Etasoft
Building blocks of integration

XTranslator v2.0x User's Manual

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

Purpose

XTranslator is lightweight any-to-any data translator.

Package contains:

1. Map Editor for translation map development.
2. Various tools to run maps directly.
3. Developer SDK for C#.NET (DeveloperSDK folder).
4. Sample plug-in functions (sampleplugins folder).
5. Additional documentation.

Companion Packages

Companion XT Server package adds additional features to the translator. Use XT Server for job scheduling, backups, incoming file monitoring, FTP file transfers and EDI validation.

Translator is stand-alone package and can be used without companion package but additional features will be missing.

Companion package has separate license key.

Recommended Directory Structure

Each map should have separate folder dedicated to it. All related mapping documentation should be in the same folder as map file. Input and output subfolders should be under the folder with the map file.

Create one map per each specific translation.

Starting with well defined but simple directory structure helps avoid number of future problems. Maps, execution scripts and configuration files built later are based on that initial structure.

Requirements

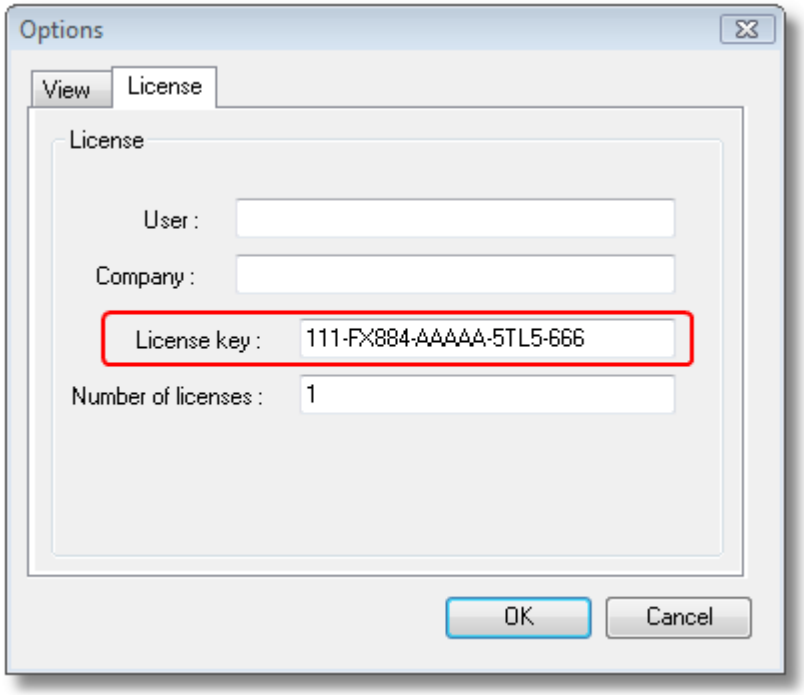
This is minimal configuration:

CPU	1500 MHz
RAM	1 Gbt
HDD	3.5 Gbt
Other	Keyboard, mouse
Operating System	Windows Vista/7/8/10, Server 2003, 2008, 2012, 2016

License

Licensing model is based on number of per machine installations. Install retail version with setup license key on as many machines (computers) as many licenses have been purchased. It is possible to buy unlimited install and distribution licenses. Please check licensing details on our website.

Purchase licenses on our website. After the purchase you will receive email with new license key. You should go to submenu “Options” under menu “Project” and enter that key in “License Key” field.



Enter purchased license key.

If you do not enter new license key or trial has expired, translation will not run and license key has to be purchased.

Once translator is installed on a new computer it defaults to trial license as long as purchased license key has not been entered into the Options screen.

Important: When moving translator software to new PC/server make sure to enter purchased license key. If you do not enter purchased license key into newly installed software trial period will end in 3 weeks and processing will stop until valid key is entered.

Map Editor

Map editor is divided into four panes:

1. First and third pane is for adding new map items.
2. Second pane is used to display relations between map items.
3. Fourth pane is used to display selected item properties.

Translator maps are sets of data translation rules defined using graphical mapping tool and saved in *.xmp files. Use map files to process data and transform it into some other format or relational database.

Mapping can take a long time if you have never used other mapping tools before. If it is a new experience it may take more than few hours to make the mapping work. Complex maps take on average 3-4 days.

Mapping process goes from left to right on the screen divided into input and output. You have to define at least one root item on both sides and one-or-many contained items underneath. Each root item represents a file or database connection. So it is the source or destination of data. You should define *DataPath* property in order for map to be functional. *DataPath* can be path to actual data file for input or output or it can be a constant value. In case if it is a constant value, you will be able to pass real path to the data via command line parameters. Please read more about it in a topic that describes how to pass data from command line.

However you can use wizards to do the mapping for you. In many cases they produce one side of the map, it can be input or output, and if you run wizard twice you can have most of the map created in minutes. All you will have to do is finish mapping using "Map/Unmap" menu in popup menu on the tree view in the right side.

Top level root item is always “Message” or “Database Connection”. When performing manual mapping add segments or tables to the top level item. Use “Add” menu or “Copy” and “Paste” menus.

Left pane is for input data source. It is used to read input data. Right pane is for output file or database connection. It is translation destination.

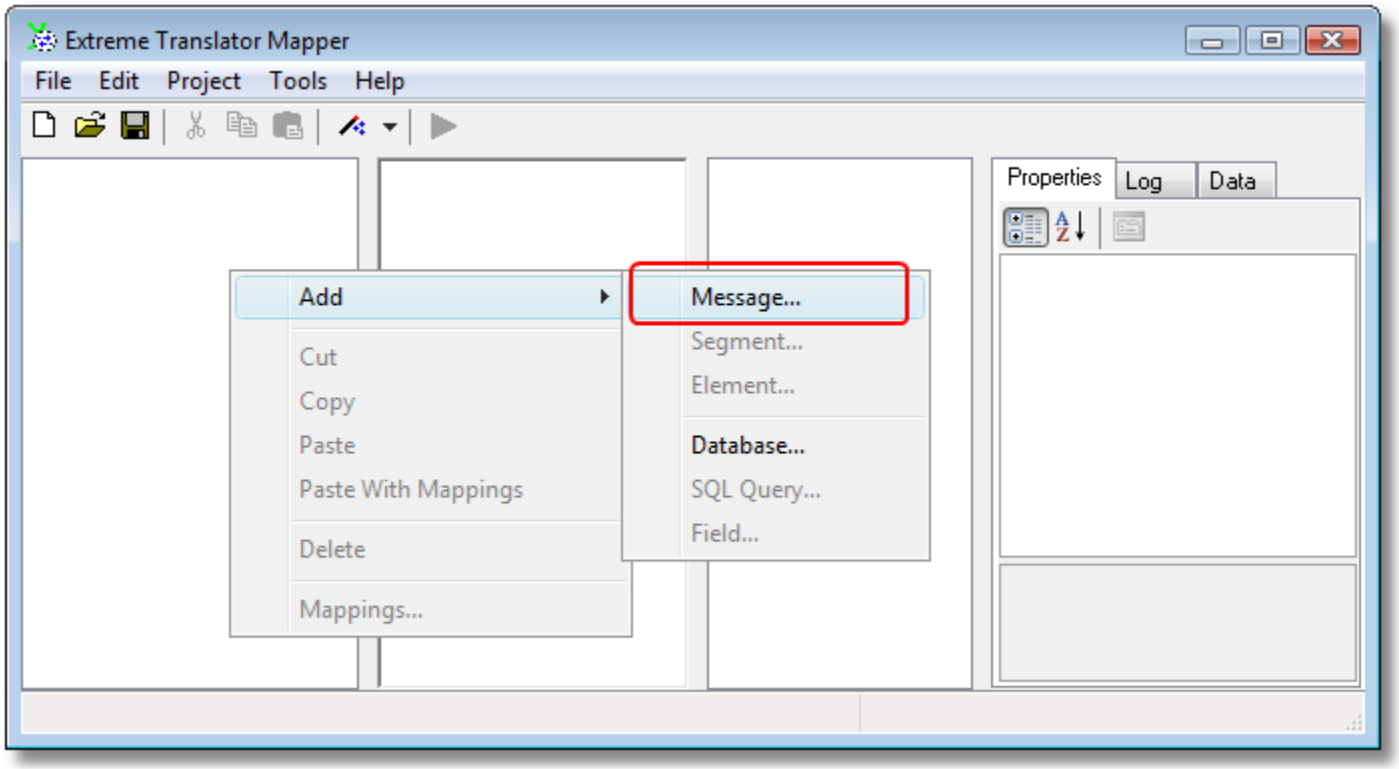
Translator supports one or many-to-one or many-to-many mappings. That means you may have one or many messages (files) mapped to one or many output messages. So you can have one or many root items on each side. This feature can be useful in complex mappings when you want to read input data from few different sources or output data into two different output destinations (such as write into output file and database at the same time).

If you just starting with the translator try to keep it simple, use only one input message and one destination message, that means – one root on first pane and one root on third pane.

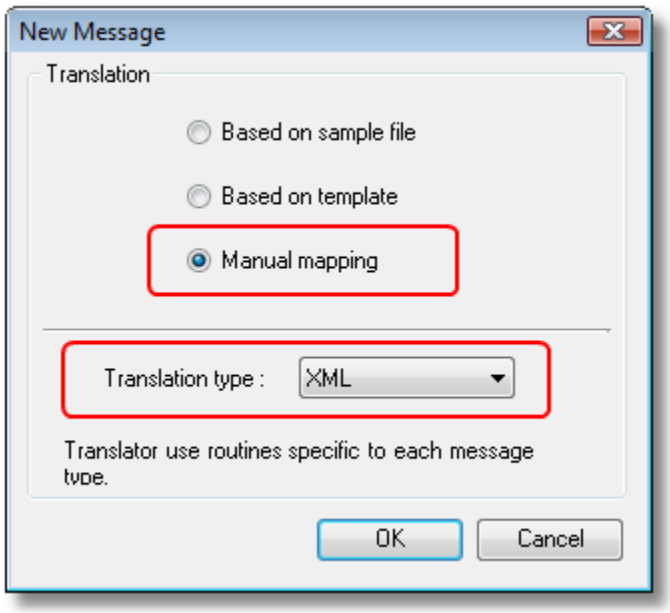
Restrictions when adding items manually:

1. Only segments can be added to the message.
2. Elements and segments can be added to segments.
3. Nothing can be added to elements. Elements cannot contain other items. If you need to map to sub-elements common in EDI X12 and EDIFACT formats use “Mappings” screen.

Start new manual mapping by right-clicking on the first pane, and choosing “Add” -> “Message” menu.



Add message to the map.

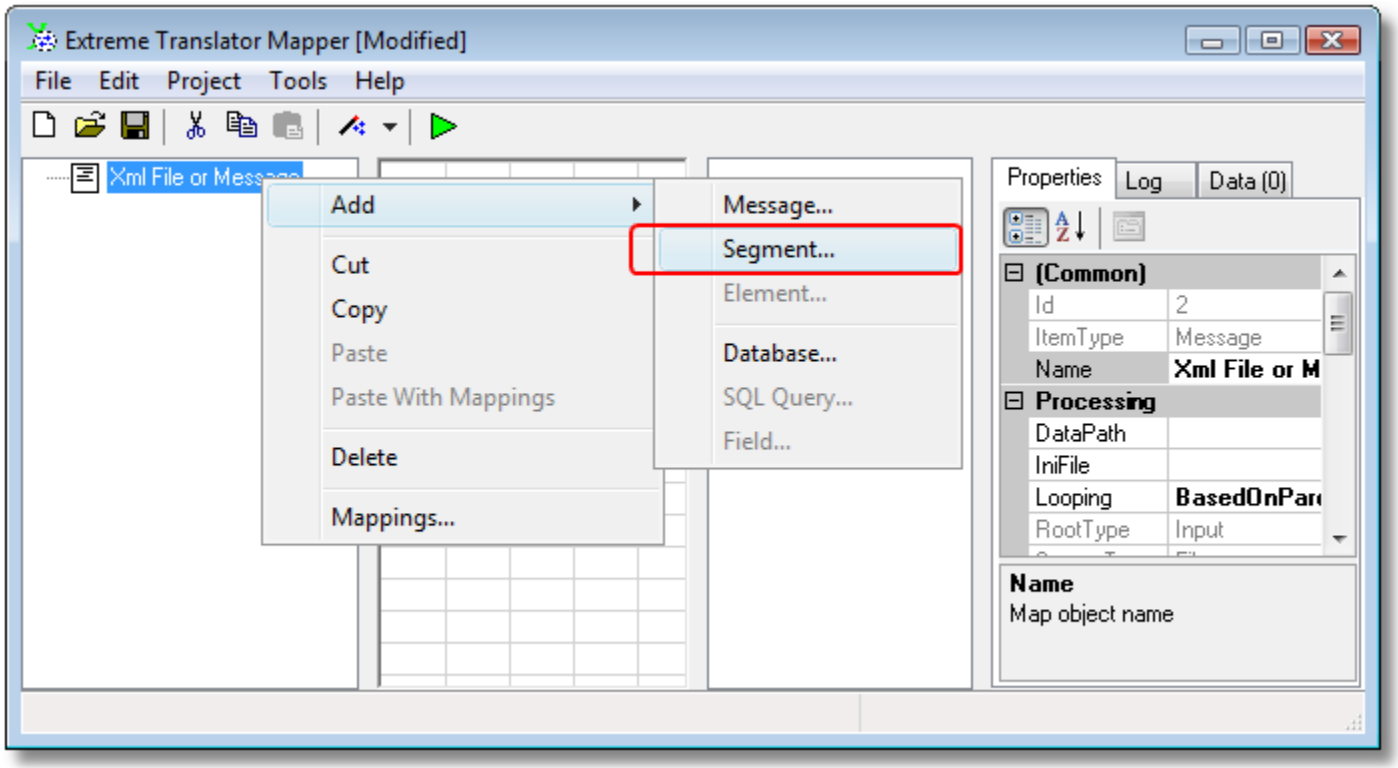


Choose translation type that is most accurate to the message (file) you want to process.

Manual mapping lets you add all the items to the map and set properties for each item in the map. This is most flexible but also most time consuming process. Consider using “template” or “sample file” instead for faster way to import data structures into the map.

In this example we choose to perform manual mapping for the XML.

If you have sample XML file choose “Based on sample file” instead.



Continue manual mapping. Add first segment under the message.

You can define whole map by manually adding items. It is much faster to use template or sample file to import structure into the map. But there are times when manual mapping is the only option: when you do not have sample file and file structure is in non-standard format.

You can also use Copy, Cut and Paste to move items around. All these actions can be applied both to first and third panes. So, you can copy items from one tree view and paste into the second.

Drag and drop using mouse can be performed as well, however you should be careful using it because it is very easy to drag-n-drop items and have them added to incorrect location. Sometimes it is even better to disable drag-n-drop. You can do it in under menu “Options”. If you hold “Ctrl” key pressed during drag-n-drop operation it will result into copied and pasted items in the new location. If “Ctrl” key is not pressed it will result into cut and paste operation.

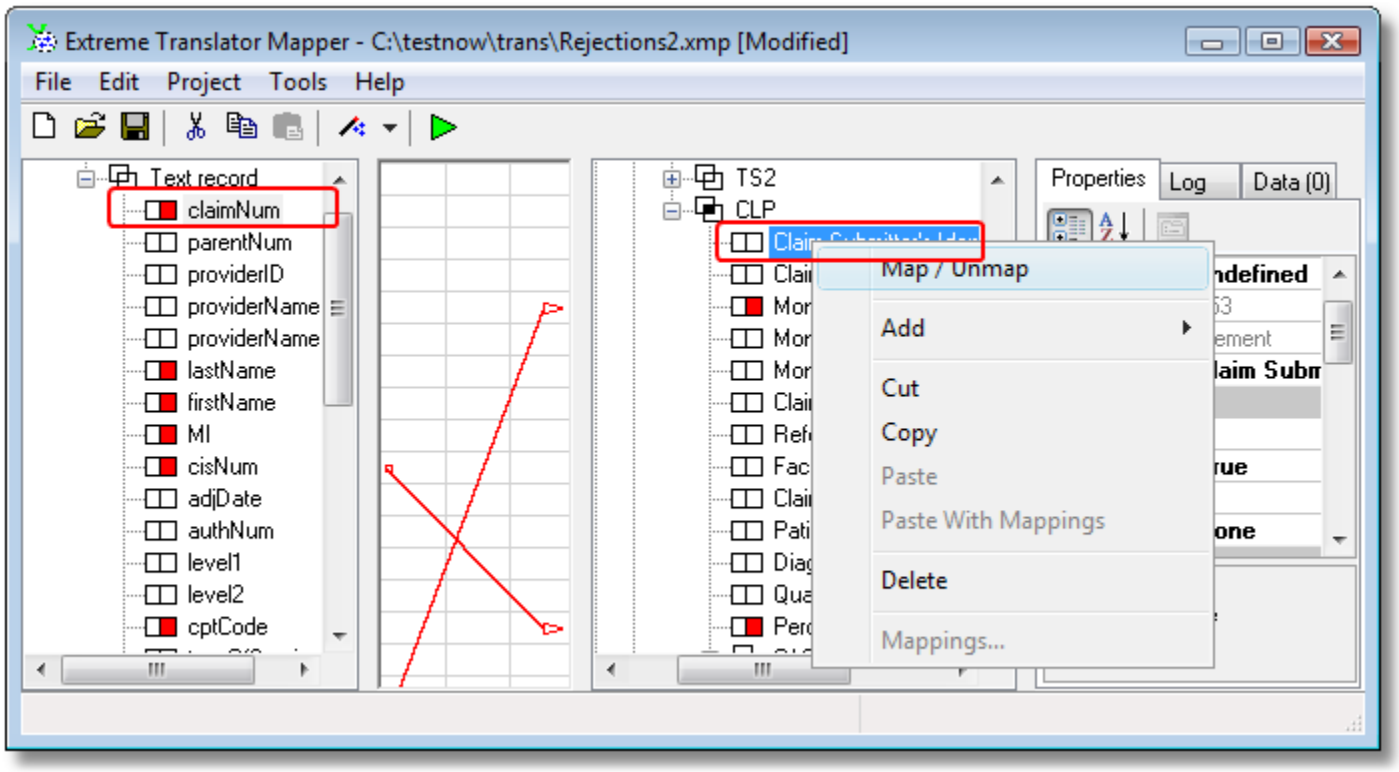
Note: You can map/un map items that are at least two levels down in the tree view. If it is root item or item directly attached to the root (nested only one level) Map/Un map menu option will be disabled. The reason is that you should define loops, SQL queries and segments under root. In case if you do not have any loops, just define at least one segment under the root and hang all the items under it. That way you can map them all and segment is just a placeholder.

You can view mappings displayed as red arrow lines. However lines are painted only if mapped item is visible (expanded) on both tree views. You can also edit mappings by clicking “Mappings” menu item on the left side of the screen.

How to do the mapping?

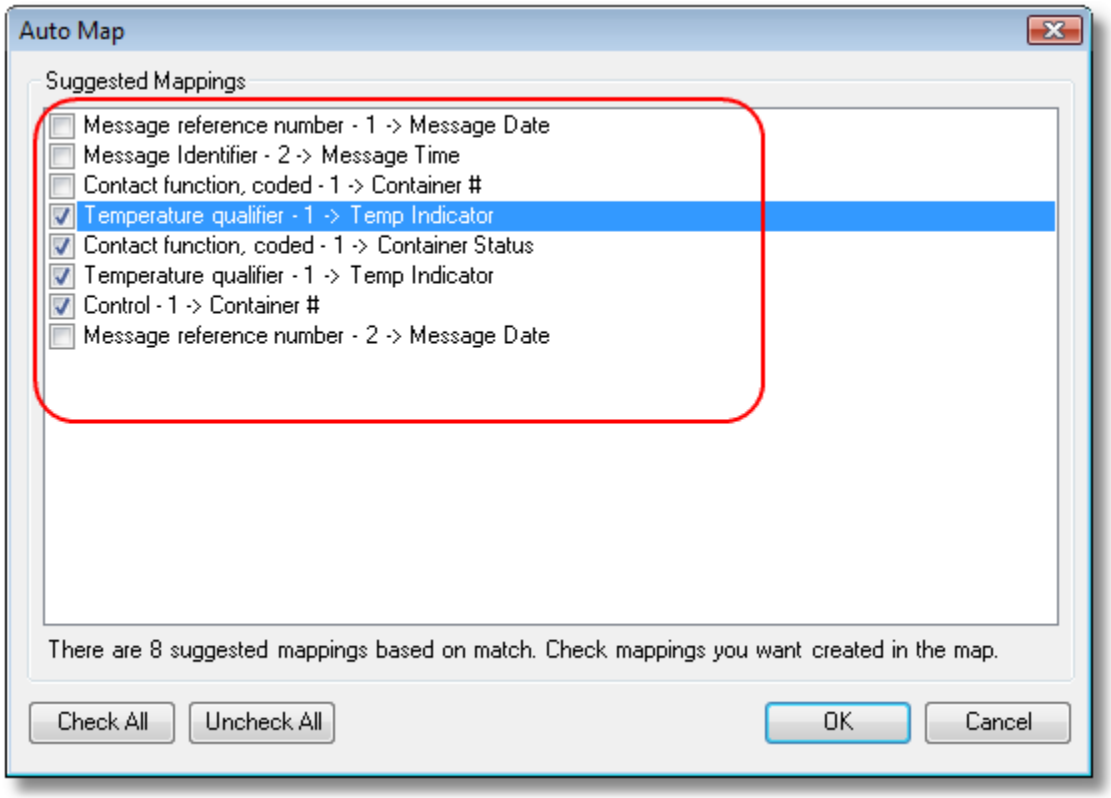
After you define both map segments and elements for input and output you can use popup menu in the right tree view. Follow these steps:

1. Click on the item on the left tree view
2. Click on the item on the right tree view
3. Press right button on the mouse to get popup menu
4. Click menu item "Map/Unmap".



