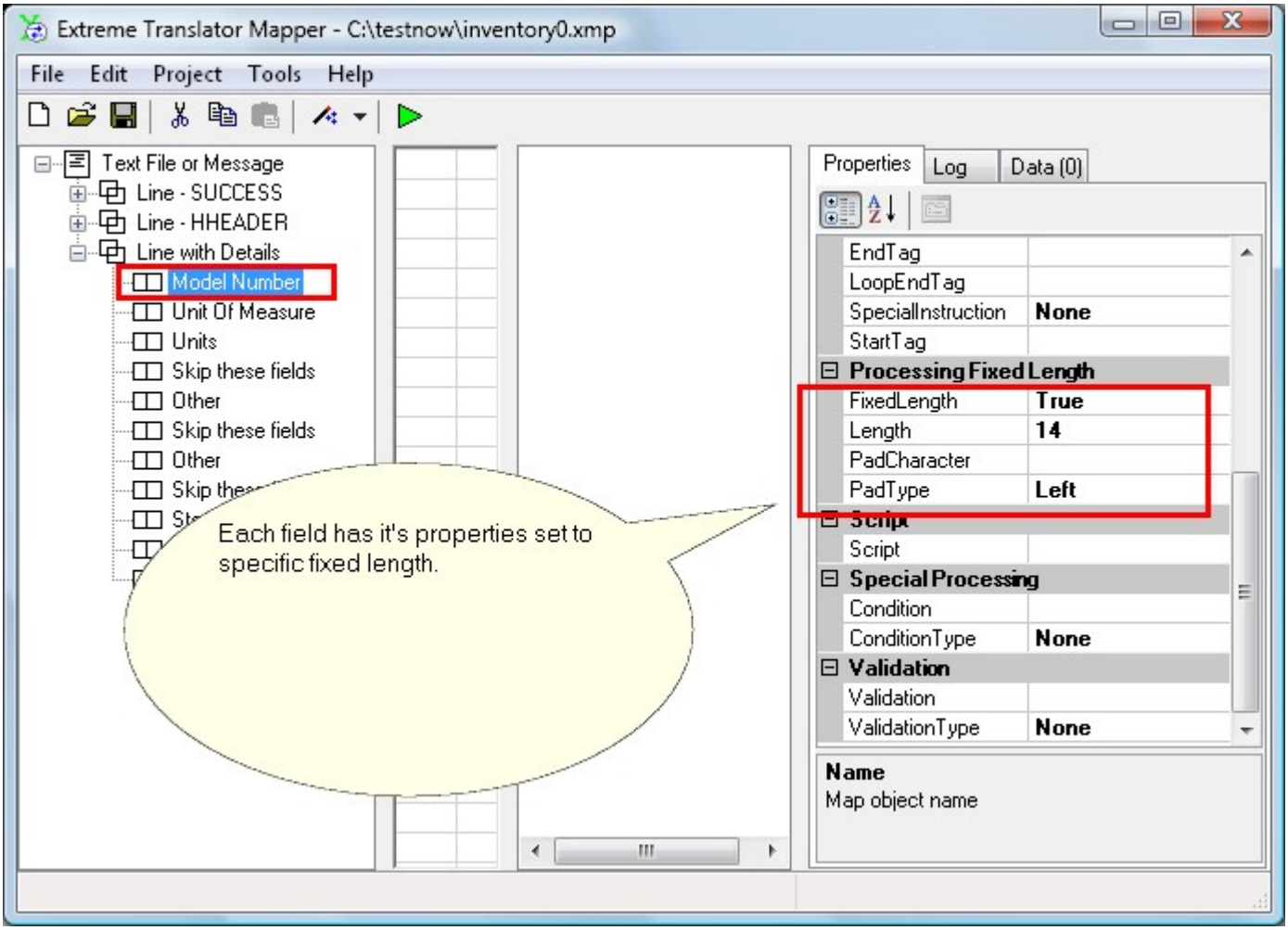
We expect each header line to start with word “SUCCESS” and end with carriage return and line feed. Those two features are expressed via properties StartTag and EndTag. #13#10 are decimal codes for carriage return and line feed.



We also add field to the segment so it would read all the additional data that might come after word “SUCCESS” and before carriage return and line feed.

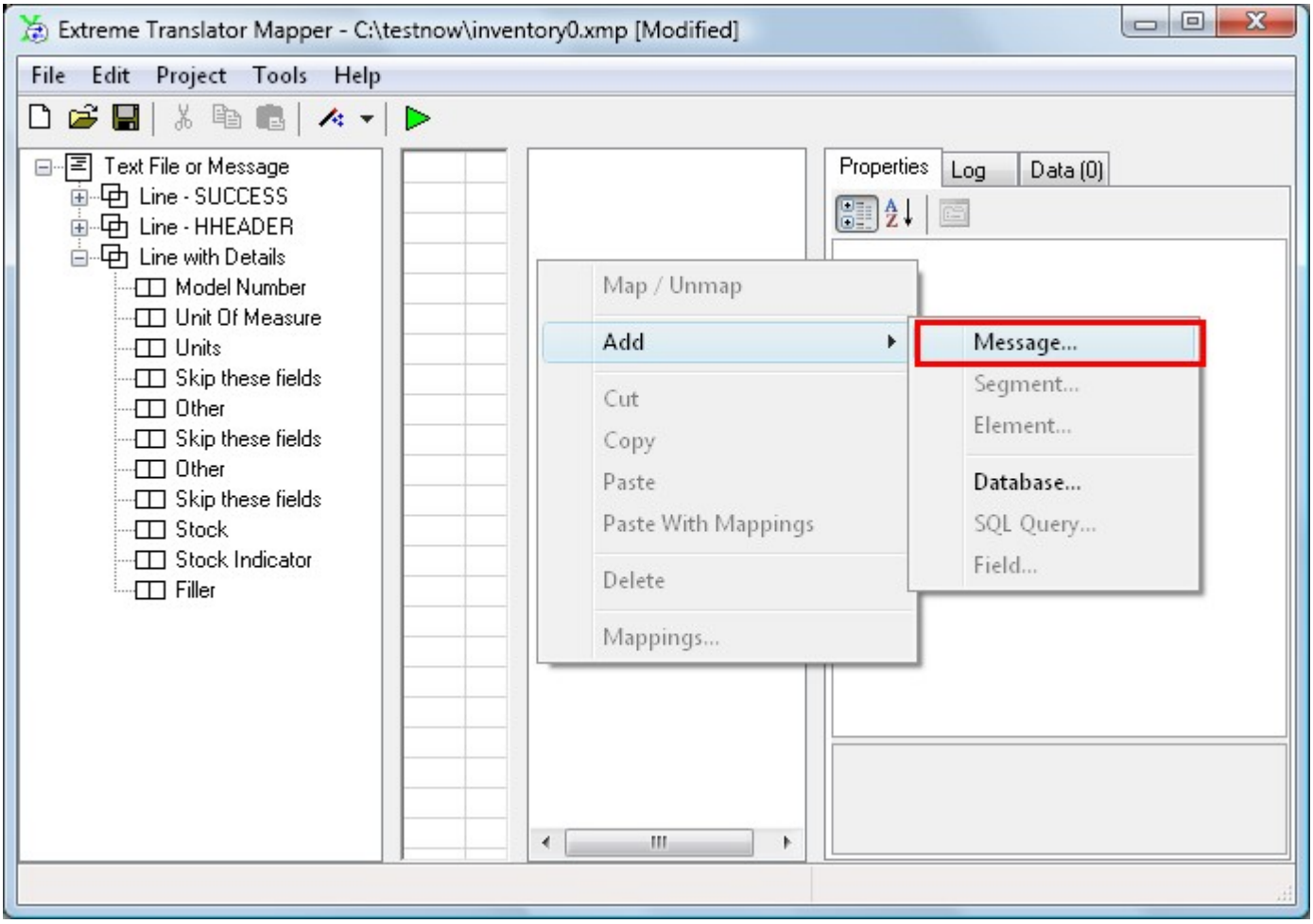


We expect each detail line to start with text “DA” and end with carriage return and line feed.

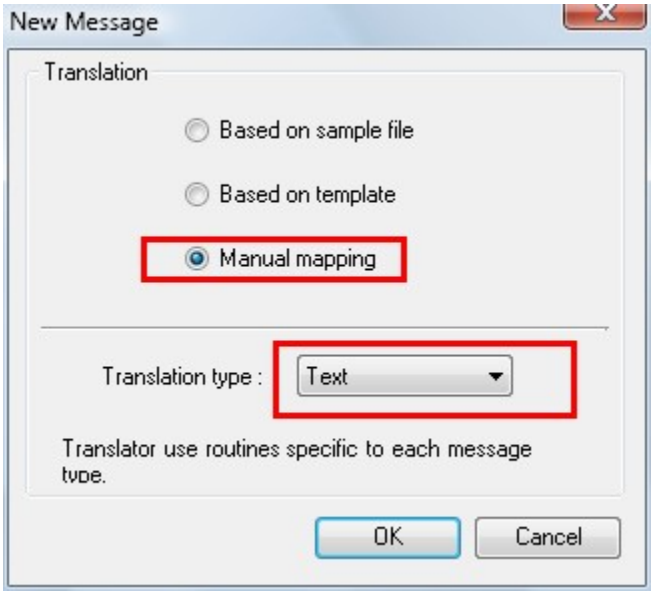


All detail line fields have specific length. We set each to FixedLength=True and Length to specific value.

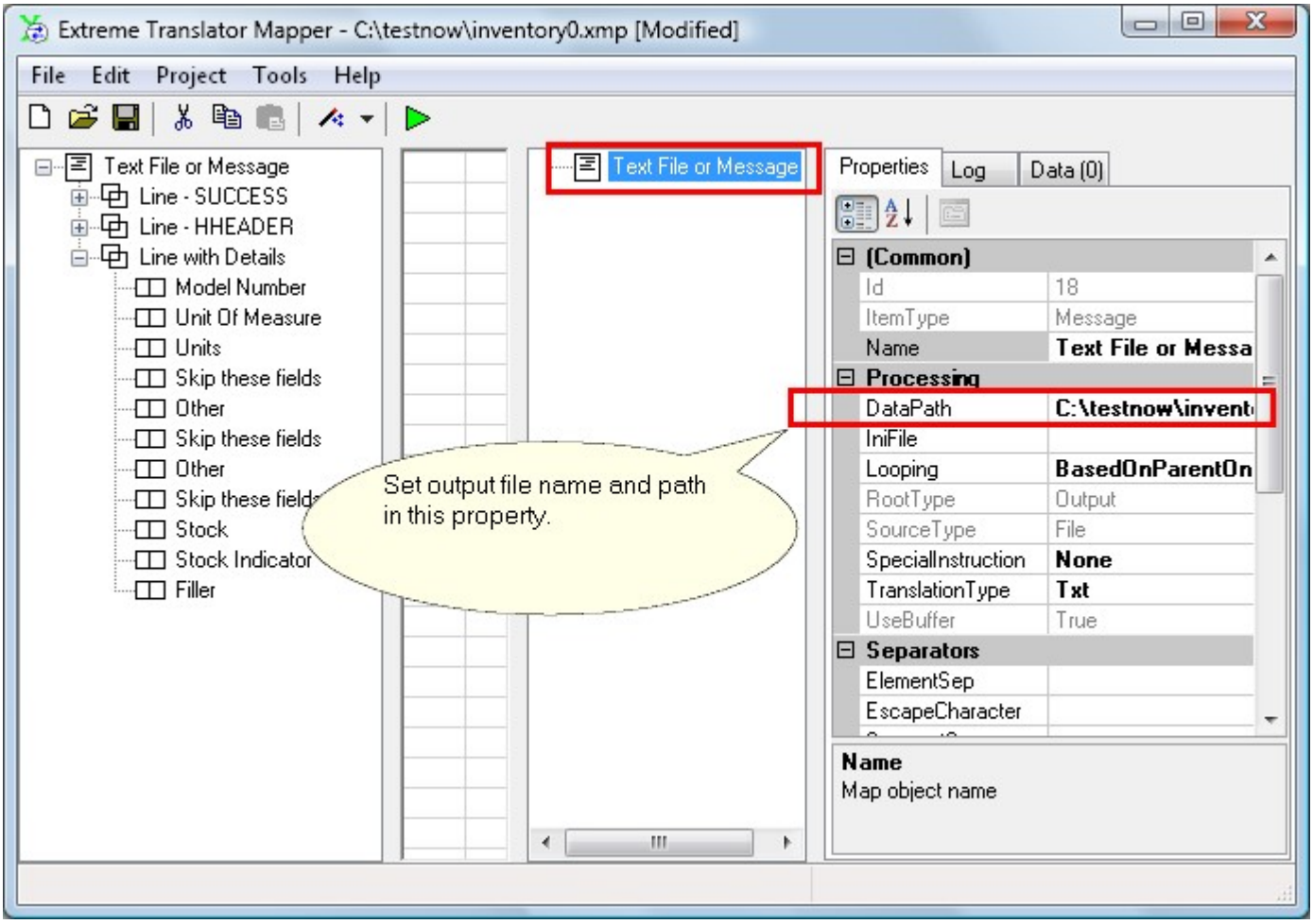
Input side is set. Now we will set layout for comma separated value file (CSV) output.