Perform mapping using right click popup menu Map/Unmap on output side of the map.

If input and output field or element names are the same there is a way to create mappings faster using Auto Map feature. Find Auto Map under “Edit” menu. Auto Map suggests mappings based on translation type and matching item names on both sides of the map.



Auto Map feature in action. Once mappings are checked and accepted, they are added to the map.

.NET components

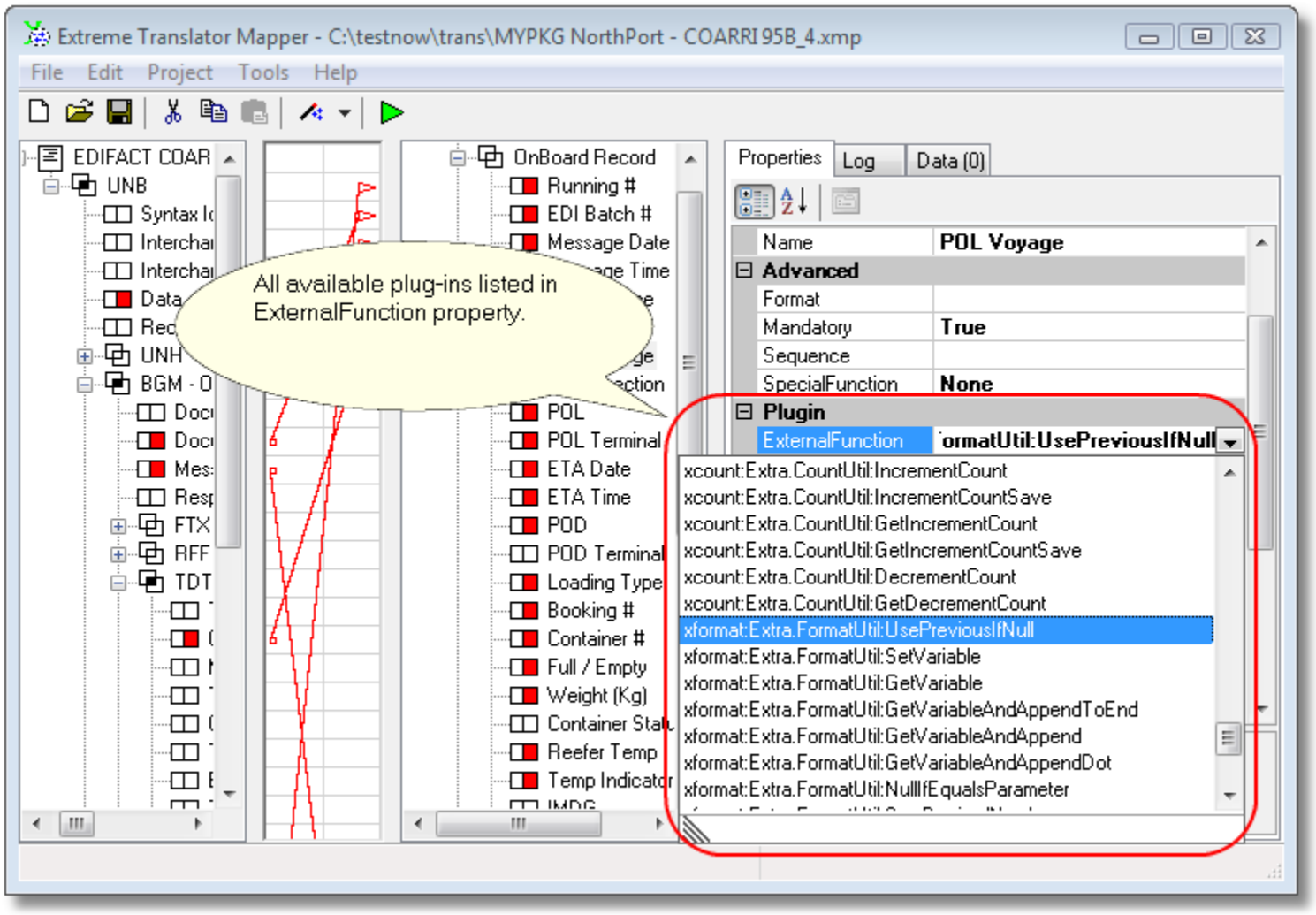
Integrate translator engine into your applications using .NET components and classes exported from the translator library (DLL). Use SDK and execute translations directly from the application without a need to run external processes via shell commands.

Find SDK C#.NET examples under \DeveloperSDK folder in translator installation directory.
Please purchase Developer license to use translator Software Development Kit (SDK).

Plug-ins and External Functions

You can extend translator with your own functions called “plug-ins”. Plug-ins are packaged into software libraries (DLLs) developed using C#.NET.

Translator comes with number of pre-built functions listed under “ExternalFunction” drop down list. Pre-built functions source code also included under \pluginsamples folder. Please take a look before building your own functions.



List of available functions stored in libraries in plug-in directory.

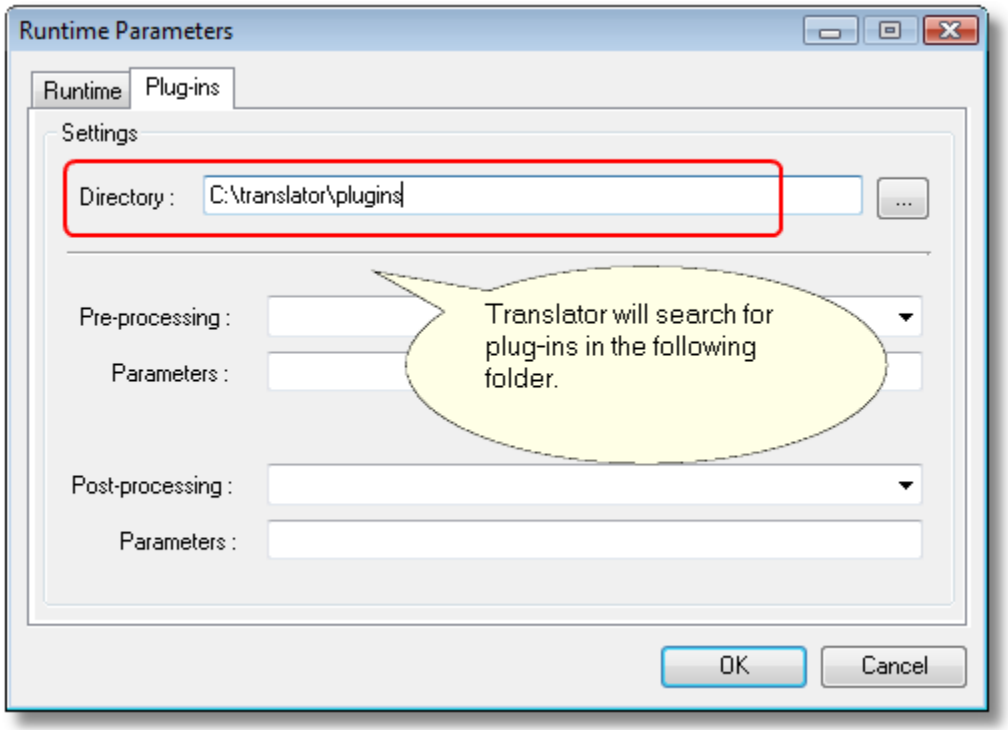
However you can use plug-in feature and add your own functionality to the translation. You do not need any special license to use and develop plug-ins. Special functions such as specific date time conversions, formatting can be achieved using external .NET libraries. Each element and segment in the map can have attached to external functions.

External functions must have certain interface to be integrated into the translator:

1. Should be compiled into .NET class library (DLL) and placed into /plugins directory.
2. Class must have default constructor that has no parameters.
3. External function method should have special signature: return string, and take three parameters – integer map object Id, actual processing data as string, and function parameters coded in the map as string as well.

Library can be built using VB, C++ or C#. /pluginsamples directory contains samples of plug-in functions in C#. You are free to copy the code and modify it for your own needs. Library can be built using IDE tools or build from command line. If you use command line tools, go to the /pluginsamples directory and in order to make a build type: csc.exe /t:library EDIMain.cs /nologo

Metadata about plug-ins is loaded into translator map editor only once during startup. If you update or drop new plug-ins library into plug-ins directory map editor will not list all the new external functions until you close and reopen editor again.



External functions are stored in subdirectory called “plugins” by default.

Plug-ins directory can be passed into translator batch process via parameter PluginDirectory. Example: PluginDirectory=C:\mytransalator\myplugins. In this case it will overwrite value used in the map “Runtime Parameters” dialog.

Pre-processing plug-in is called at the beginning of processing, and post-processing plug-in is called at the very end of translator processing.

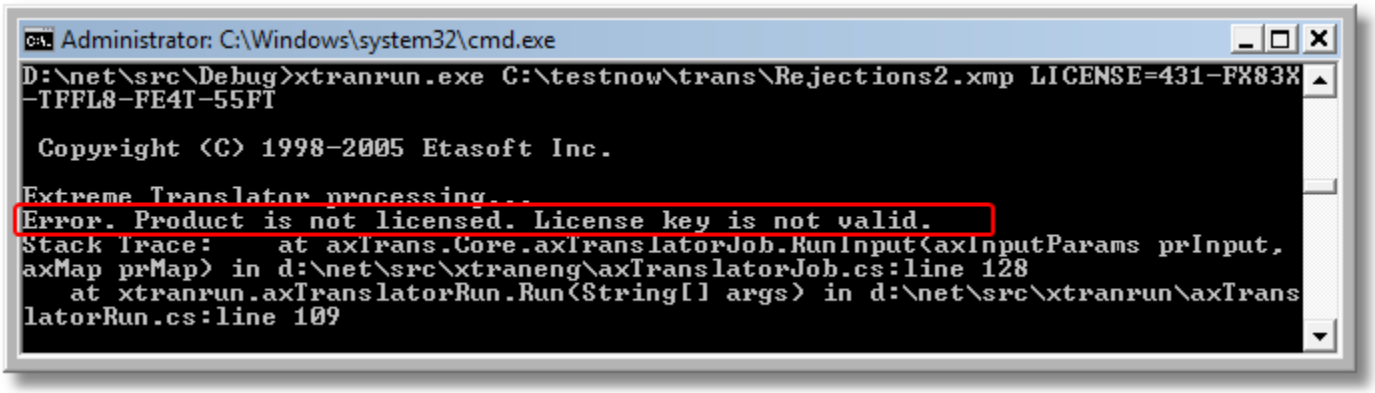
Use pre-processing and post-processing plug-ins to move, rename, delete files. When pre-processing plug-in is called passed in Id equals 0, and when post-processing plug-in is called passed in Id equals 1. If pre-processing plug-in returns **null**, it is treated as translation abort event and translation does not continue. This feature can be used to receive data via some network communication, and if data is not present, stop actual translation from execution.

Command line batch processing

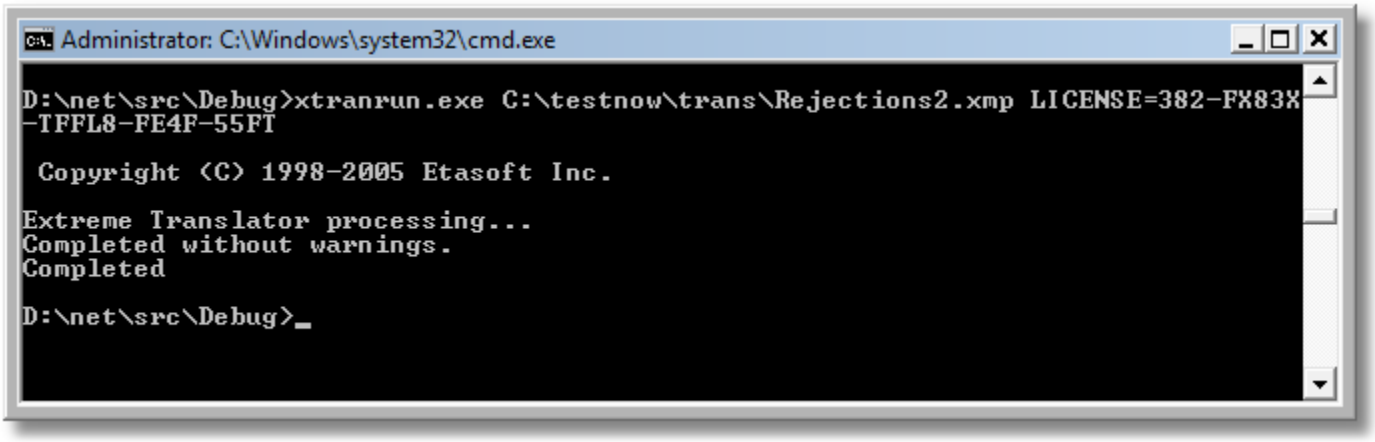
Use utility "xtranrun.exe" to run translator in Batch Mode. Find it in main application installation directory.
If you want to see accepted parameters simply run it without any parameters in the command line.

Batch utility returns 0 (zero) if processing has completed successfully and –1 (minus one) if it has failed.

Basic usage is
xtranrun.exe map_file_name.xmp License=YOUR_LICENSE [optional_datapath1] [optional_datapath2]



Running the map in command line with invalid or expired license key produces error.



Running translation with input and output file(s) set in the map.

Example 1:
You want to perform file translation, and you have made a map for it. If you pass Input and Output parameters you can process specific file each time. You can also use wildcards to process multiple input files.

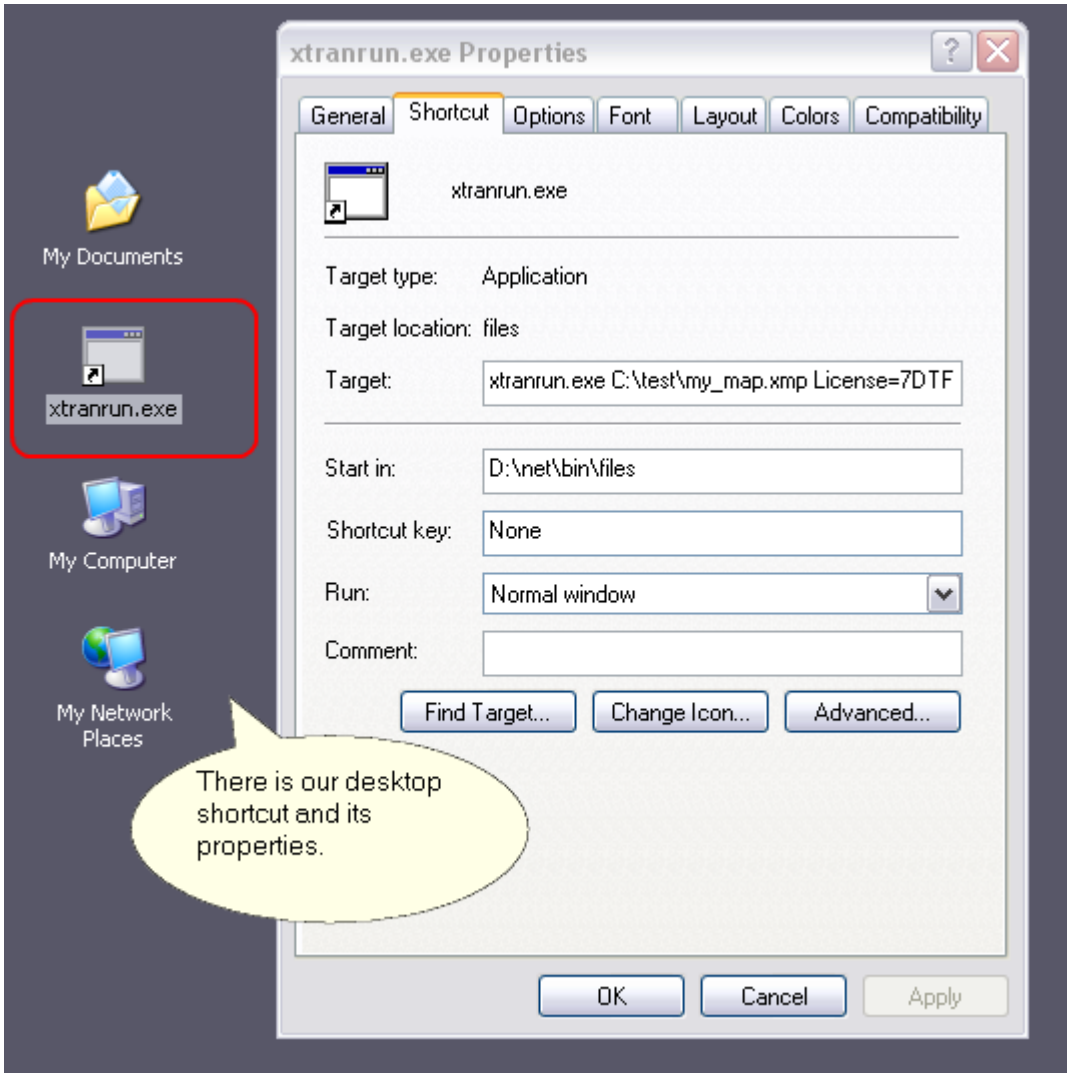
xtranrun.exe C:\test\my_map.xmp Input=C:\temp\data*.txt Output=C:\temp\output.txt License=YTRD3-34DFFZ

Example 2:
You do not want to use constants like in previous example. You want to hard code data path into the map for simplicity. In this case all you have to do is supply one parameter in command line and it is the map file name. PluginDirectory parameter lets change the folder name for the plug-ins.

xtranrun.exe C:\test\my_map.xmp License=7DTF-CDF74-DFA PluginDirectory=C:\myplugins

This assumes that you have put correct values into *DataPath* properties of the map and they are pointing to input and output files. Plug-in can be passed into translator batch process via parameter PluginDirectory. Example: PluginDirectory=C:\mytransalator\myplugins. In this case it will overwrite value used in the map “Runtime Parameters” dialog.

You can also setup shortcut on the desktop so every time it is double clicked, specific map would run.



There is an example of shortcut and properties. It will run C:\test\my_map.xmp translation when executed.

Passing parameters via command line

Most property values set in translator map can be overwritten by passing new values to them via command line.

This is very powerful feature if you need to do some pre-processing before map is called and would like to change map properties during runtime for each map execution.

Parameter passing might look something like "Id:138.Mandatory=true". Here “Id:138” means the item that has property Id equal to 138 should have its property Mandatory set to “true” during map execution. Whole command line may look like:

xtranrun.exe C:\test\trans\mapping_new2.xmp "Id:138.Mandatory=true" License=T3D-6UXF3D-U6LXF

Two or more properties can be passed in as parameters, for example: Mandatory and Format.

C:\trans>xtranrun C:\test\trans\mapping_new2.xmp "Id:138.Mandatory=true" "Id:138.Format==Value(TEST)" License=T3D-6UXF3D-U6LXF

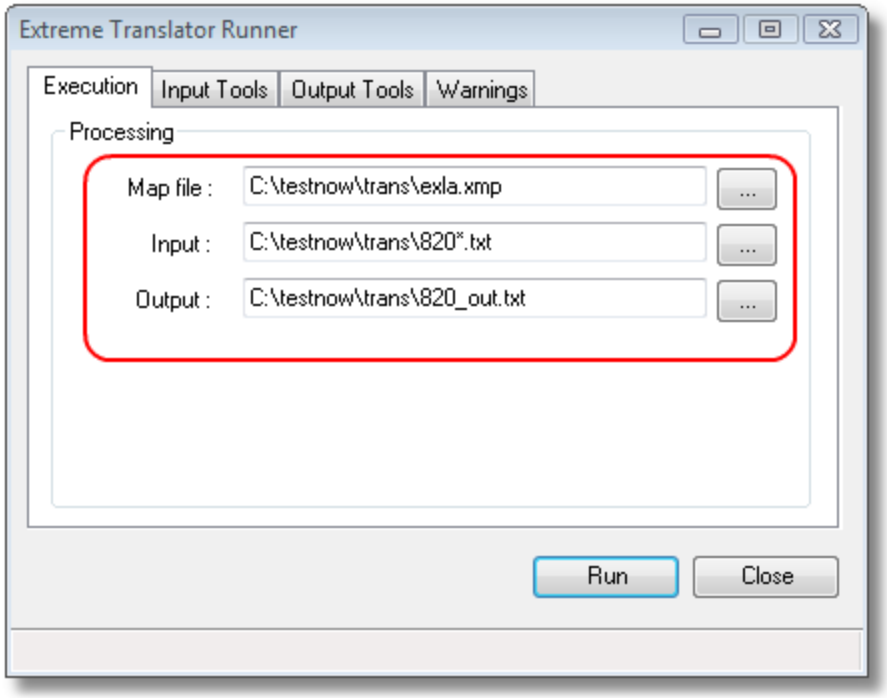
In this case Format property for item Id 138 is being set to =Value(TEST) string.

There are limitations to command line parameters that can be passed in to properties:

1. Item with specific Id must exist in the map in order for parameter to be accepted.
2. Property must not be read-only. It can be changed via Map Editor.
3. Property must be string, integer or Boolean value. Fixed dropdown list type properties cannot be reassigned or changed via command line.

Dialog based processing

Files can be processed using dialog screen when operator has to enter data and map file names. Use “xtranrunw.exe” to do that type of processing. This utility accepts map file name as parameter. If you start it with map file name in shortcut parameters, “Map file:” field will be pre-populated.

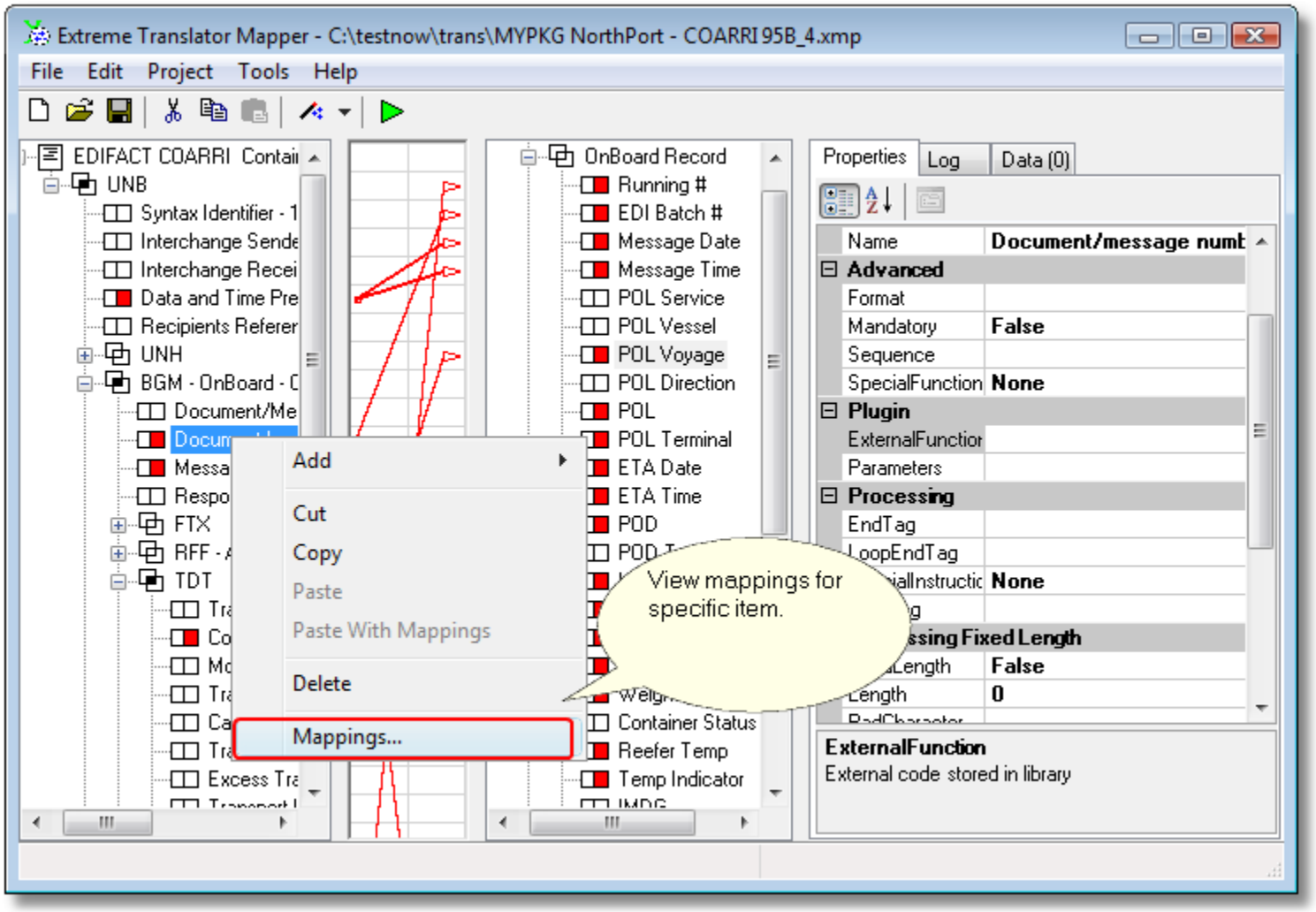


Dialog based processing.

Mapping Sub Elements

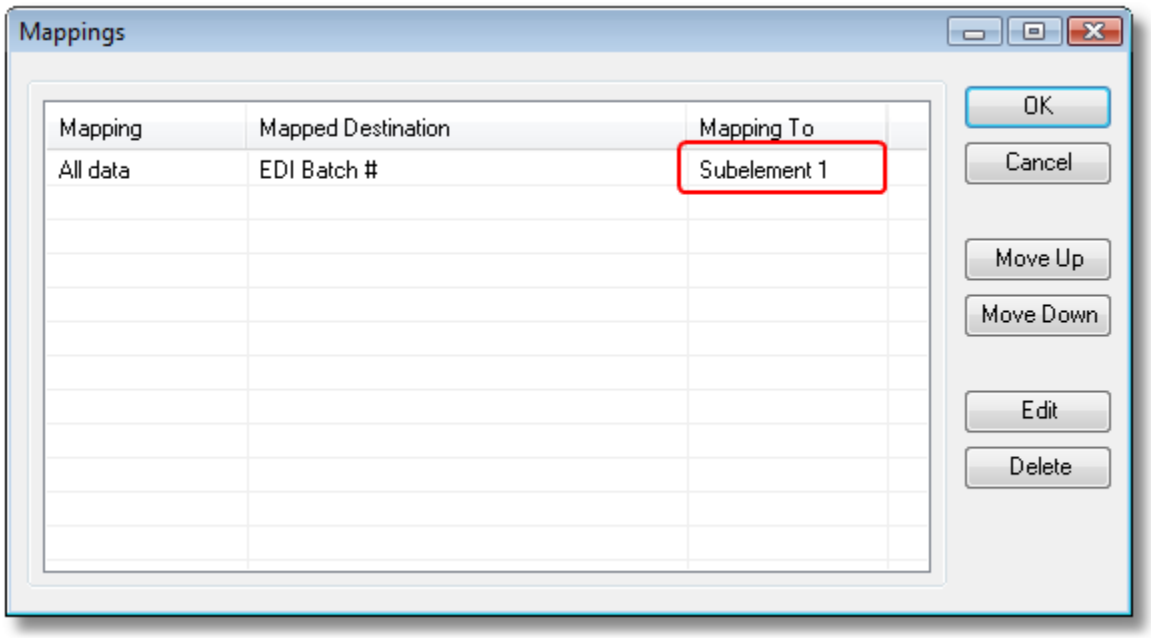
Some EDI message types contain sub elements. They are especially common in EDIFACT messages. Map Editor does not display sub elements as part of main view. Lowest level in the map is at element level. Mapping to sub elements is supported via additional “Mappings” screen.

When creating mappings to sub elements follow these steps:
1. Map elements that contain sub element(s) traditional way using “Map/Unmap” menu.
2. Right click on the input side element you have just mapped.
3. Open “Mappings” screen. Modify mapping to map “from sub element #” or “to sub element #”.



This is how to reach mappings for editing.

Mappings can be rearranged, deleted or edited. When you use menu option “Map/Unmap” all data from the item is mapped to the destination (output). But you can change this default mapping option and specify actual sub element you want to extract and use. This is especially useful for EDIFACT mappings where many data elements are composite of sub elements.

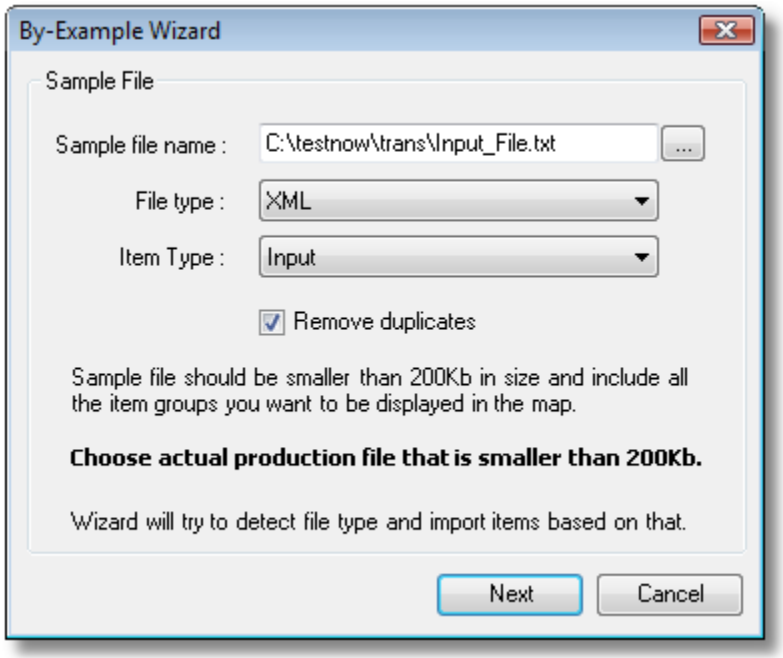


There is how you can map to or from sub elements.

Mapping Helpers

There are two mapping helpers (wizards): template and sample data based.

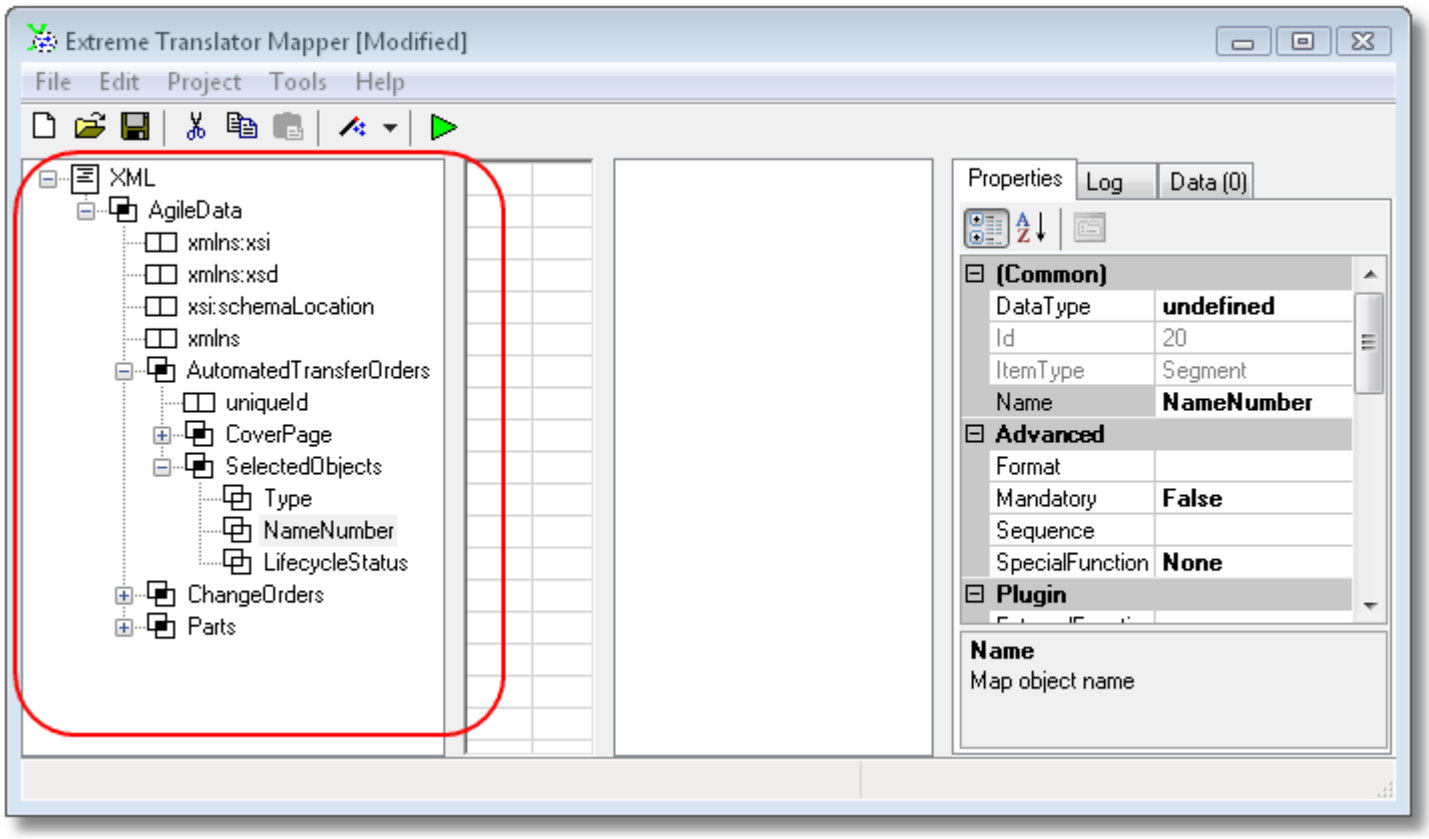
1. Translator comes with number of pre-built templates for EDI X12 and EDIFACT formats. Template Wizard helps you choose and load one of the templates.
2. By-Example wizard is a bit more complicated. It can read the file, detect file structure and create most of map objects based on your sample file.



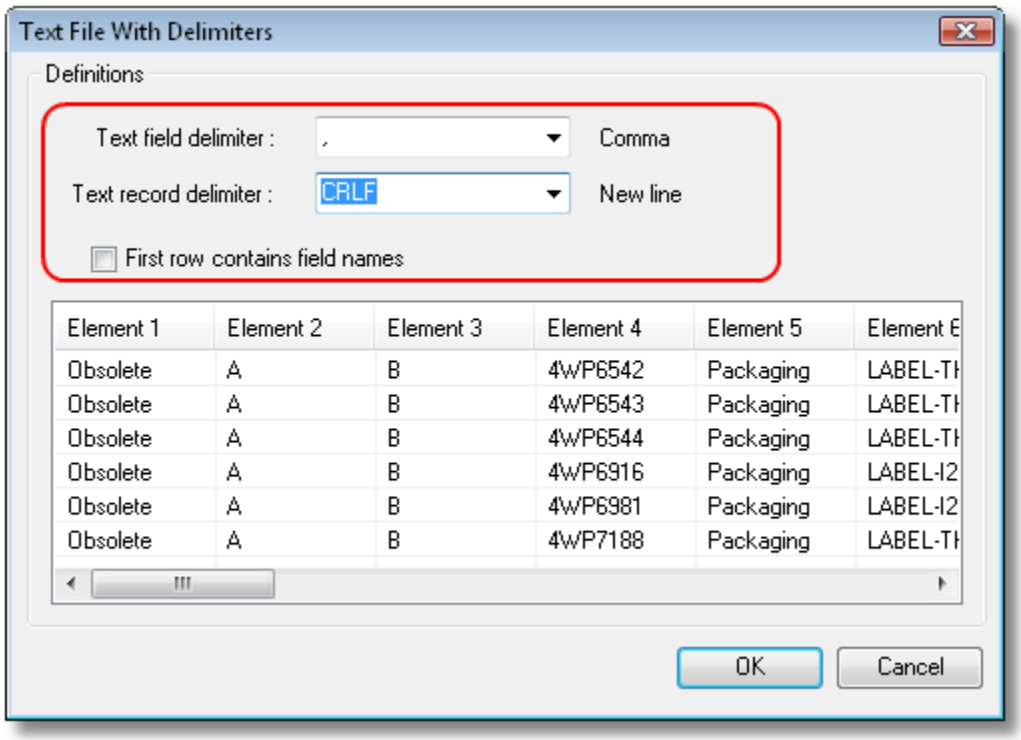
By-Example wizard takes sample file and loads structure into one side of the map.

There are certain requirements to the files you can feed into the By-Example wizard:
a) File should have all the items you expect to have in real production environment. In most cases we recommend to use actual file from production because it should reflect actual data you are sending or receiving.

b) File should be small. Recommended size should be less than 20Kbt. If you try to use files that are more than 100Kbt in size wizard will take a long time to process it.



Imported sample XML file.



If you choose “Text file with delimiters”, wizard will give you data preview based on delimiters you choose.

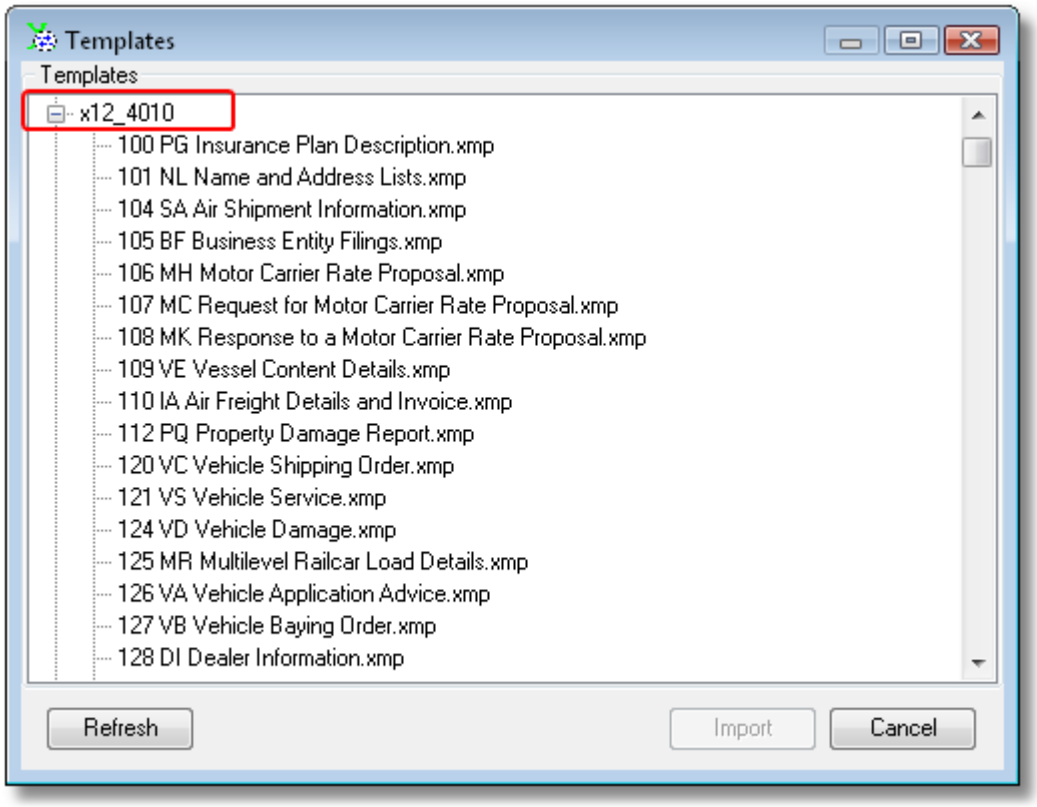
Templates

Software comes with over 5000 pre-built templates. Most templates are based on EDI standards.

Templates are grouped based on standard release version or date. Major standards are supported. For example: EDI X12 major standards are usually numbered as “xx10”. So EDI X12 4010 or EDI X12 5010 are major standards while EDI X12 5040 or EDI X12 4020 are minor standard releases.

Template Wizard contains major releases but might be missing minor releases. In many cases major releases are not different from minor releases in terms of mappings (segments and elements are the same). Therefore you can use major release instead of minor release when mapping.

For example: let say your task is to map EDI X12 5040 to CSV flat file. You notice that there are no templates for 5040 but there are templates for 5010. Use 5010 and make minor manual adjustments if required.

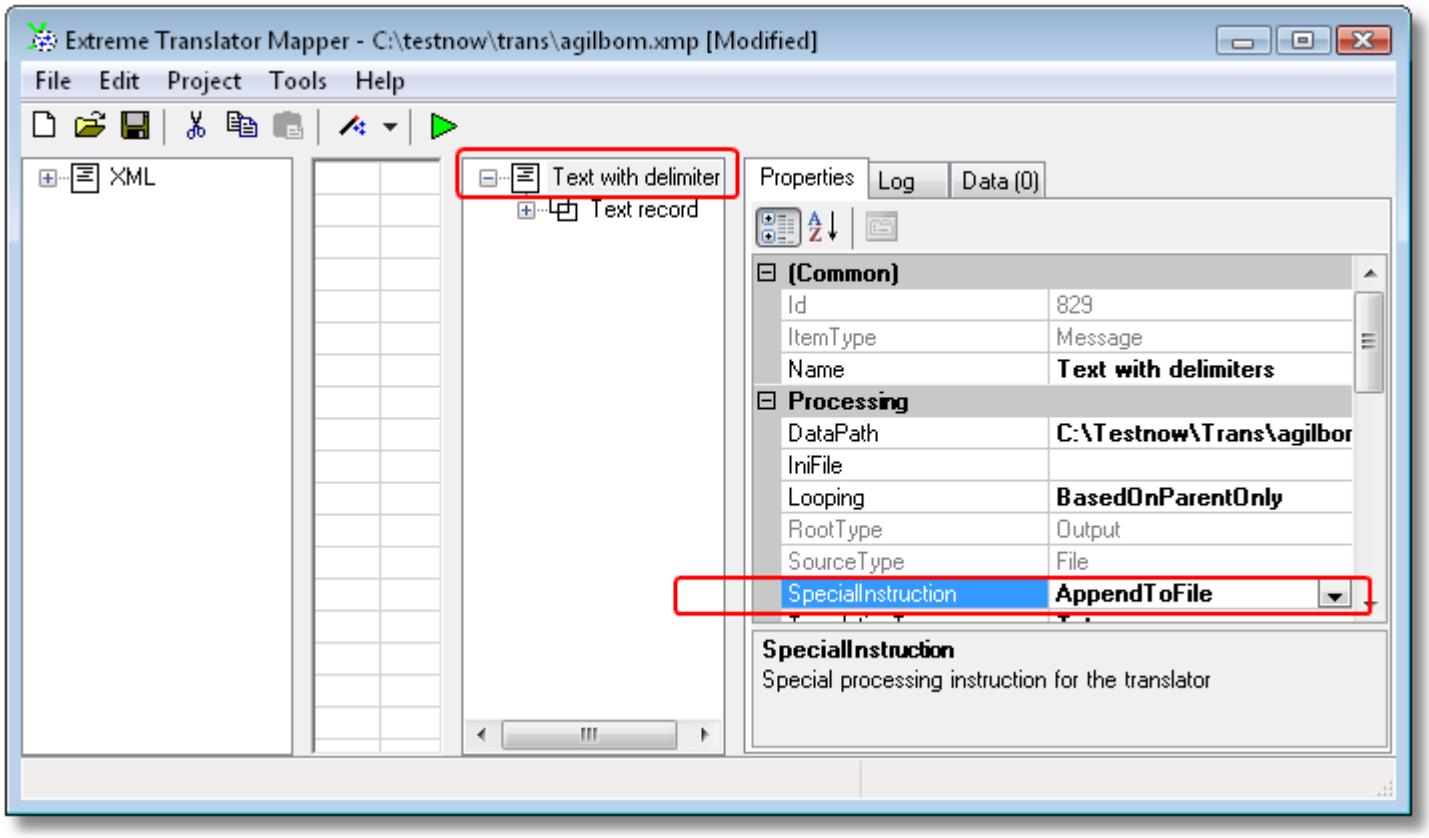


Import existing templates for most of standard messages of EDI X12, EDIFACT, SAP.