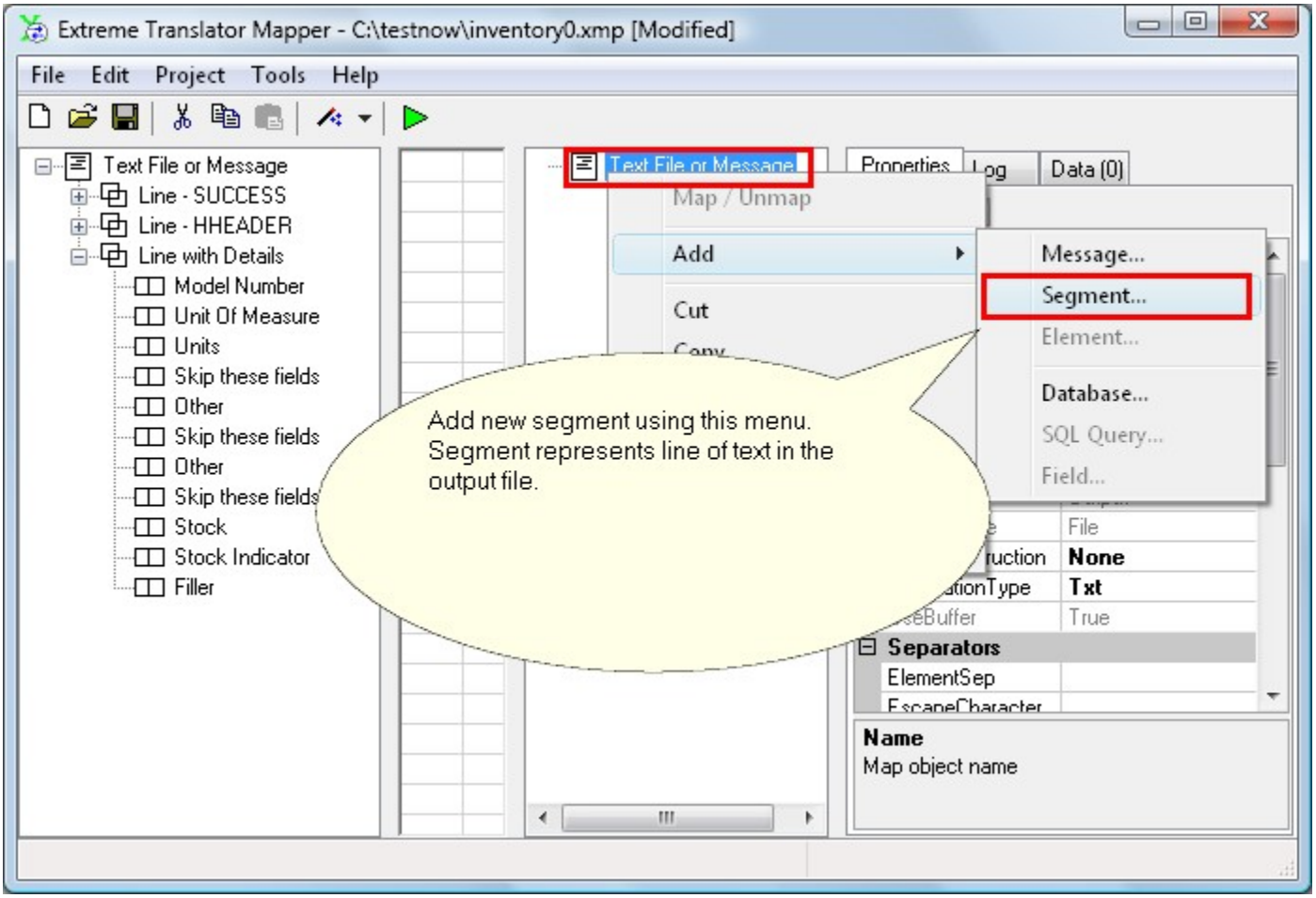
Add output message by right clicking on the third pane and selecting Add->Message menu.



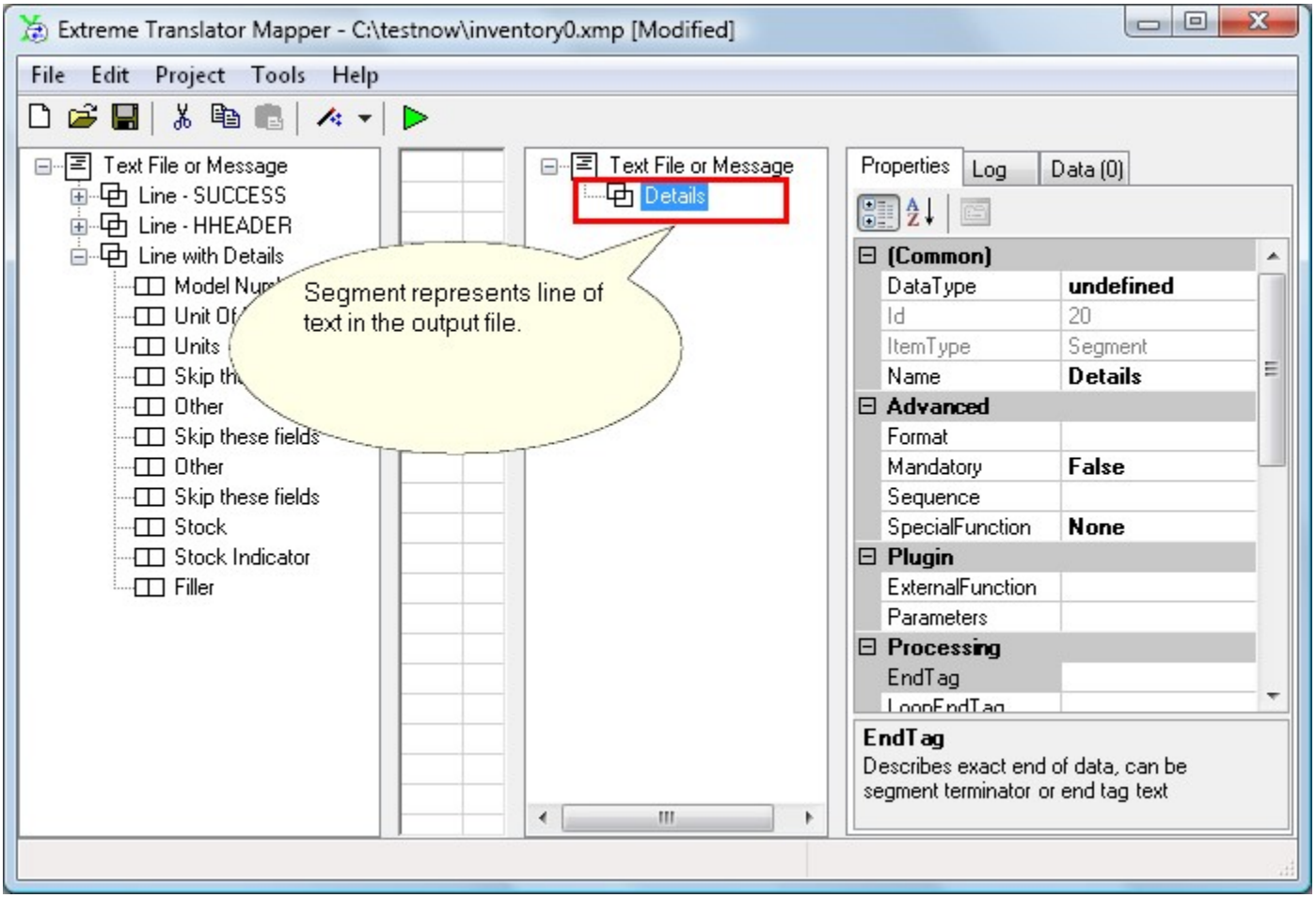
This layout is easier to setup using By-Example Wizard if we would already have sample CSV file we want to output. Since we do not have sample CSV file we choose Manual mapping.



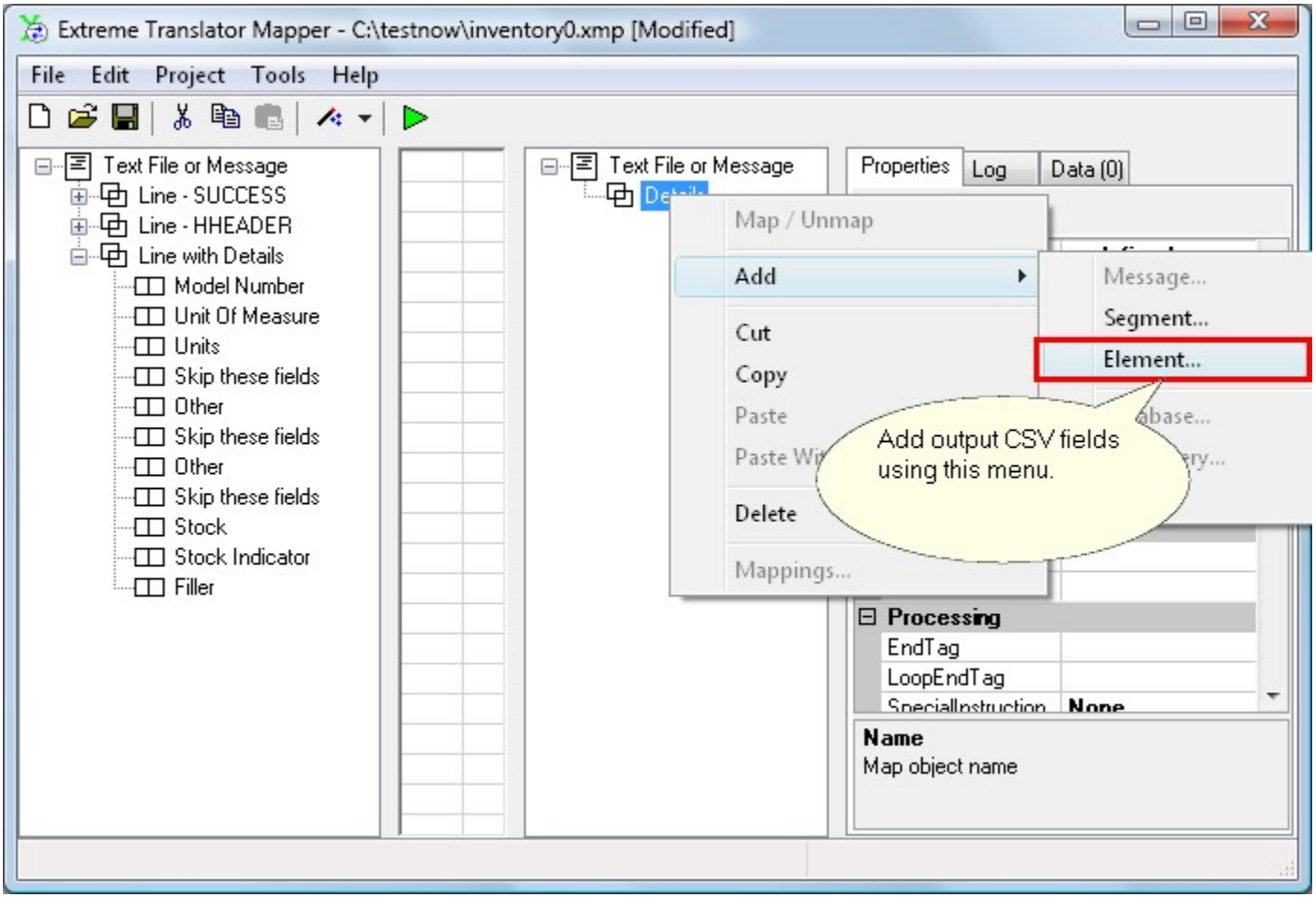
We setup output file name and path using DataPath property.