Special Instructions

All map items have SpecialInstructions properties. If used on segments and elements they help rearrange output to create proper looping (expected output ordering). There are two SpecialInstructions that deal with input and output files:
1. AppendToFile is used on the output side message (root item). If set translator will append to output file instead of default behavior of overwrite.
2. DeleteWhenDone is used on input side message. If set translator will delete input file when processing is finished.

By default translator creates (overwrites) output file when processing starts and does not delete input file. Two special instructions mentioned above let you alter default behavior.



Set this property on the output root item if you want data to be appended to the output file.

Getting Started With EDI

It is important to gather all available documentation and sample files before starting the mapping. Contact your trading partner and request all available documentation PDFs and EDI files.

Knowing only EDI message type is not enough for creating EDI map. Real world X12 files have too many corner cases and exceptions that might be specific to your trading partner EDI implementation. Not knowing specifics makes mapping process difficult.

If you are sending EDI files out ask your trading partner if they have test region where you could submit your files for verification before going into production. Many EDI systems have regions for test file processing. Test regions usually produce error reports and in some cases 997 acknowledgements.

If you are receiving EDI files ask your trading partner to send you at least few different sample EDI X12 files. Make sure to run EDI validation on samples. Files might be invalid because they might have been hand-edited and never checked. Operator took a file and tried to remove confidential information before sending it to you. During removal process file structure has been changed making it invalid EDI file.

When working with EDI X12 translations guiding rule is “trust but verify”. Trust your trading partner but verify validity of the files.

When creating EDI maps try using Template Wizard and pick existing template. Only if template does not exist use By-Example Wizard or perform manual mapping.

We have number of EDI mapping Guides online. Most Guides come with complete maps. If your message type is the same – take existing map from the Guide and use it to start your map. Modifying existing map is fastest way to get started.

If you are working on database to EDI translations avoid creating tables for each EDI segment. If you do not follow this advice mapping might result in 20-50 database tables map that is overly complex and slow.

Instead create database tables only for main loops. Basic EDI file usually has header information, details, sub-details and trailer. In case trailer does not have much useful business information (typical case) you will end up with just 3 tables: Header, Detail, Sub-Detail.

Important: Once you load template or load structure based on sample EDI file change DataPath property to point to your input or output files. You can also run maps even if only input side has data structure defined. In that case map will not produce any output but you can check Data Tab to see data loaded for input side items.

EDI X12, EDIFACT Translation

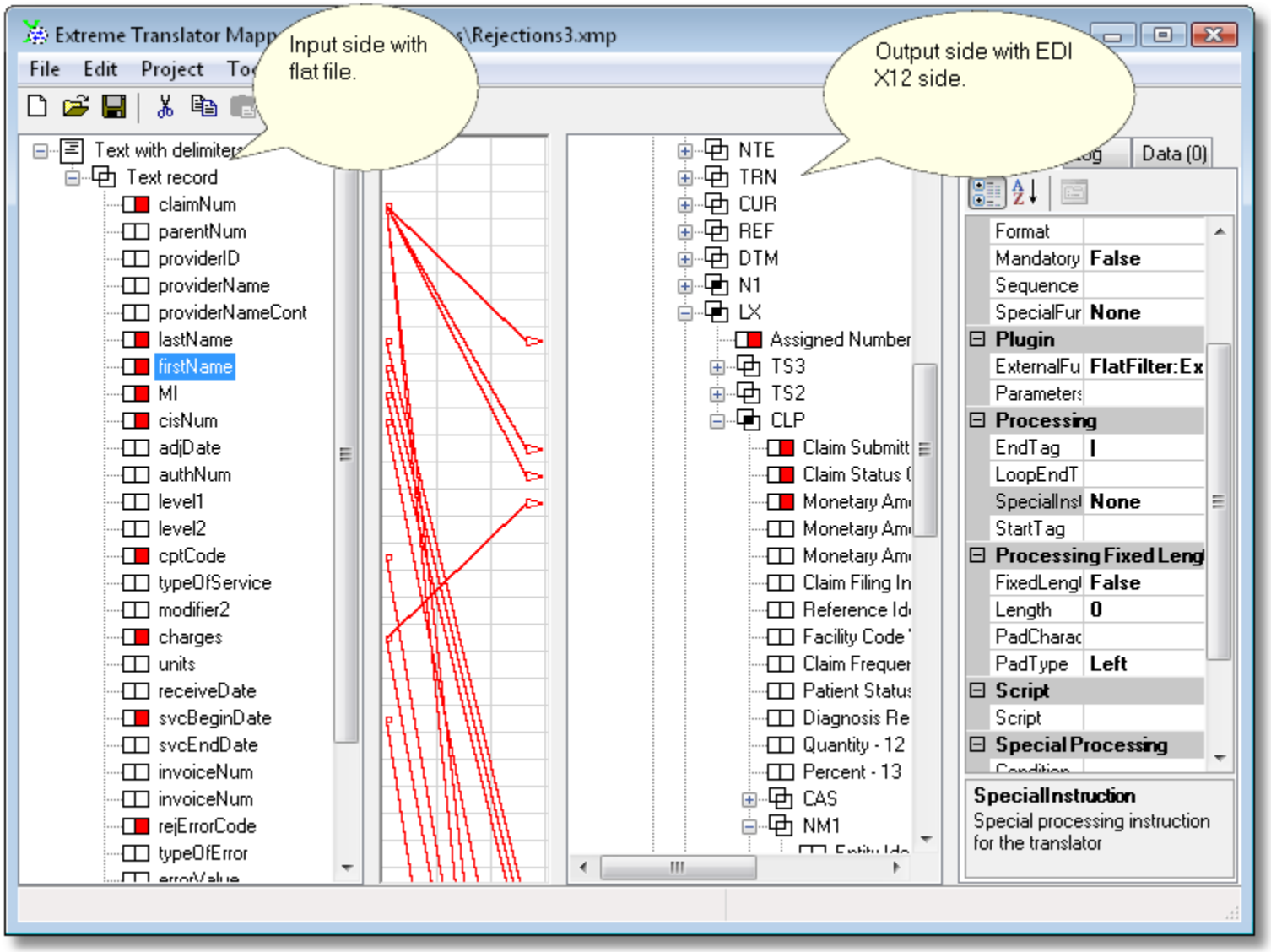
List of most important properties for EDI X12 and EDIFACT maps

StartTag	Used only for segments (loops). It should be setup to segment name, example: ISA, ST, SE, UNA, UNB, etc.
EndTag, LoopEndTag	Not used
Mandatory	Should be setup “true” for mandatory EDI or EDIFACT items, such as ISA, GS, ST, UNA, UNB, etc.
AutoDetect	If setup to “true” will try to locate segment and element separators
UseSep	If setup to “true” will use separators setup in other properties. It is opposite to “AutoDetect” property
SegmentSep, ElementSep, SubElementSep	Separators coded in the map. Used only if “UseSep” property is setup to “true”
Filter	Can be setup to #13#10 to filter carriage return and line feed characters from the incoming data

Product supports EDI X12 and UN/EDIFACT. For EDI translation you need to have separators defined at the root level of the message.

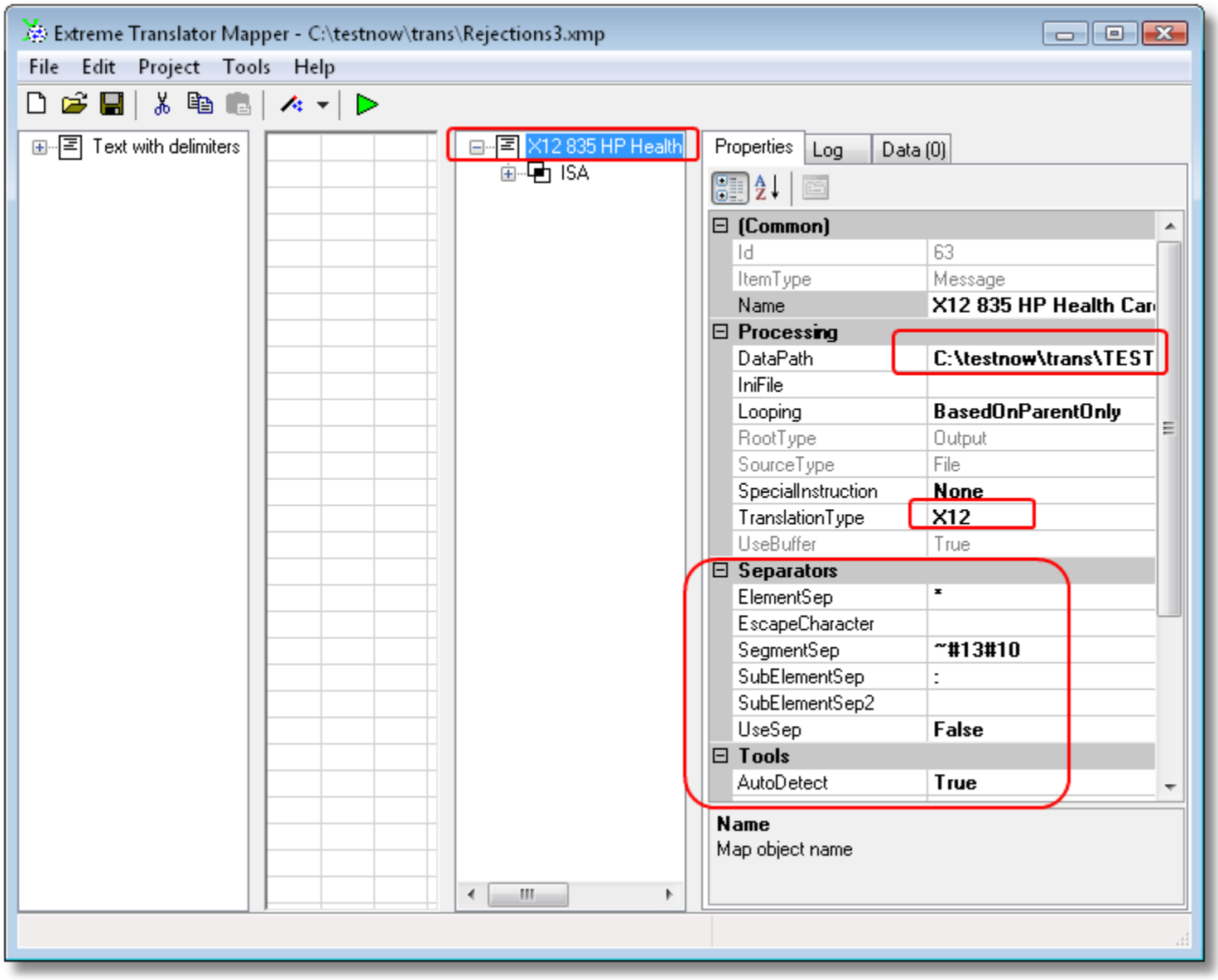
If you do not know expected separators, or you expect EDI files with various separators, use option "AutoDetect" and set it to "true". Separators are special characters that separate segments and elements in EDI document. You can find all three separators: sub element, element and segment separator, defined in the EDI X12 file at the location starting position 103 after ISA. If you use the wizard separators will be defined for you.

In EDI X12 and EDIFACT mappings only elements should be mapped to output.
If you need to map sub-elements use “Mappings” screen covered in chapter “Mapping Sub Elements”.



Flat file to EDI X12 mapping example.

If your EDI message type is listed in the set of available templates use it to start your new mapping. If your EDI message is not in the list of available templates because it is unique non-standard EDI try By-Example Wizard. If By-Example Wizard cannot recognize your file manual mapping is only option.



Properties that should be set for EDI messages

Important properties relate to EDI separators are listed in Separators group. You should setup ElementSep, SegmentSep, SubElementSep if you setup UseSep to “true”. If you do not setup UseSep property to “true” you should set AutoDetect to “true” in that case translator will try to find separators in default locations: ISA header in EDI X12 or UNB header in EDIFACT. We also setup Filter property to filter carriage return and line feed characters (decimal codes 13 and 10).

Let say this is a fragment of EDIFACT we want to translate:
UNB+UNOA:2+SGSIN+HOITD+021220:0953+SINFO019'
UNH+1+IFCSUM:D:01A:UN:UAS120'

Mapping is based on two segments on the input side called UNB and UNH. There are some composite data elements that have sub elements we need to extract. Examples would be Element 1 in UNB or Element 2 in UNH. In order to extract sub elements mappings should be edited using menu option “Mappings” on the popup menu when you click item on the left.

We also want to output carriage return and line feed characters at the end of each segment. Since carriage return and line feed are invisible characters only way to enter them is via special codes. #13 code denotes carriage return and #10 code denotes line feed. Please see screenshot above.

When SegmentSep is set to ~#13#10 that means: end segment with 3 characters: tilde, carriage return and line feed.

HIPAA Support

HIPAA in EDI X12 is supported through the map templates.

EDI X12 - 270 Eligibility, Coverage or Benefit Inquiry
EDI X12 - 271 Eligibility, Coverage or Benefit Inquiry
EDI X12 - 276 Health Care Claim Status Request
EDI X12 - 277 Health Care Claim Status Notification
EDI X12 - 278 Health Care Service Review
EDI X12 - 820 Payment Order Remittance Advice
EDI X12 - 834 Benefit Enrollment and Maintenance
EDI X12 - 835 Health Care Claim Payment Advice
EDI X12 - 837 Health Care Claim

Please find additional Guides and sample mappings for HIPAA formats inside Documentation section on our website XTranslator page.

HIPAA mappings are essentially EDI X12 mappings.

Properties

You can see there are many properties describing each item in the map. As you click on the item all properties are displayed and you can change them. Some properties are common to all objects, such as *Name*, *Id*, *DataType*, *ItemType*.

There is the list of most used properties and they explanations:

Name	The name of map object, can be any name defined by the user. You should use names that are meaningful
DataType	It is the data type of the object. It can be “String”, “Integer”, etc. It is used for conversions and in special functions. You may leave it “undefined” unless you are going to use special functions
Id	Internal object ID used by translator
Format	See special chapter “Format Property”
Mandatory	This item must be present. This especially useful during export when you want certain elements produced in the file if other elements are produced. However you cannot make all the elements in the segment mandatory because it would create infinite loop during export
Sequence	Name of the sequence to be used to generate unique numbers. Useful at times when translator should produce EDI X12 control numbers. Other property called *IniFile* should be setup to get unique numbers.
IniFile	Path and file name of Initialization file. It will be used during translation to generate unique number and maybe used in other functions as well. Example: “C:\temp\editrans.ini”.
SpecialFunction	Extra functions. You can use them to get total segment count, SE count used in EDI X12. Also get current input and output file names.
DataPath	Location of file or data for input or output. You can override this

	property by passing parameters in command line. File names may contain wildcards, such as * or ?. Example: C:\test\trans\inputxml* would pickup all the files having names start with “inputxml” in that directory. Wildcards cannot be used in directory part of path (see special chapter “DataPath Property”).
Looping	BasedOnParentOnly – default looping model when each output item repeats in synchronization with its parent one level up. BasedOnAllParents –looping model when each output item repeats in synchronization with all of its parents up.
TranslationType	Translator performs certain optimizations on some file formats that it has special routines for performance and validation. “Txt” type is most generic and should be used only in cases when no other type matches.
UseBuffer	*Reserved for future use*. Means that translator will cache data in memory during processing. This is good option for small files, less than 1Mbt in size. This property will allow optimize translation for large files. Default is “true”.
ElementSep, *SegmentSep*, *SubElementSep* *SubElementSep2* *EscapeCharacter*	Are separators used in some translations, such as EDI X12, EDIFACT. They separate data segments, such as ISA, GS, GE, UNA, UNB, etc. EscapeCharacter is used in EDIFACT for special cases when SegmentSep should be treated as simple data character and not a segment separator. SubElementSep2 is used in HL7.
UseSep	Indicator to use hard coded separators
AutoDetect	Automatically detect features of the file just before processing and use them during file processing. For example in cases if those files are EDI X12, EDIFACT files separators fill be detected automatically. This property overrides other properties such as *UseSep* property for example. However in some cases AutoDetect is impossible. Example: your EDI X12 files come without ISA header. In this case translator will not be able to find separators and translation will fail.
Filter	Can be used to filter garbage characters from the file or message. It is mostly used for removing CRLF (carriage return and line feed characters) from EDI X12 and EDIFACT messages, decimal codes #13#10. Some junk characters are being added by graphical file editors for display purposes and saved back into the file. In many cases they are not valid and do not contain any business data and serve no purpose during processing.
Substitute	Can be used to substitute characters in the input with other characters or strings of characters. Example of use: 1=TEST,2=TEST2 this will replace all instances of 1 with word TEST and all instances of 2 with word TEST2.
Remove	Can be used to remove strings or characters from the message. Regular Expressions can be used in this property.
TrimSpaces	Trim spaces from the end or start or both sides of each data element. Can be used for translation of fixed length flat files into other formats.
NullValues	If set to “Ignore” translator will skip incoming NULL values. If set to “TreatAsBlankValues” translator will treat NULL values as empty values. This setting is mostly used for mappings from

	database in cases when non existing field value should still be produced in the output as spaces or blank value.
ChangeCase	Change case on each data element. Can be used when translation should output data only in upper or lower case.
Encoding	Used to define of characters during translation. Use ASCII_7bit for English, European_8bit for European languages and Unicode for languages that are Unicode based.
SpecialInstruction (use of this property is also discussed in additional document related to looping issues and it is available as separate download on our website)	Tells translator to perform special operations during data translation FlatOutput – special instructions can be used on input elements. Tells translator to output looping element inline. See EDIX12-to-FlatFile example. It can also be used on output elements of flat file. See end of Text Translation chapter. AppendToFile – instruction to append output to file rather than overwriting and creating a new file (use on root item only). DeleteWhenDone – instruction to delete files after processing has finished (use on root item only). OutOfLoopData – instruction for translator do not pay attention to sequence of this element in the loop. It can be used on looping elements that are inside of the loop but data into them is coming from other loop (use on elements only). It should be used only as exception in cases when default looping does not work correctly. FlatOutputInline – it can be used on output element to make it line up correctly in the output file.
ConditionType, Condition	Two properties form simple conditional “IF” statement. (Currently implemented for TranslationType = X12, Edifact, XML). You can specify multiple conditions under one segment by simply entering them separated by commas.
ValidationType Validation	Type of validation to be performed. ValidationType is MustEqualTo then Validation should be the value you expect in the input ValidationType is SchemaFile then Validation should be path to XML Schema file for validation.
StartTag	It is a generic name for data block that indicates start of segment or element. If it is used for input it indicates boundaries where data starts. If it is used for output it indicates what will be placed to output before the data.
EndTag	It is an end marker for data block. Should be used for items that have no nested items attached. It is supplement to *LoopEndTag*.
LoopEndTag	It is an end marker for the block of looping data. Should be used for items that have nested items and those items have nested items too.
SQL	Free form SQL statement. It must be used in queries where SQLType set to Select. Can also be used in queries with SQLType InsertOrUpdate or Update. In this case SQL is added to WHERE clause for database Update. Example: if you want to update records only if they have field State=1, in this case your SQL should be “ AND STATE=1”.
SQLType	Select should be set for database query used on the input side.

	All other flag values are for database tables on the output side. Translator will use first PrimaryKey field in the database to perform Insert, InsertOrUpdate and Update operations. InsertOrUpdate is a combination of possible operations on the database where translator will select row from the database based on PrimaryKey, and if it exists translator will perform update otherwise insert will be performed.
MasterQuery	Name of other query in the map that will act as Master to form Master-Detail relationship and supply parameters to this query.
FieldType	Represents relational database enforced type. It is used to reorder and populate resulting records.
Size	Database field size.
FixedLength	Indicates to treat item as fixed length
Length	Actual length of item in characters
PadType	Dictates how fixed length item is padded with pad characters
PadCharacters	Character to be used for padding
ExternalFunction Parameters	Translator will call external routine stored in .NET assembly to get data. External routine must have certain function signature (see Plug-ins chapter in this document). Parameters will be passed into external function.
Script	Translator will call script that was defined via Scripts screen in Map Editor. Each script is compiled once before it is executed for the first time during map processing.