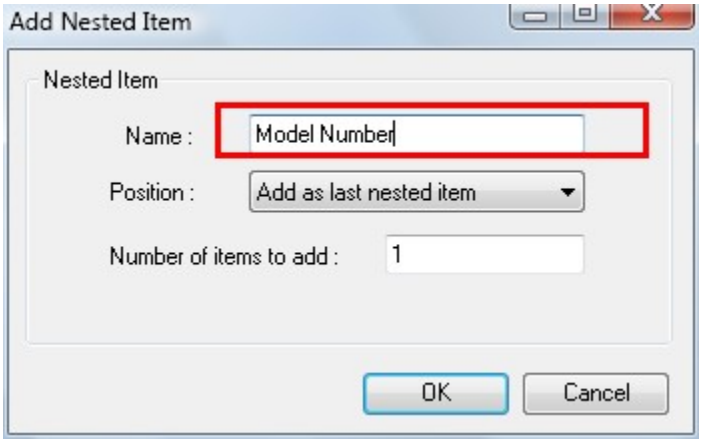
CSV flat files have same basic idea as complex flat files – segment represents line of text. Typical CSV files have one line that repeats multiple times. So all we need in translator is single segment that will represent all these lines of CSV file.



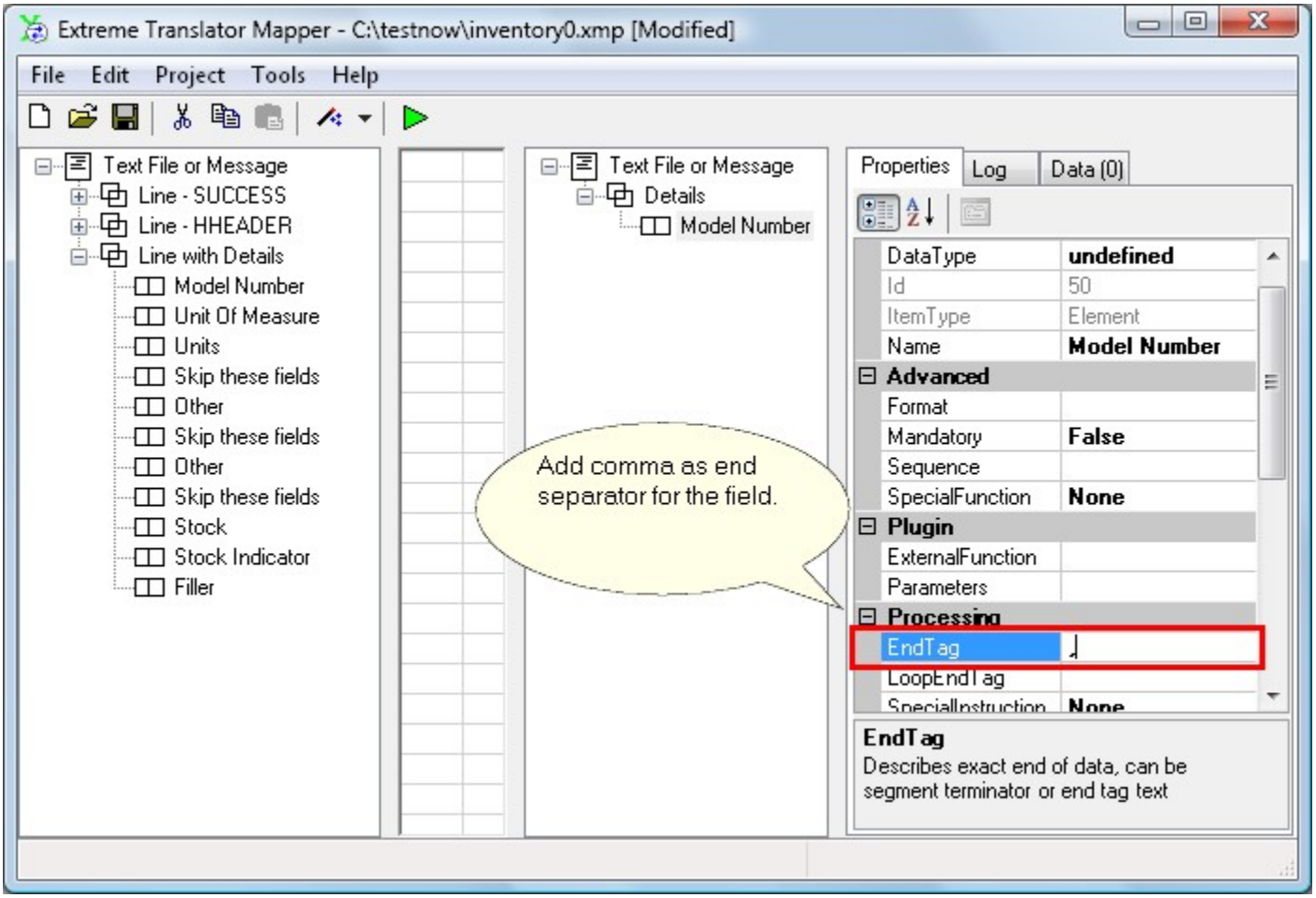
There is newly added segment.