DataPath property

DataPath property is used on root items of the map for both input and output. It is generic name for data source that can be local file, file on the Internet accessible via http, ftp protocols or database connection string.
There are different rules what should be in this property depending on what is actual data source or destination. It is recommended to test processing using local files first and move them to ftp or http servers later.

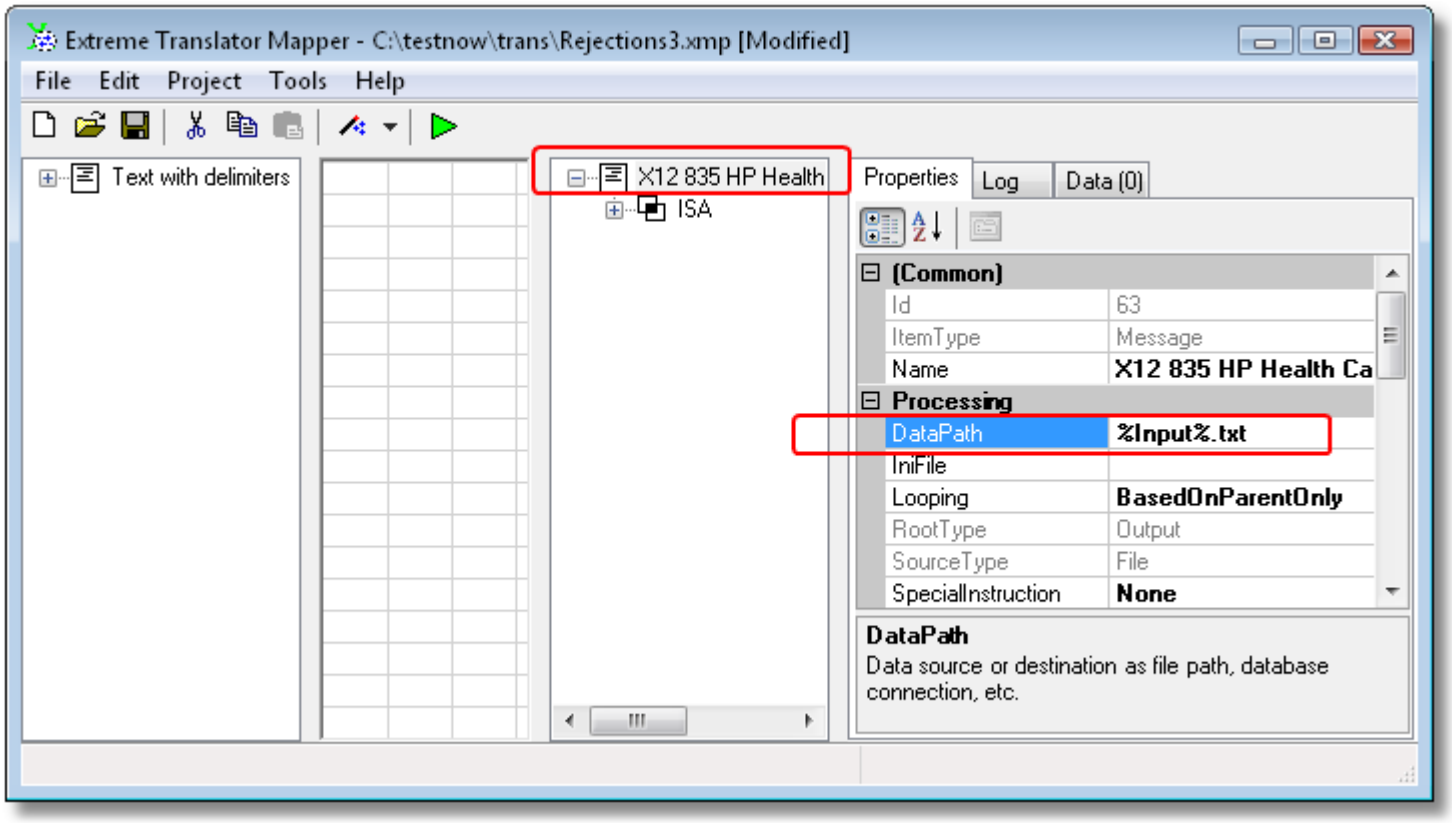
Local or network file(s)	DataPath property can be a path to the file including file name. File name also may contain wild cards like '*' and '?'. Example: C:\test\trans*.txt would pick up and translate all the text files in C:\test\trans directory. It can also be a shared directory on another computer in local network, like \\MYSERVER\testing*.txt
Internet file	DataPath property can be an URL to the file on the web. It should start with string "http:". Example: http://www.somewebsite.com/somedata.xml Translator would pick up XML file from the web site. Wild cards are not allowed. Translator uses HTTP GET to retrieve file and HTTP POST to submit files to the web server.
Database connection	DataPath property can be a connection string to the database. Example of connection string to ODBC data source: DSN=mydatasource;UID=myusername;PWD=mypassword
FTP file(s)	DataPath property can be a connection string to file on FTP server. Connection string can contain user name, password, FTP server address and remote server directory. On the input side DataPath may contain file pattern for the file to be picked up

	ftp://username:password@ftpserver.com/directory/filepattern Example: ftp://myuser:mypassword@someserver/test/.txt It would connect to “someserver” using user name “myuser” and password “mypassword”, change to directory /test and pick up file that has “.txt” in the name. On the output side FTP file should be exact file name ftp://myuser:mypassword@someserver/test/myoutputfile.txt If login should be anonymous, user name and password might be omitted. Example will fetch all XML files from FTP server called “someserver” and remote directory “somedirectory” ftp://someserver/somedirectory/.xml SpecialInstruction = DeleteWhenDone can be used on input side in order to remove processed file from FTP server. SpecialInstruction = AppendToFile is not available with FTP on output side. Files with extension “txt” are transferred in text mode. Binary transfer mode is used for all other files.

Special macro can be used to place input file name into the output and have all the output file names formed based on input file names. Macros should be used on output DataPath property.

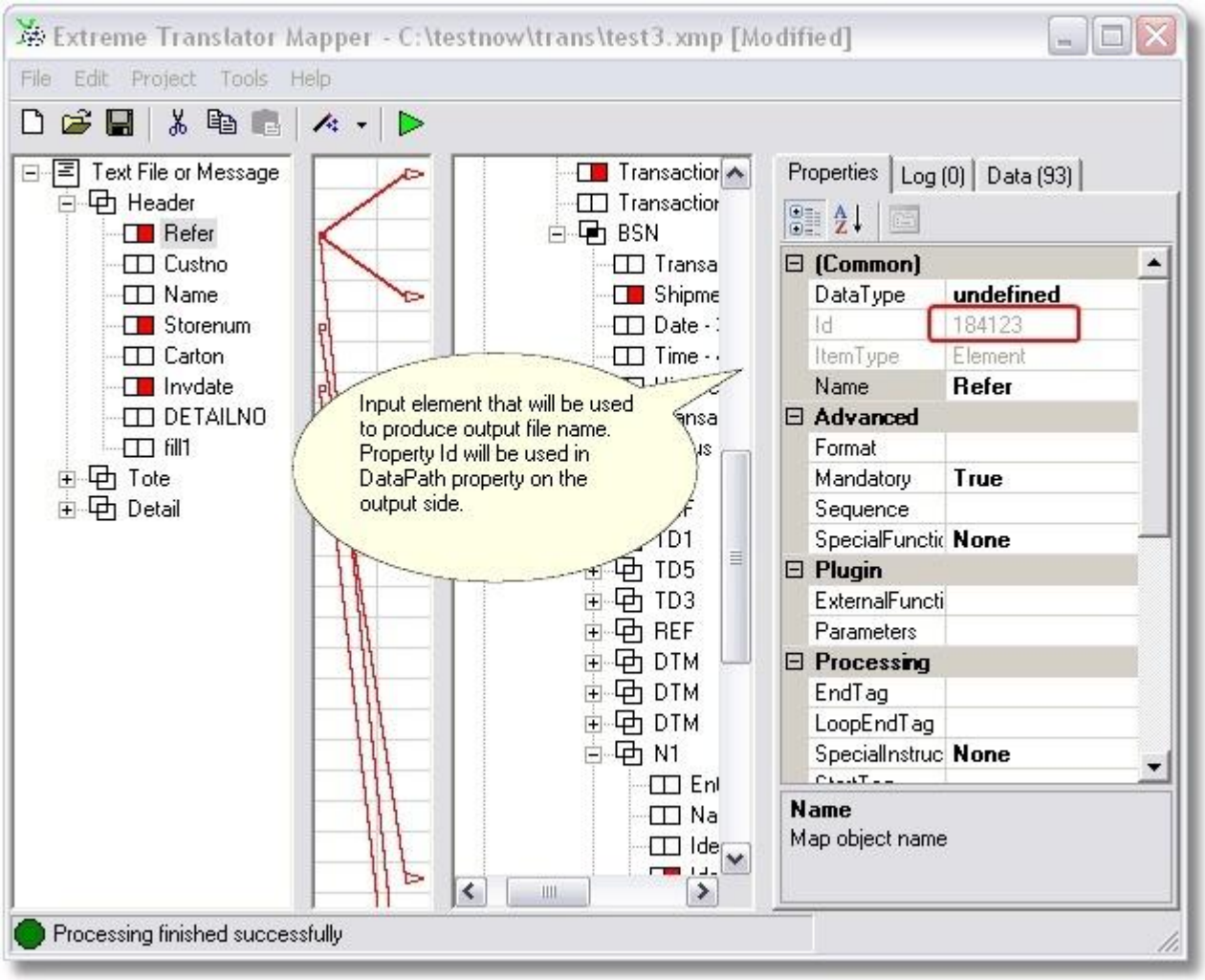
%Input%	Translator drops file extension from the input file, therefore any extension can be used in output DataPath property to form output file.
%InputFileName%	Translator drops file extension and directory from the input file, therefore any extension and directory can be used in output DataPath property to form output file.
%Count%	This macro is replaced by internal file count during processing.
%SystemDate%	This macros is replaced with current date in form CCYYMMDD
%SystemDateTime%	This macros is replaced with current date and time in form CCYYMMDDhhmm
%#propertyID%	This macro is replaced by actual processing data from input or output. Example: if DataPath is set to C:\test\output%#187645%.txt translator will replace %#187645% with the first value that item with property Id=187645 has during translation.

Important: Macros are case sensitive. They can be used only in one-to-one mappings when one type input message is mapped to one output message.



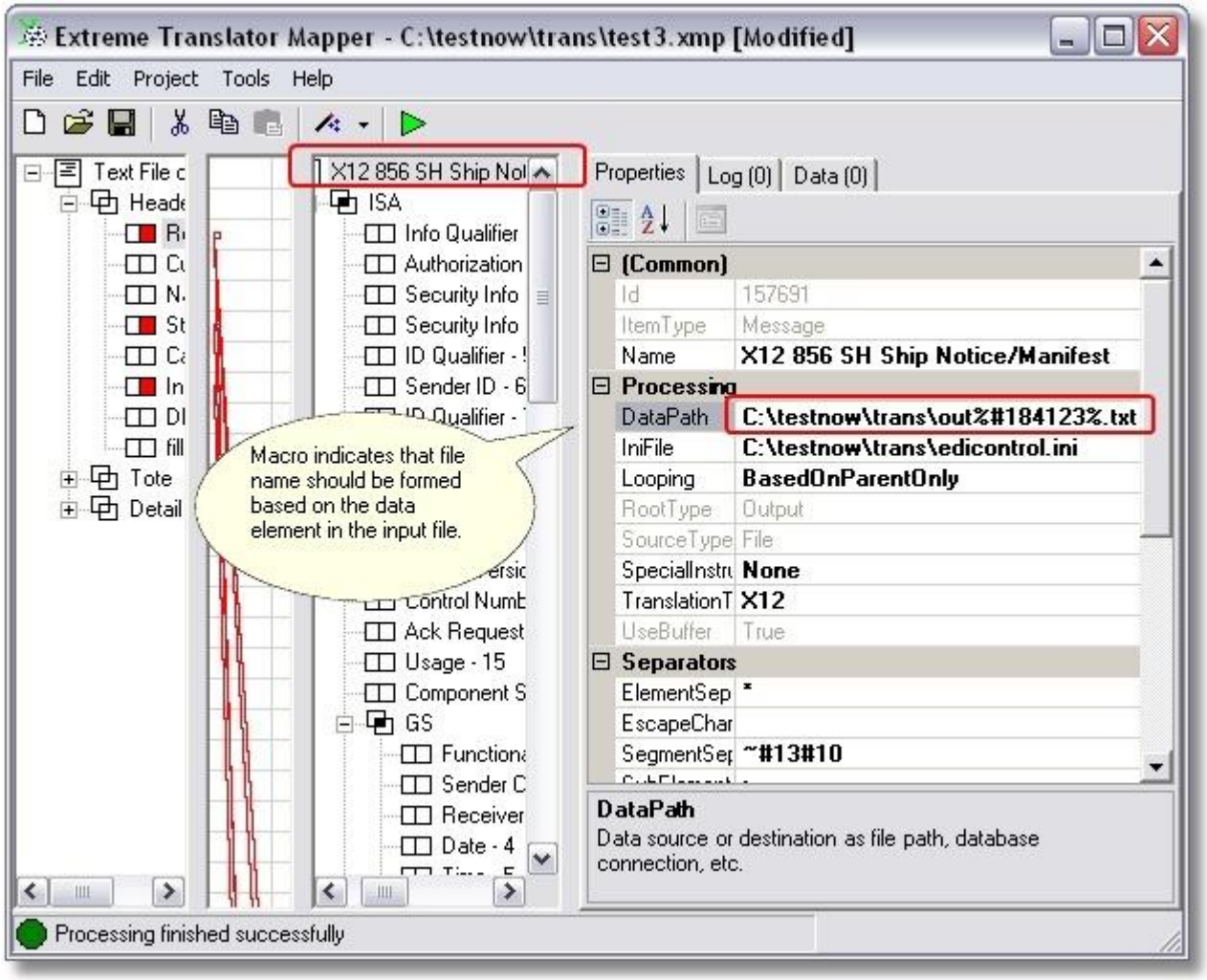
Example of macro %Input% usage.

This will place input file name as output name and append ".txt" extension to it. You can also combine other macros in the same DataPath property such as %Input%_%Count%_%SystemDateTime%.txt or similar.



Example how %#propertyID% macro can be used to form DataPath based on the input or output data. See next screen shot for more details.

“Id” property is unique for each item in the map. Since it identifies each item uniquely it comes in handy in number of different scenarios.



Example how %#propertyID% macro can be used.

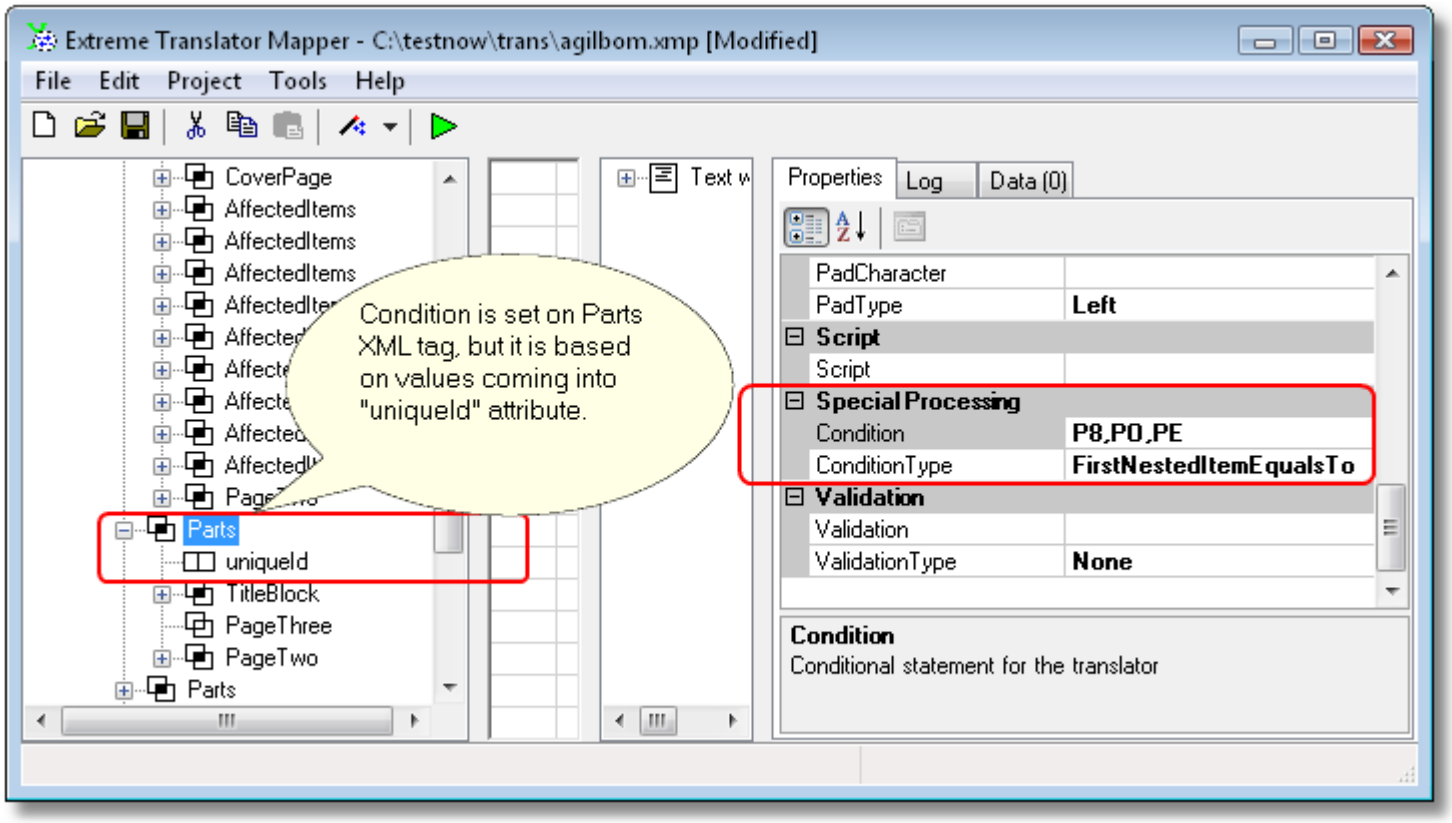
Condition and ConditionType properties

Properties Condition and ConditionType provide a way to filter or route incoming data to separate locations in the output file.

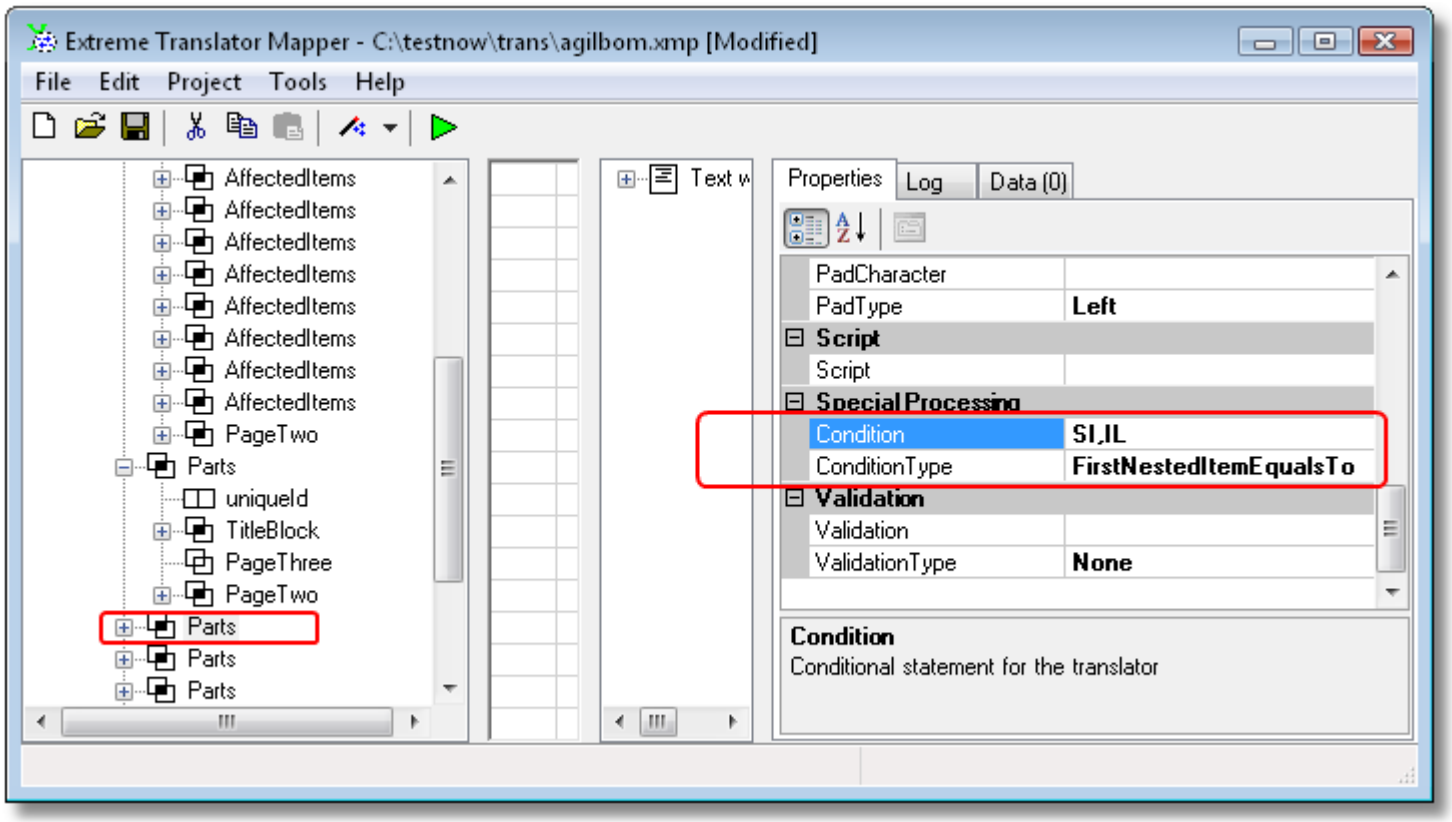
They are especially handy for EDI X12 files that contain same segments separated only by specific qualifiers or identifiers.