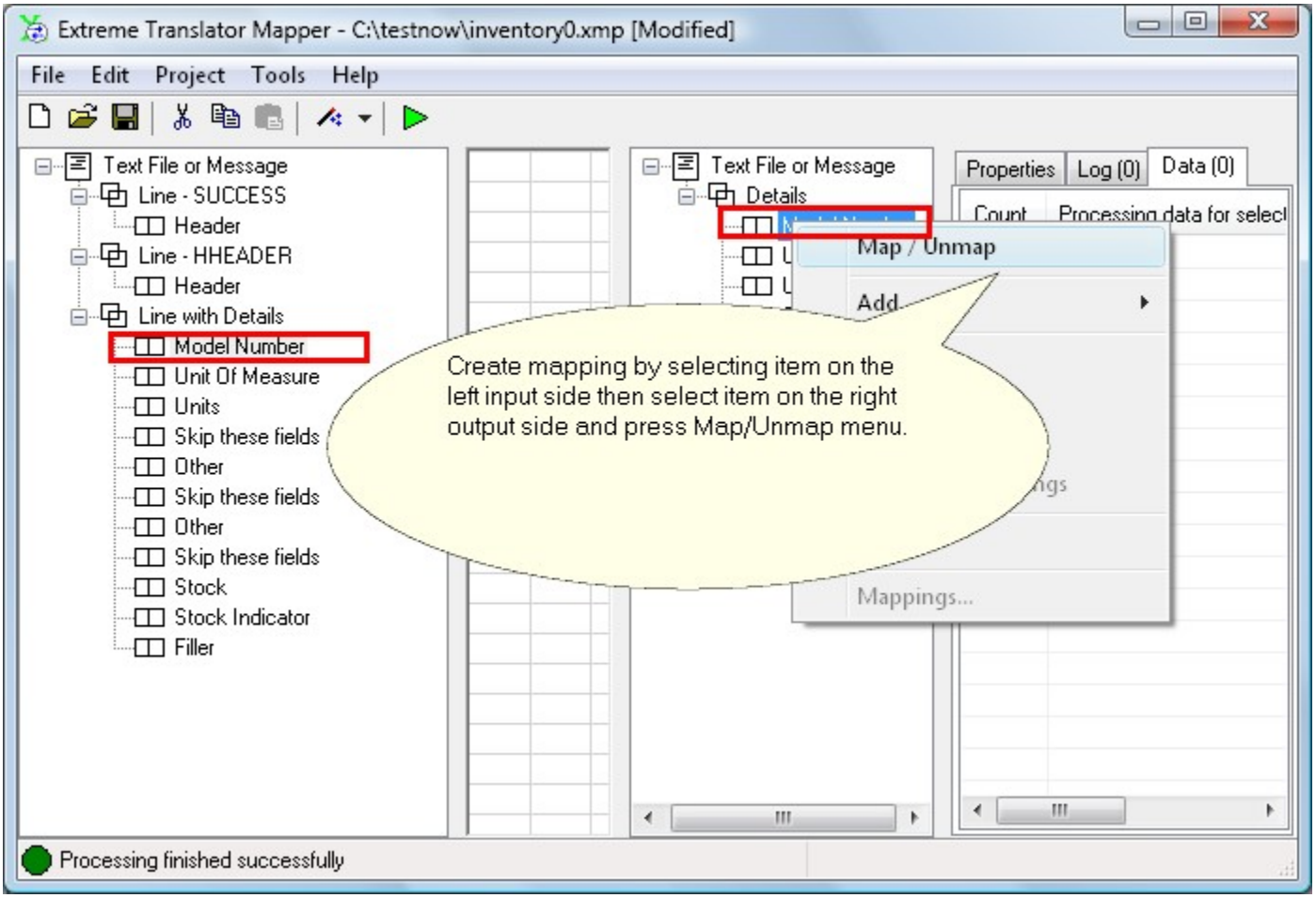
We add elements (fields) to the output side segment (line).



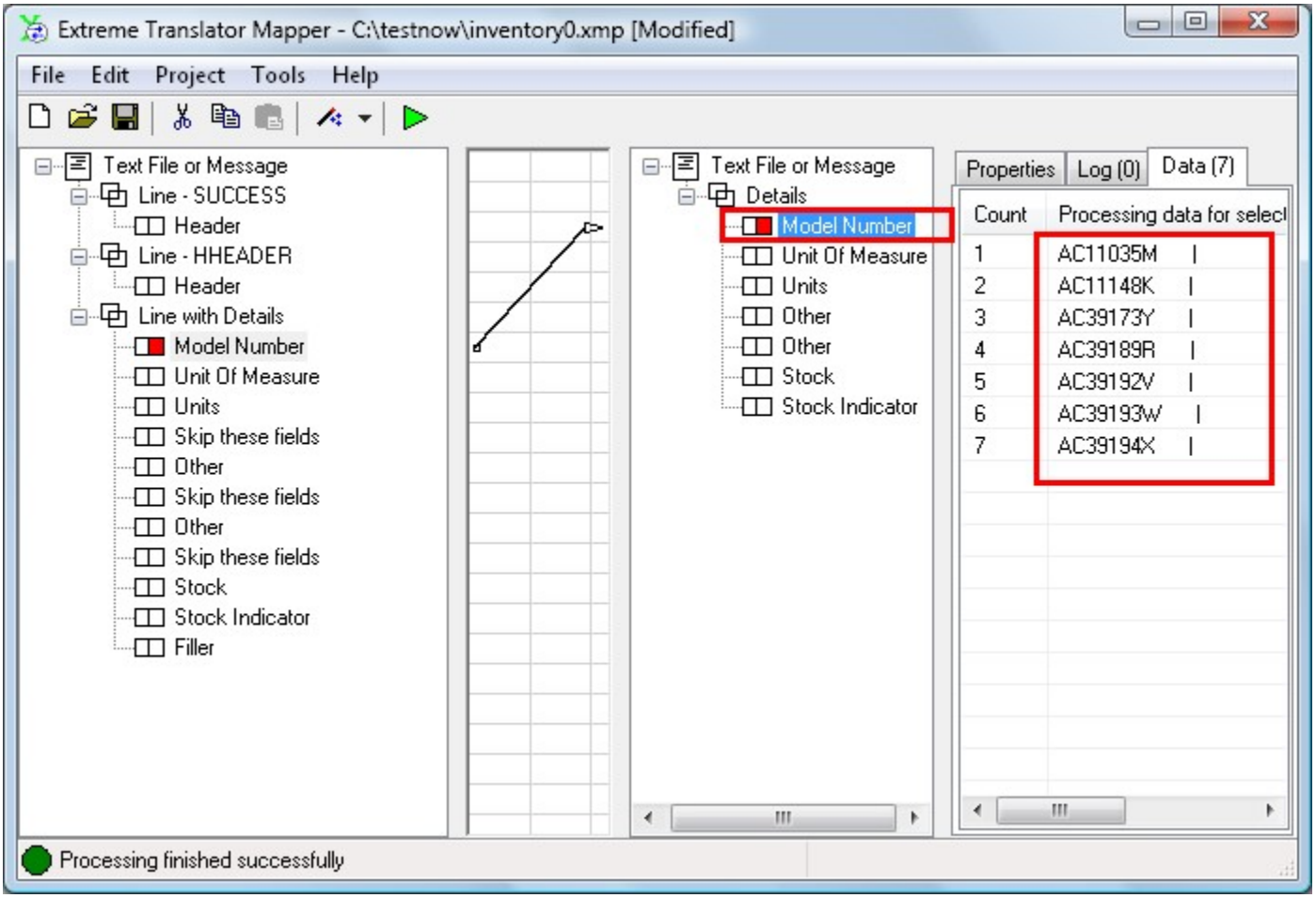
We change field names. It is also possible to change Position where new field is added. This helps to add fields in the middle of already setup fields.



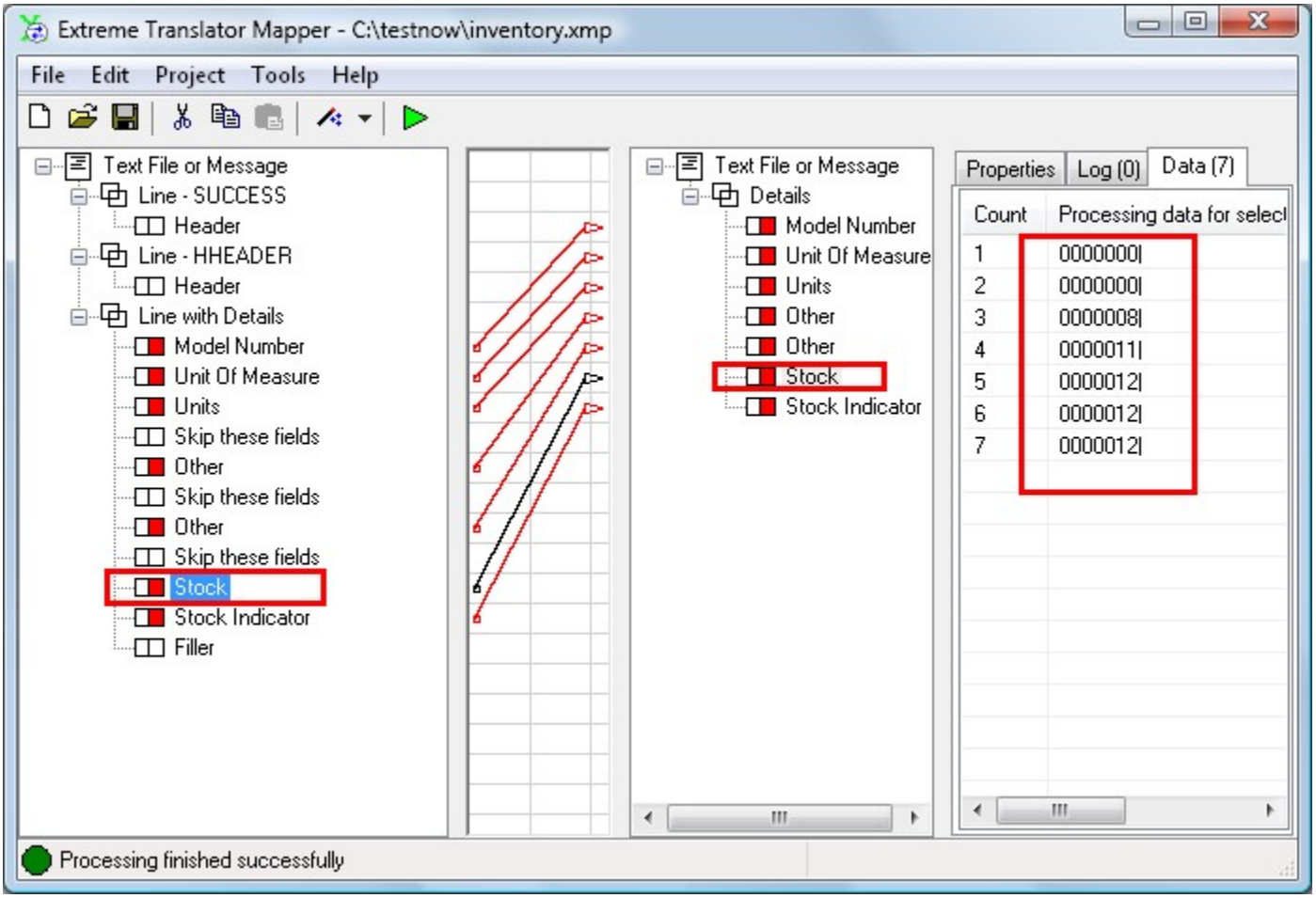
Each comma separated field has to end with comma separator. This is easy to setup using EndTag property. Just enter comma as indicated in the screenshot above.



We create mappings from input to output. This is simple: click on the left side (input) click on the right side field (output) then right click on the right side field and click menu Map / Unmap.