Typical example would be NM1 segment in number of EDI X12 files. For example: NM1*IL holds subscriber data (here “IL” is qualifier) while NM1*QC holds patient data (“QC” is qualifier).
Use Condition/ConditionType to separate each NM1 on the input side and then map them to different fields on the output side. Copy & paste each NM1 and change Condition/ConditionType to FirstNestedItem=IL or FirstNestedItem=QC.

In this example below only certain XML data with attribute “Qualifier” equal SI and IL should be placed into the output, and all XML tags that have “Qualifier” equal P8, PO and EQ should be filtered and will not make into the output.



In this example MiscRecord with P8, PO and EQ values is filtered and not placed into the output.



Only MiscRecord's with SI and IL are placed in the output.

Note: Conditions work only on EDI and XML input side messages. Condition/ConditionType should be set on segment and performs filtering based on value(s) inside segment's element. Let say you have XML message then

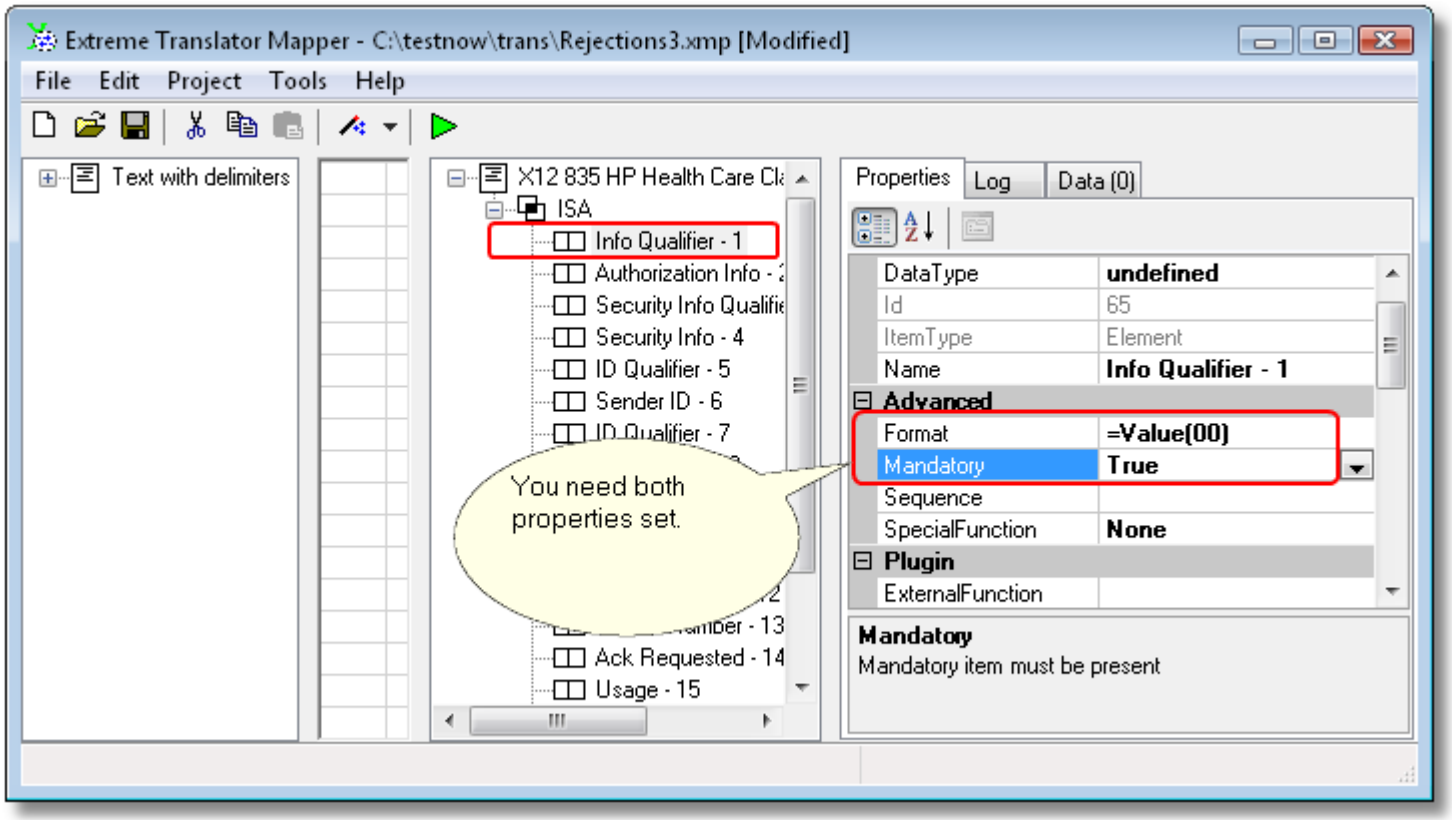
Condition/ConditionType could be set on the tag (segment) and filter based on tag's XML attributes. Cannot filter tag based on other tags nested below.

Format property

It is advanced property that allows special formatting to be applied to data during translation. It is powerful feature. There are number of types of expressions you can use in this property:

Function	Description
=Value()	Place constant value into the element or field.
=ValueIfNull()	Place constant value if data for that element or field is NULL (no data).
=ValueIfNotNull()	Place constant value if data for that element or field is not NULL (has data).
=Form()	Reformat data.
=Match()	Match using Regular Expressions.
=Replace(;)	Replace using Regular Expressions.
=Substitute()	Substitute certain values based on the list provided.
=Evaluate()	Perform simple arithmetical operations on data, append or add additional character data.

Value is used to place default value in the map item during processing.
Example of use: =Value(defaultvalue).
ValueIfNull is used to place default value in the map only if incoming data is NULL or empty string.
ValueIfNotNull is opposite of ValueIfNull.



Default value placement in output.

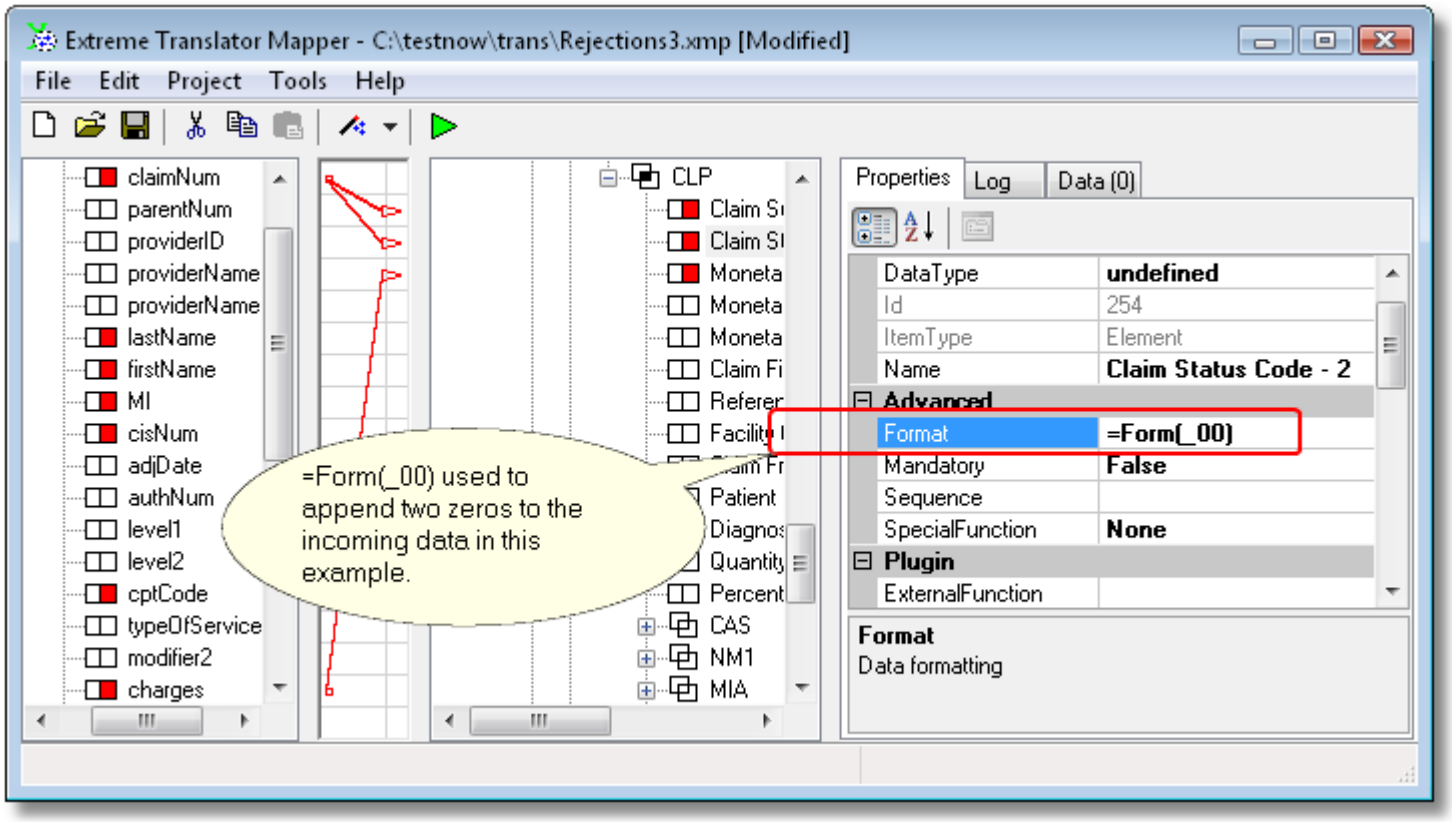
Form is used to form output data based on input.
You may use special escape characters to manipulate data. Escape characters are: "@" and "_".

Usage:
@Position number - may look like this "00@21" would print "00" + character at position 21 in the data. Position number count starts from 1, not 0.
_ - will place all data in this location, so "00_" would print "00" + all the data

Example 1:
You have data in format YYYYMMDD coming from EDI X12 file. You want to convert data into format MM/DD/YYYY. In order to do that you can setup *Format* property like this:
=Form(@5@6/@7@8/@1@2@3@4)

This basically says: take character at position 4 from input and place in position 0, then take character at position 5 and place at position 1, place "/", take character 6 and place at position 4, etc. If you need to add some constant data to the output of some element, simply use "_".

Example 2:
You want to add four zeros in at the end of data. Your *Format* would look:
=Form(_0000)



Example of "=Form()" usage.

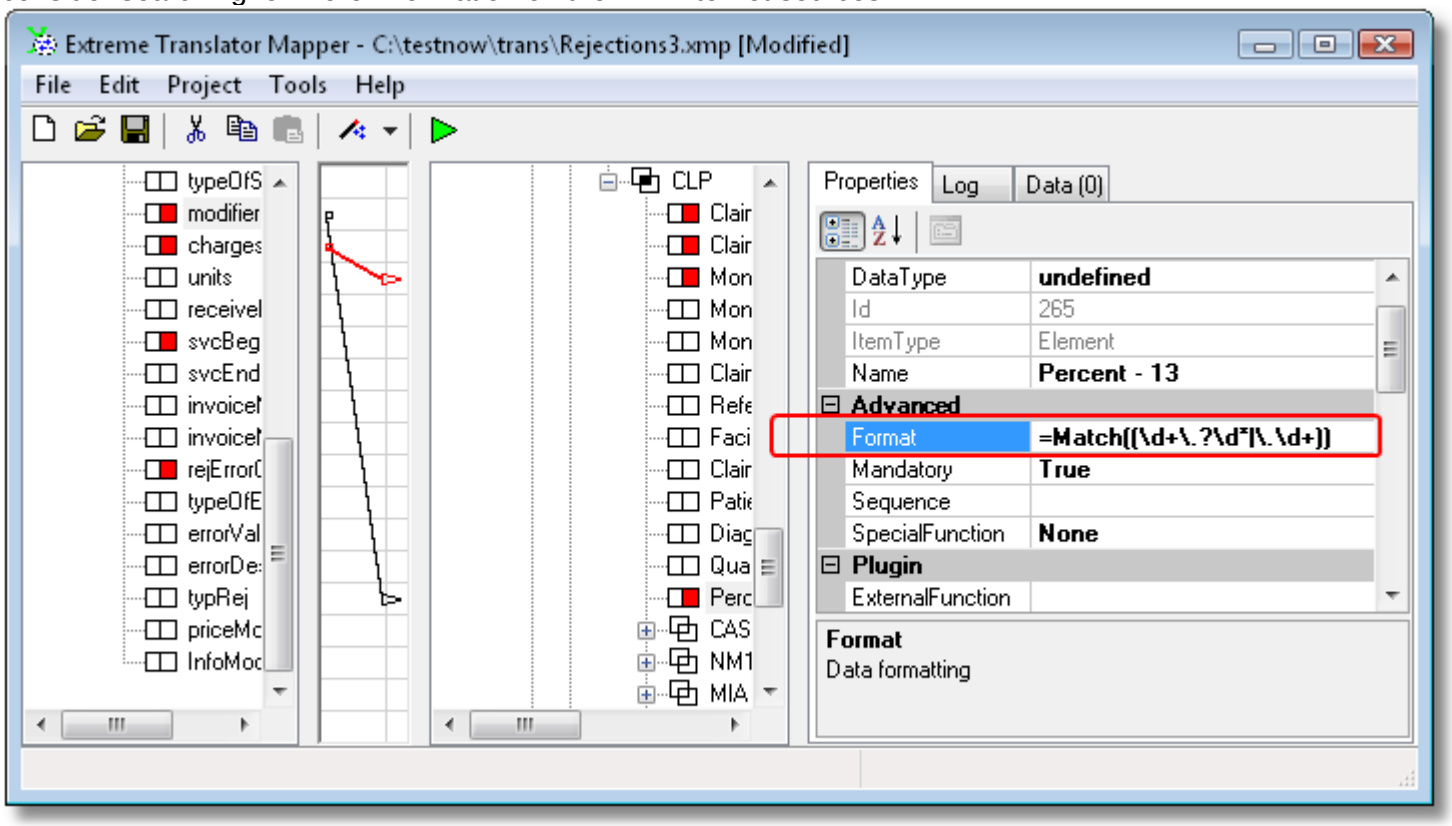
Match and Replace is based on regular expressions processing used in Perl, awk and many other utilities and languages. There are books written on how to write powerful regular expressions. Translator supports a subset of all available regular expressions.
Those are some of examples on how to use "=Match()" and "=Replace(;)".

If input is "abracadabra" then "=Match((a|b|r)+)" would give us "abra" which is the amount of string that has been successfully matched.
If input is "abracadabra" then "=Replace(zzzz;abra)" would give us "zzzzcadzzzz", in which all occurrences of the matching pattern are replaced by the replacement string "zzzz".
Note: semicolon is used to separate replacement string "zzzz" and actual regular expression "abra" in previous

“=Replace”.
If semicolon has to be in your input data it can be escaped with backslash character. Example: =REPLACE(&APOS;;') would replace &APOS with ;’ but your desired result is to replace &APOS; with ‘ (quote), so your Format property should be =REPLACE(&APOS\;;').

Examples of some regular expressions:
”=Replace(;^\s+)” remove leading whitespace
”=Replace(;\s+$)” remove trailing whitespace
”=Replace(;\s*\r?\n\s*)” joining lines in multilane strings (or removing carriage return and line feed)
”=Match((\d+\.?\d*|\.\d+))” extract all numbers from string

As you can see regular expressions can be very powerful however somewhat cryptic to read and use. You may consider searching for more information on them in Internet sources.



Example on using “=Match()” regular expression.

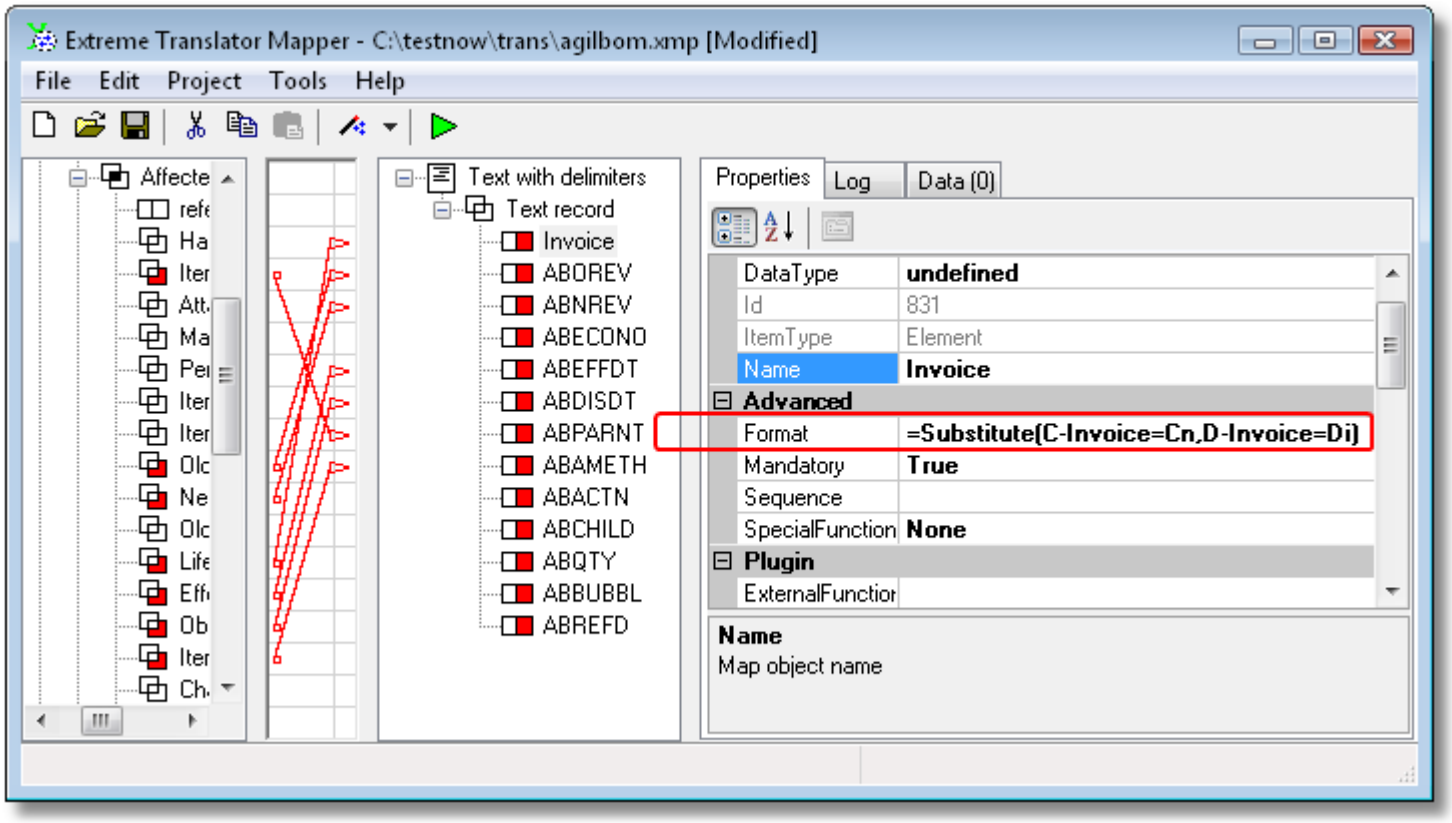
The regular expression language includes two basic character types: literal (normal) test characters and metacharacters. Regular expression metacharacters are an evolved extension of the ? and * metacharacters used with the MS-DOS file system to represent any single character or group of characters.

This is a short overview of common regular expressions metacharacters:

Expression	*Meaning*
.	Matches any character except \n
[characters]	Matches a single character in the list
[^characters]	Matches a single character not in the list
[charX-charY]	Matches a single character in the specified range
\w	Matches a word character; same as [a-zA-Z_0-9]
\W	Matches a nonword character
\s	Matches a whitespace character; same as [\n\r\t\f]
\S	Matches a nonwhitespace character
\d	Matches a decimal digit, same as [0-9]

\D	Matches a nondigit character
^	Beginning of the line
$	End of the line
\b	On a word boundary
\B	Not on a word boundary
*	Zero or more matches
+	One or more matches
?	Zero or one matches
{n}	Exactly n matches
{n,}	At least n matches
{n,m}	At least n but no more than m matches
()	Capture matched substring
(?<name>)	Capture matched substring into group name
|	Logical OR

Substitute can be used to replace specific input values with predefined output values. Property format is =Substitute(oldvalue1=newvalue1,oldvalue2=newvalue2,*=newvalue3)
Special character ”*” star means any other value will be newvalue3.



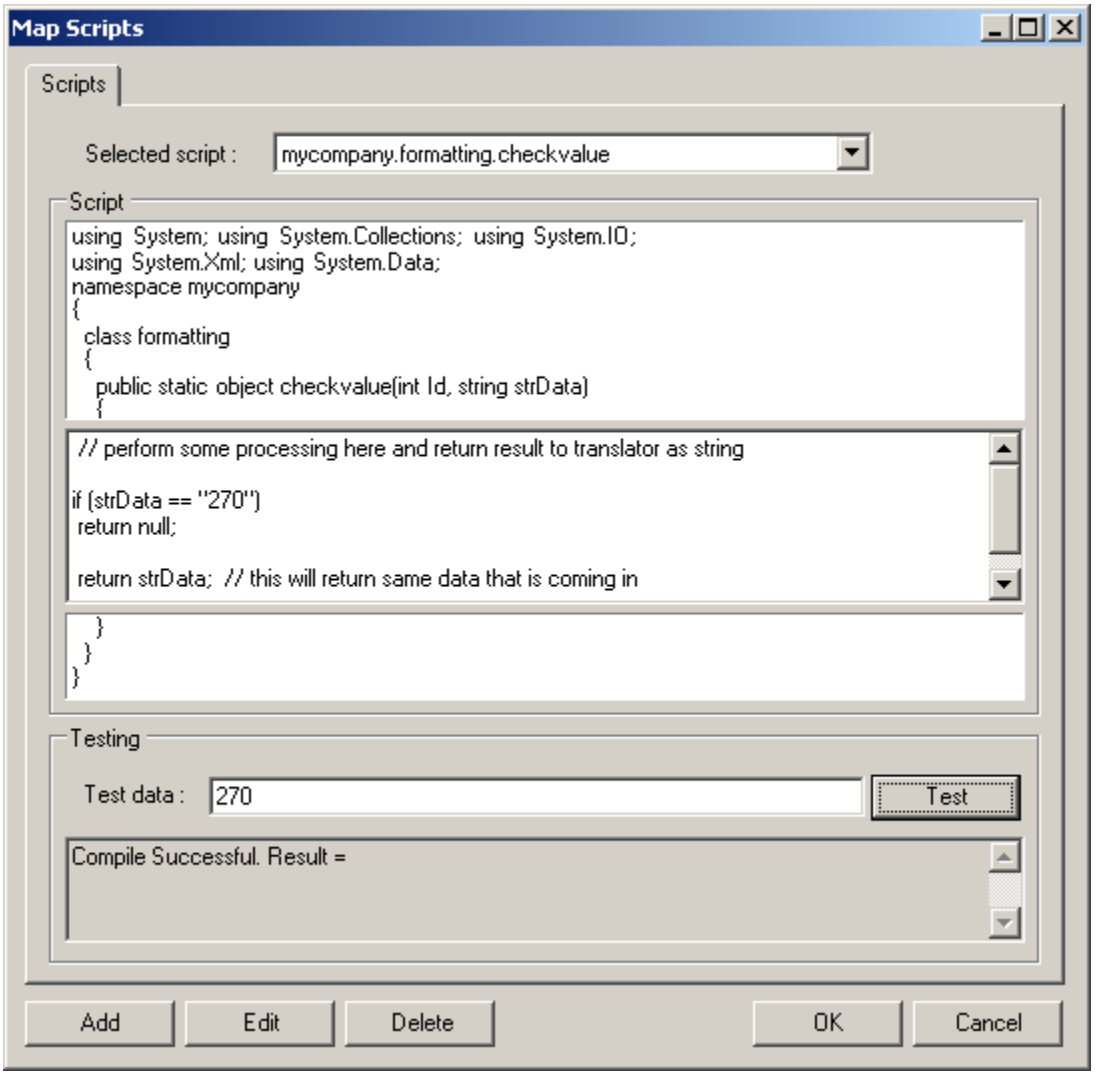
In this example if input is "C-Invoice" it will be replace to "Cn", but if it is "D-Invoice" it will be "Di" on output.

Evaluate can be used to perform simple arithmetical operations or help to concatenate character data. Special character @ can be used to indicate incoming data.
Example 1:
=Evaluate(10+@) will add 10 to incoming data. So if data that is coming to element or field is for example equal to 85 then result after evaluation will be 95.
Example 2:
=Evaluate(4 + @ + ‘AAA’) will add 4 to incoming data and append AAA to the end result. So if incoming data is 34 then end result would be 38AAA.
Example 3:

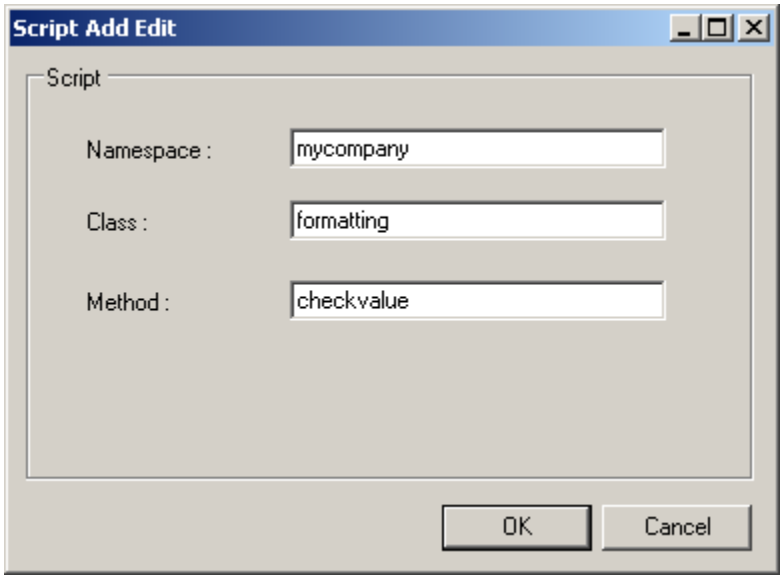
=Evaluate('ABC' + @ + 'DEF') will append ABC characters to the front of incoming data and append DEF characters to the end.
Example 4:
=Evaluate(@ / 10) will divide incoming number by 10.

Script property

If Format property cannot format data the way you want, you can use full power of C#.NET language to do extra processing using Script property. Add scripts via "Scripts" menu in Map Editor, then click on specific item that has Script property and assign script from the list.



You can use Map Scripts dialog to add and test new scripts. If you click Test button script will be compiled and executed with data from "Test data" edit box assigned to strData variable.



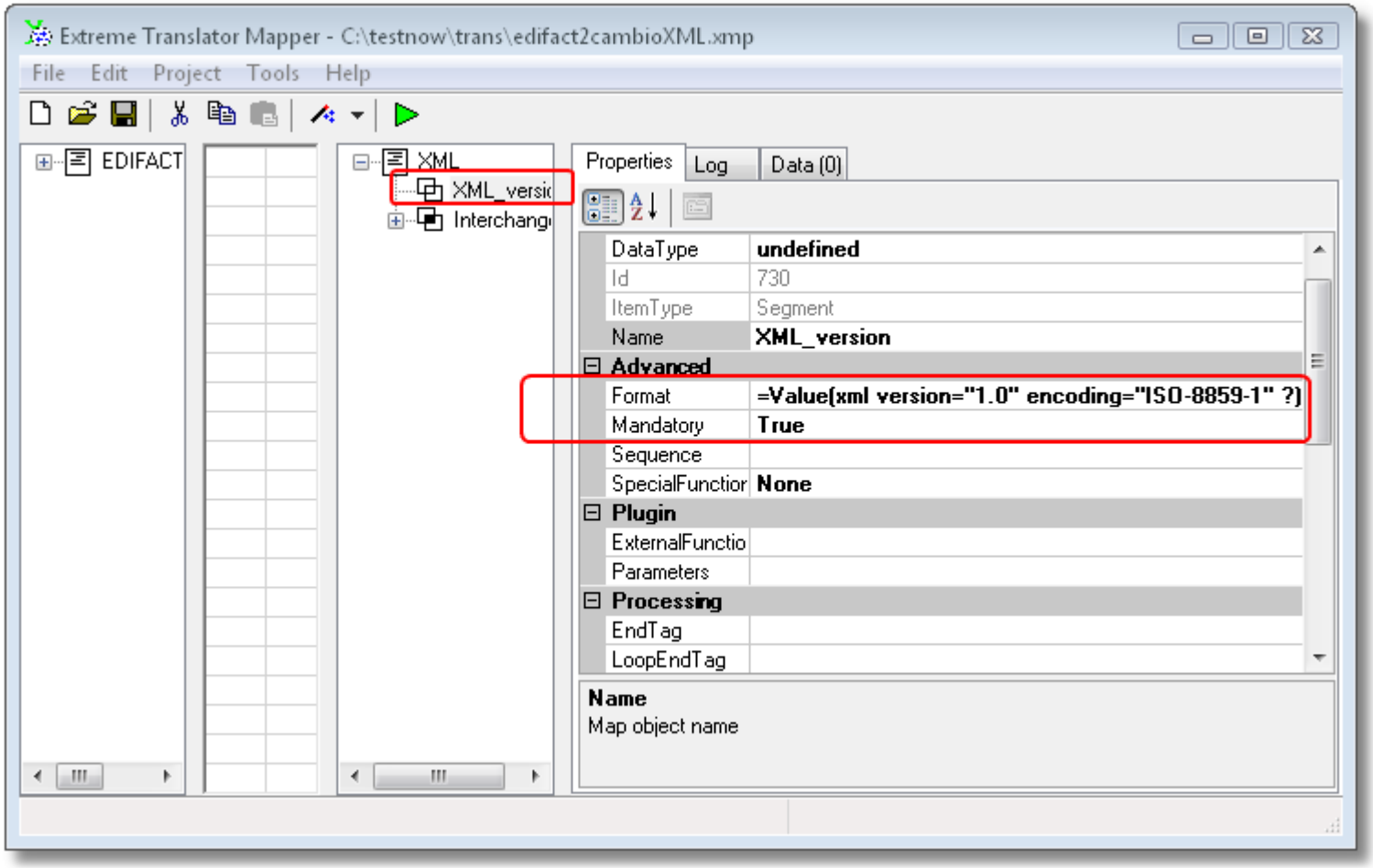
All scripts have to contain namespace, class and method names. For simplicity you can use your company name as namespace. Class and method names should be different.

XML Translation

List of most important properties for XML maps

StartTag	XML tag or XML attribute name. Do not use “<” and “>” brackets in this property
EndTag, LoopEndTag	Not used
ValidationType, Validation	If ValidationType is set to SchemaFile then Validation should point to actual XML Schema file

Translator uses DOM parser included in .NET Framework.
Warning: XML file should have proper valid XML format including first XML item, example:
<?xml version="1.0"?>.
If this item is missing By-Example Wizard would not be able to setup map items based on XML file provided.



Screen shot on how to setup XML attribute for the output.

Segments as well as elements can be mapped to output. Segment might look like:

<segment_name>some data</segment_name>

Elements are inside of segments and look like:

<segment_name elem1="some element data">
some segment data
</segment_name>

When you create mappings to produce XML, certain characters will be escaped in order to produce valid XML output. There is a table of escaped characters:

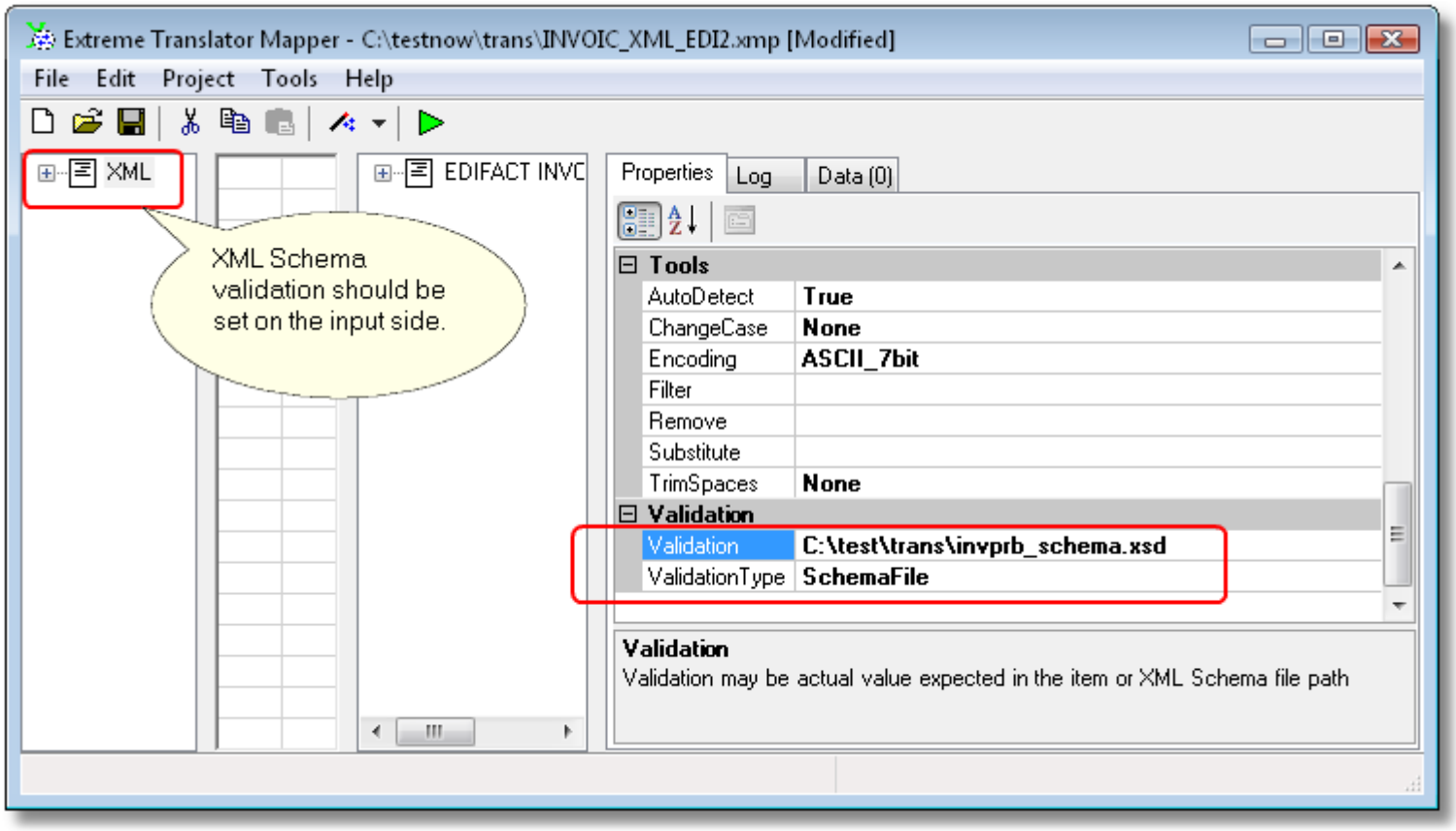
Input character	*Output string*
‘ (single quote)	'
“ (double quote)	"
> (greater than sign)	>
< (less than sign)	<
& (ampersand)	&

XML Schema Validation

Validation against XML Schema can be performed using XML Schema file. Validation and ValidationType properties

have to be setup to perform the validation.

By-Example Wizard can be used to create the map for you. After you run the wizard you have to click on the root item and fill *DataPath* and *Validation*, *ValidationType* properties.



Example of setting XML validation on the incoming XML file using XML Schema.

Flat Text File Translation

List of most important properties for delimited variable length text file processing maps

StartTag	Block of text or single character that marks the start of the item. You can think of it as an opening bracket in the text data. Most of the time it is delimiter, such as comma, new line character, space character, etc.
EndTag, LoopEndTag	Block of text that marks the end of the item. You can think of it as closing bracket. Translator will extract data between StartTag and EndTag/LoopEndTag.
SpecialInstruction	It can be setup to FlatOutput or FlatOutputInline on output element to make data in the output line up correctly. Please read to the end of this chapter for illustration.

List of most important properties for fixed length text file processing maps

FixedLength, Length	If FixedLength is true the Length of actual data item to extract
PadCharacter, PadType	Actual character to be used if data should be padded on the Left or Right
Format	Can be used to produce desired output in a special format

Text files can be of any structure: flat files, fixed length files, files exported from Excel or MS Access. Any file that has text structure can be converted.

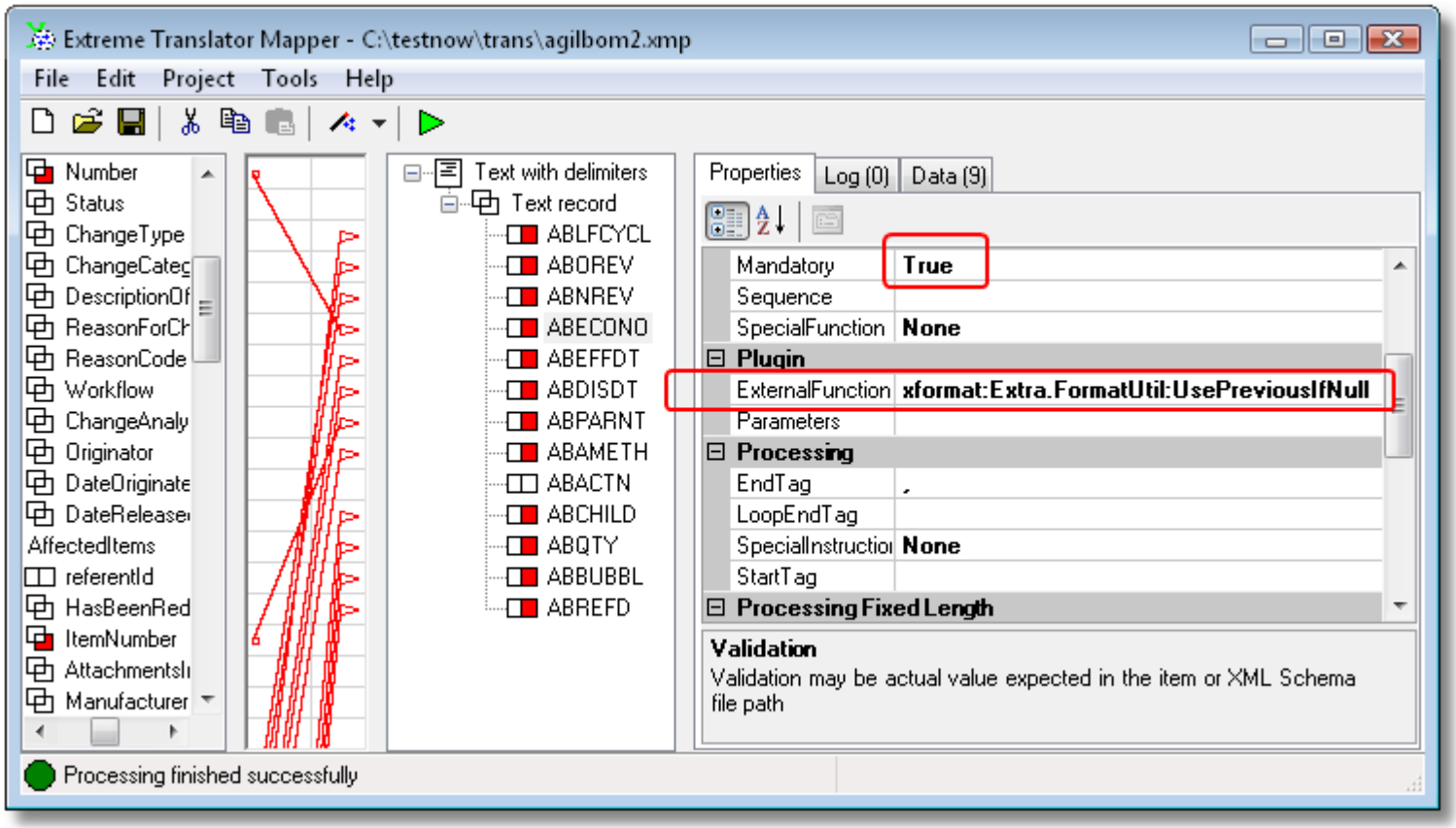
There are two most important properties you have to setup in the map in order to process delimited text files. Those properties are called “StartTag” and “EndTag/LoopEndTag”. You can think about the two properties as open and closing brackets. Just imagine brackets in your text file you want to process and based on this create segments and elements.

”StartTag” is the block of text translator will compare incoming data against. If data matches then translator will try to find “EndTag/LoopEndTag” and if it can find the end, it will step one level down into the map and try to locate nested items.