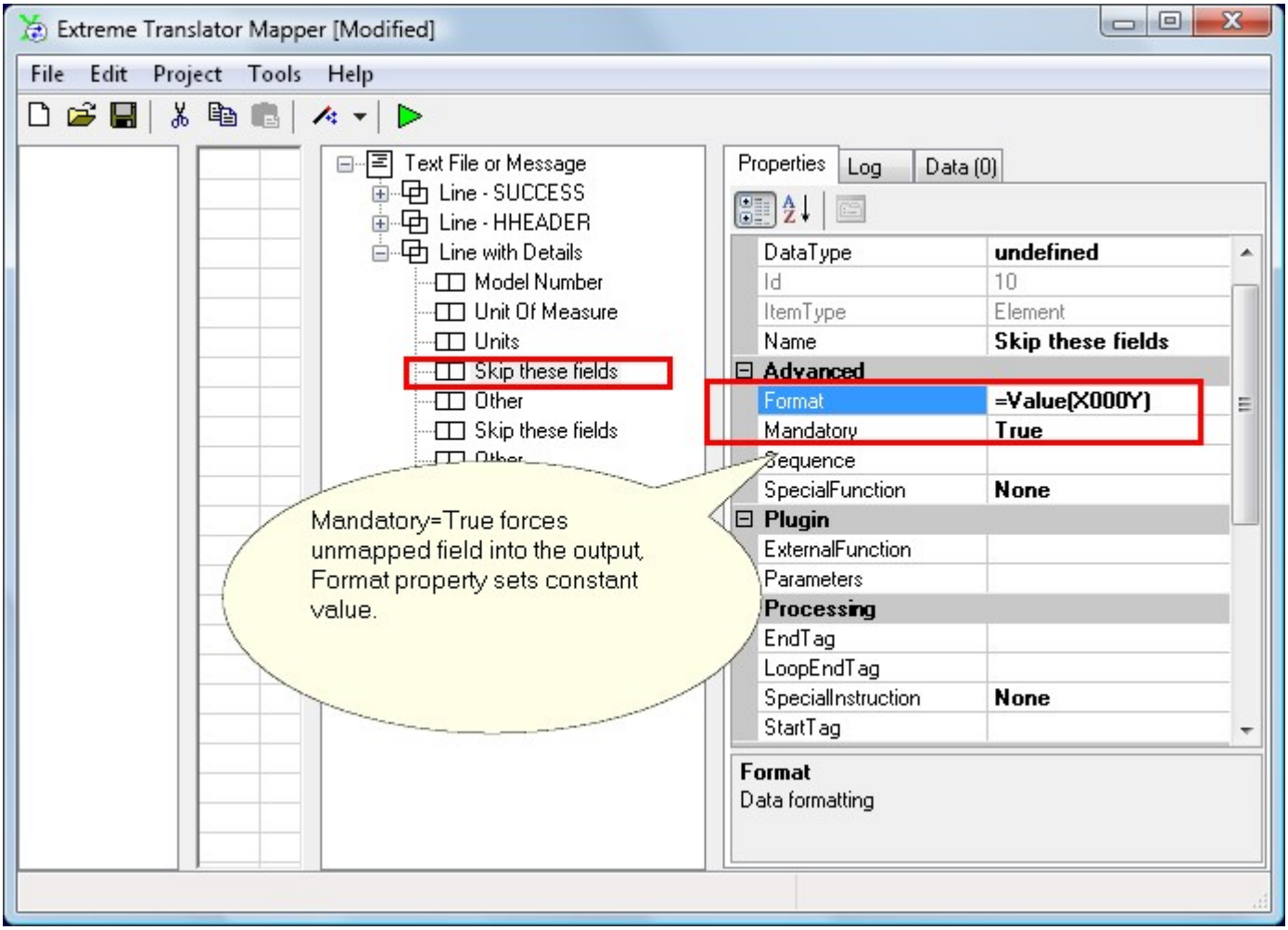
After at least one mapping is created run the map using menu Project->Run or simply press green button in the toolbar. Once map runs, you can click on any item on the screen to see data that translator placed for that item via Data tab on the fourth pane.



After you map more and more fields continue running translation during mapping process and check results in Data tab. This helps to ensure that your mapping work goes in the right direction. You should see more and more data in the output as your work progress.

Complex flat text file as output

Steps to setup layout for complex flat text file in the output are essentially the same as for the input. Please review previous chapter to see how to define complex flat file layout in general.
When you have complex flat file as output sometimes you might want to hardcode certain constant values in the output fields so they would always be present without even being mapped. This is done using Mandatory and Format properties on the element (field).



Simply set Mandatory property on the field to True and set Format property to =Value(_your_constant_here_) . Mandatory set to “true” forces field to be produced into the output and Format function =Value() produces certain constant. Mandatory set to “true” forces field into the output only if all the fields before it in the same segment are mapped or set to Mandatory=True as well.

Format property has many other functions that can help you reformat the data, replace certain characters, substitute one incoming value with the other. Check Translator User’s Manual for complete list of available functions.

Review

This chapter provides more in depth coverage of how translator creates or reads complex flat files. It also reviews some of the properties mentioned in previous chapters. Feel free to skip this chapter if you already understand how to define your flat file layout.

Basic idea behind mapping is identifying traits that your complex file has and setting up translator map to parse incoming data based on those traits or producing data including those traits (if complex file is your output). If your file contains data records in lines of text and each line starts with unique identifier all you need to do is setup segments that would represent each unique line. Then add elements to each segment to extract fields that you need.

Translator uses sequential left to right parser to read text data. If complex text file is produced then translator traverses output map layout (called output tree) and produces data sequentially from left to right.
When we say “sequential” parsing we mean that translator keeps a bookmark of current position in the file, and moves through the file one item at a time. So if current item has StartTag set to some text value, translator will look for that text value not from the file start but from the position where previous last item was located. That’s why map layout should follow file’s physical structure.

Each item in complex flat file layout (except very first root item called Message) has same properties called FixedLength, Length, StartTag and EndTag. Those 4 properties are most important when you define complex file layout. They tell translator where to find specific data in the input file, or how to place data in the output (in case you are producing complex flat file).

If property FixedLength=True then StartTag and EndTag properties have no effect and only Length property is used. Then Length defines how many characters item has and how long it is.
Otherwise if FixedLength=False then StartTag and EndTag properties are used for parsing. Then StartTag defines what translator should search in the file. Once translator finds data based on StartTag it looks for EndTag to find where the item ends.

Following table covers some processing situations based on what properties are set for the item:

Fixed vs. Variable length flag setting	**Parsing behavior**
FixedLength = True Length = item's expected length	Length is used to read item. StartTag and EndTag has no effect.
FixedLength = False StartTag = text that identifies item's start EndTag = text that identifies item's end	Translator will search for StartTag, then EndTag to read the item. Length property is not used.
FixedLength = False StartTag = (blank) EndTag = text that identifies item's end	Translator will search for EndTag to read item. Since StartTag is empty start will be where previous item ended. Length property is not used.
FixedLength = False StartTag = text that identifies item's start EndTag = (blank)	Translator will search for StartTag, then it will read item to the end of the file. Length property is not used.

When setting up input layout you can run map every time you make a change to it. This will ensure that you are on the right track. Check parsed data using "Data" tab on each input item to make sure it is being parsed correctly. Complex flat file layouts take a lot of time to setup since each item has to be setup exactly and formats are usually non-standard so it is not possible to import templates from Template Wizard to shorten workload.

Other issue is data looping. Some complex flat files do not follow same looping sequence as expected output file. In some cases complex flat files are just data dump from some older legacy computer system. Input data could be so convoluted that only why to map it to the output would be by using some intermediary storage like relational database and by having two translation maps: one - reading data and loading into the relational database, other – writing data out of relational database into looping format.

We hope this document helped you. If you have any questions please contact us via Support page.