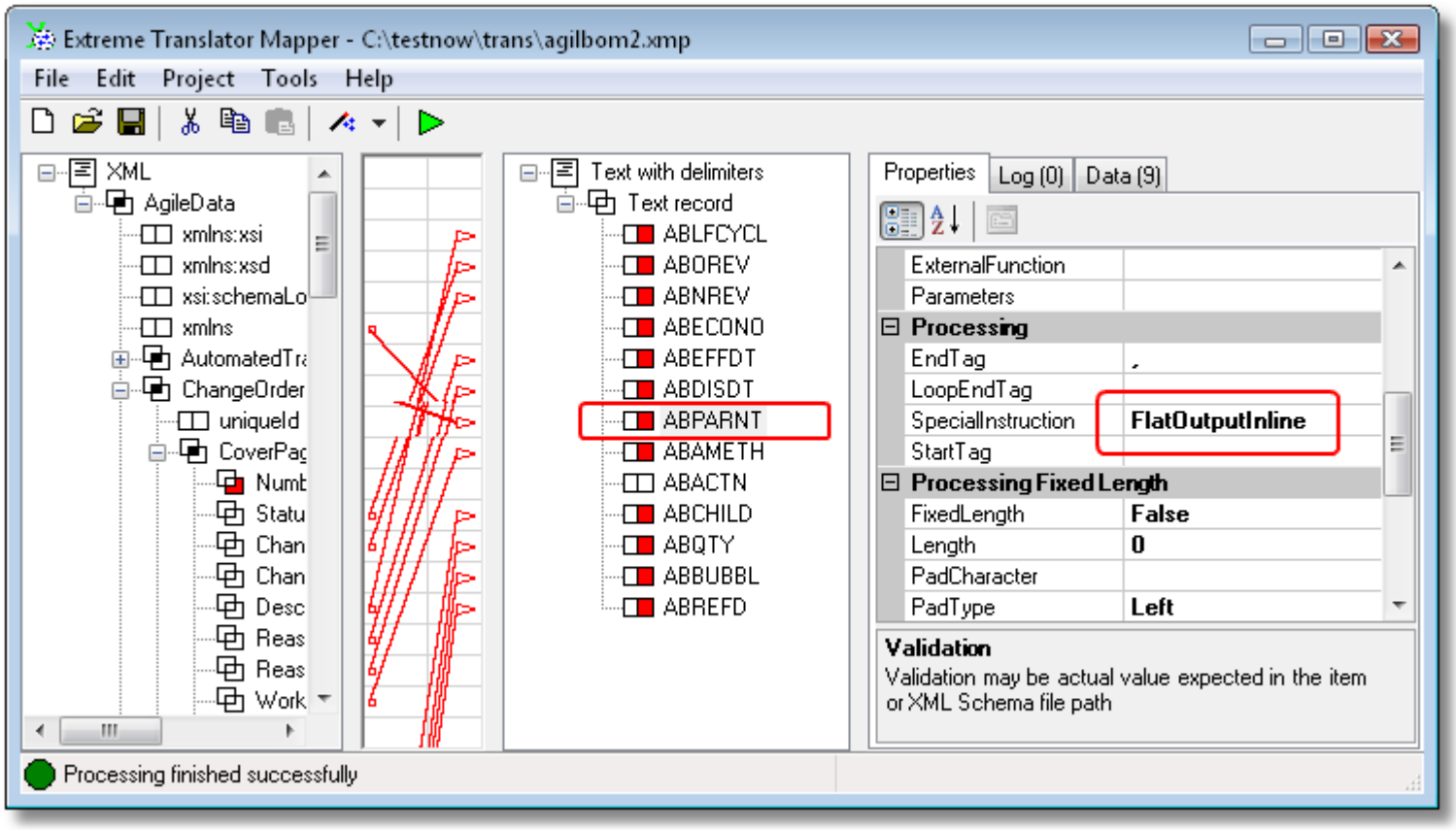
If you are trying to process fixed length files then you should use FixedLength, Length, PadCharacter and PadType properties.

It is possible to intermix text file processing and have both fixed length and delimited text processing techniques in one map.

Mapping XML, EDI X12 and EDIFACT formats to flat files can be a challenge just because they are nested and/or looping. Most of flat files contain some header and detail elements that are separated by commas or presented in fixed length. Header elements have to repeat per each detail line.



There is example of XML mapping to flat text file with comma delimiters. In this example header data has to repeat per each line of detail, so UsePreviousIfNull plug-in function is used to fill in empty spots.



Also to have all the headers line up with details SpecialInstruction property has to be setup to FlatOutputInline on first detail item in the flat file.

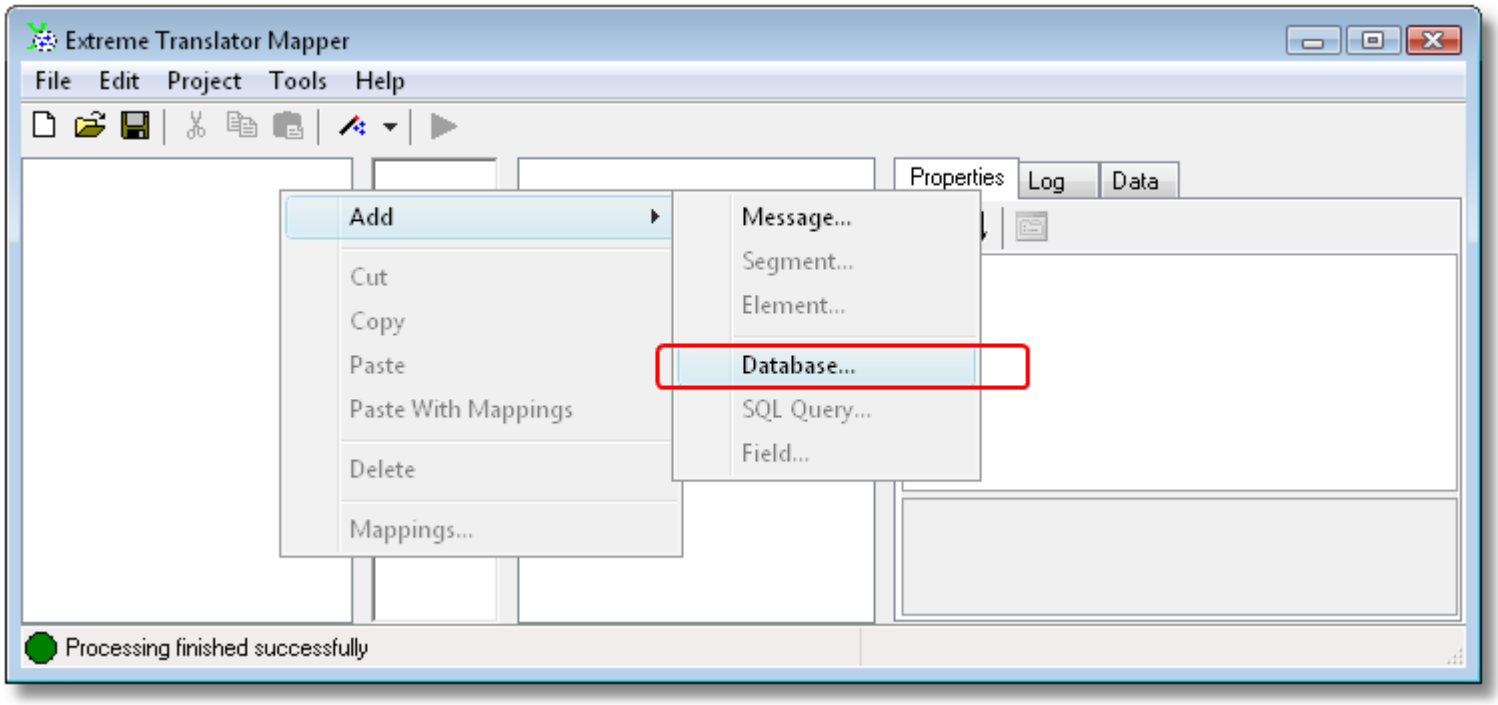
Database Mappings

Mapping

Translator can import or export data from database. ODBC connection is established to read data using SQL statements or write data into database tables. You have to define database objects in the map in order to process them in translator. Root item database object can be defined using popup menu “Add” and “Database”.

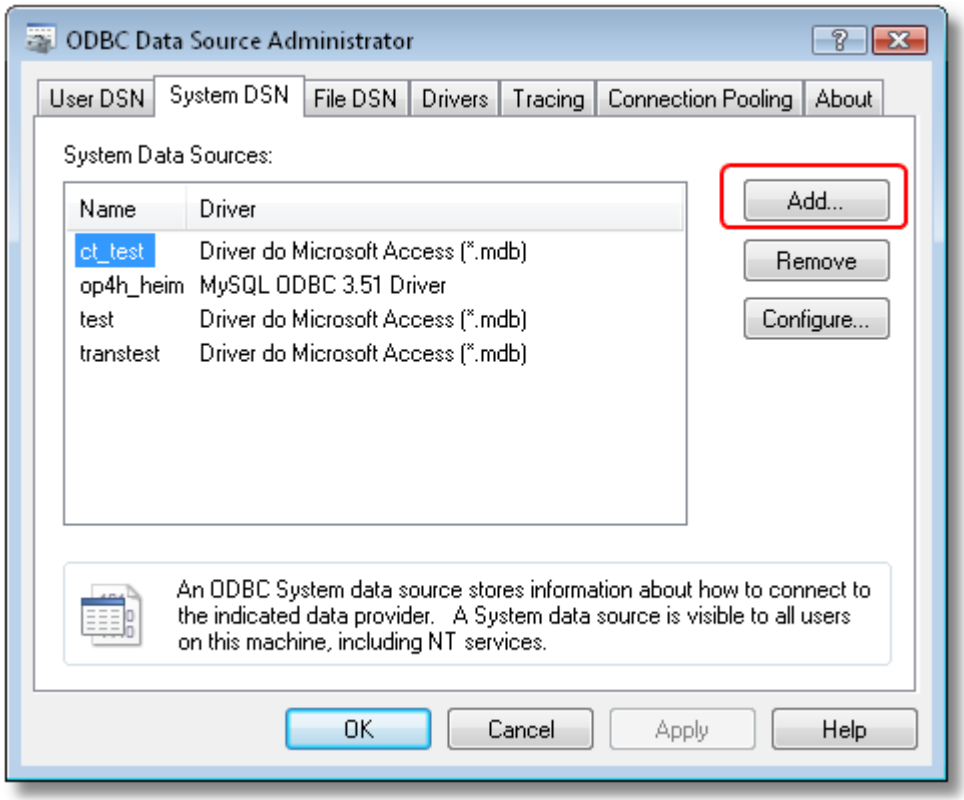
If you have Windows 64bit OS then make sure ODBC driver is also 64bit driver. Translator cannot work with 32bit ODBC drivers on Windows OS 64bit.

If you try to use 32bit ODBC driver you might be able to add it to Control Panel but driver may not work from the application.



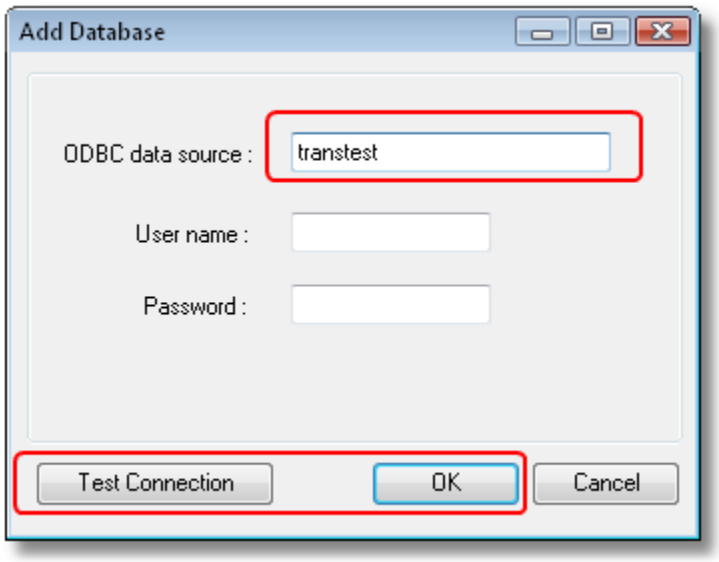
Adding database to the map.

Add Database dialog contains connection string information: ODBC data source name, user name and password. ODBC data source should be defined in Control Panel under Data Sources (ODBC).



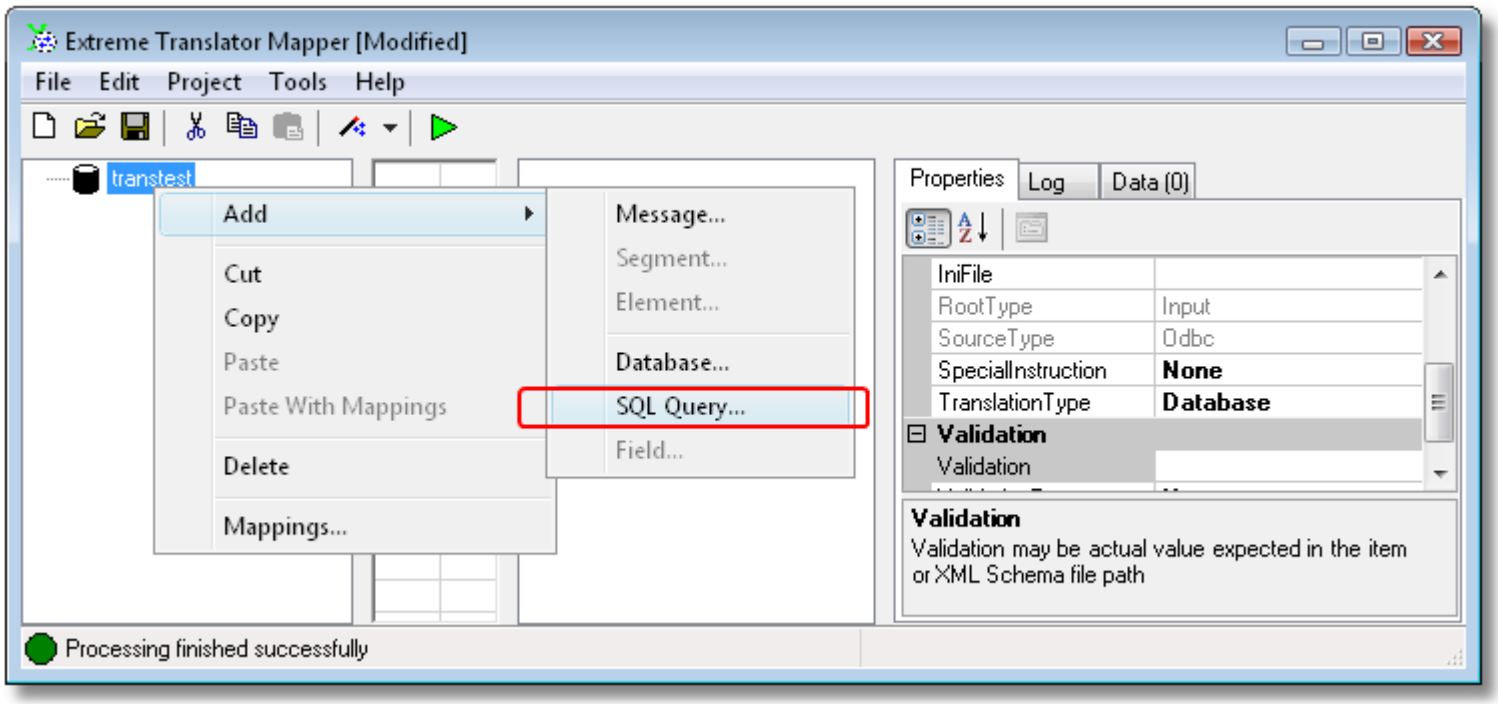
Windows ODBC data source manager.

Sometimes databases do not require user name and password in that case leave it blank. You can use “Test Connection” button to make sure connection works. After you press “OK”, connection information is used to populate “DataPath” property. Modify “DataPath” property if data source name, user name or password changes.



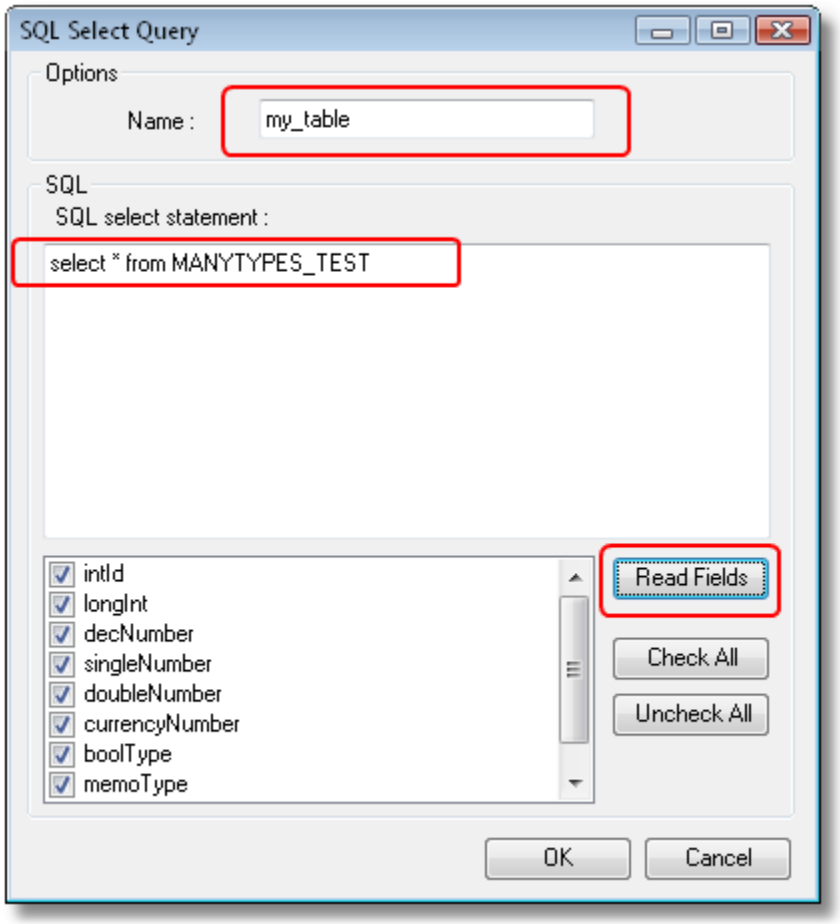
Database connection settings.

When database is setup, you can add SQL queries to it. Queries you can setup on the left side (input) are different than ones you would setup on the right (output). Input queries are SQL SELECT based queries for data retrieval. Output queries are for SQL INSERT statements. Major difference is that you can edit SQL statement of SELECT query and modify it, when INSERT statement is dynamically constructed based on the fields listed under the query.



Adding SQL queries to database.

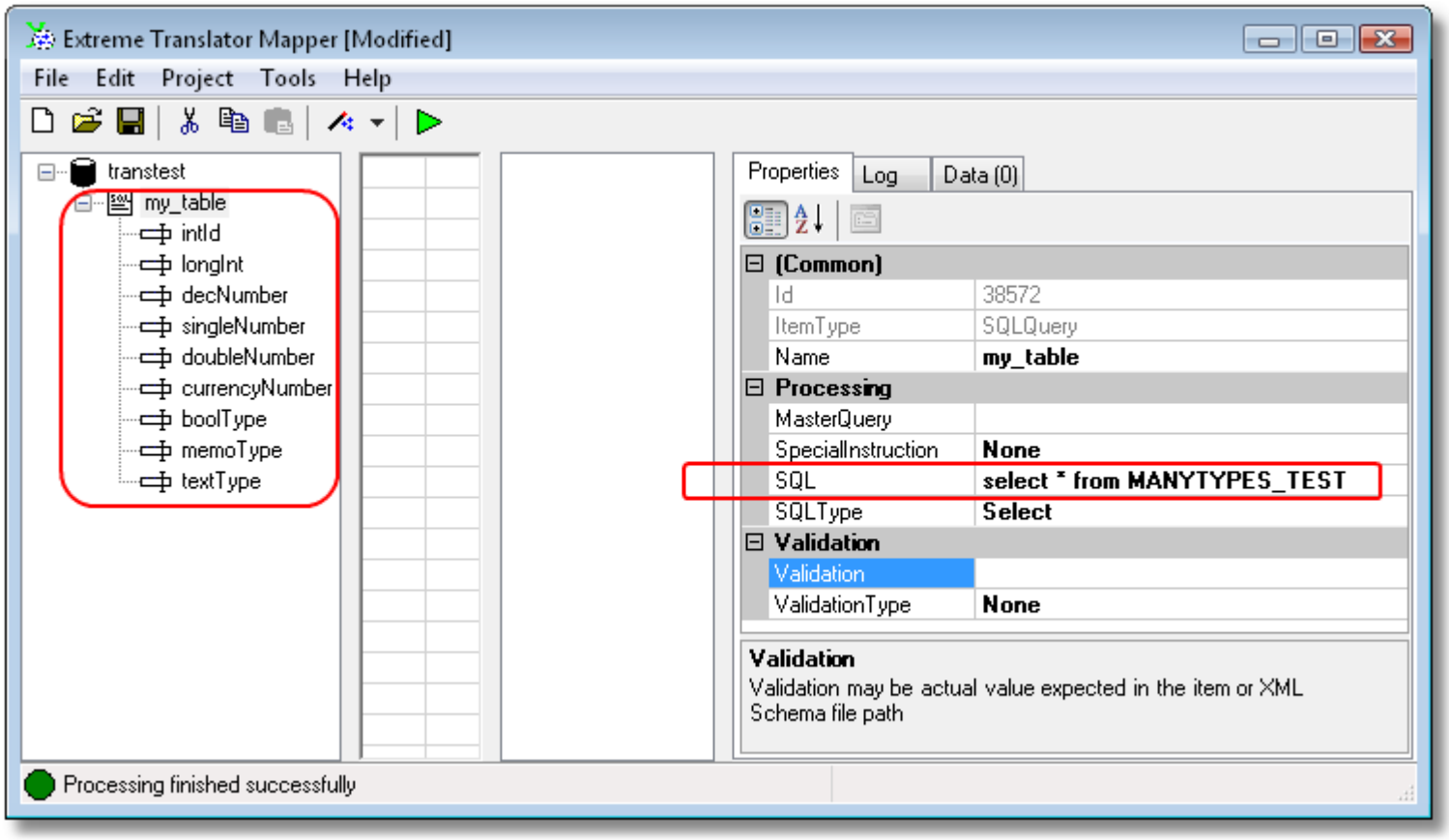
SQL queries during processing are executed in the order they are listed on the screen. Type query name and SQL statement in SQL Select Query dialog, then press “Read Fields” to get all the fields query can retrieve and press “OK”. Query and checked fields will be added to the map.



Adding query and fields to the map.

Map editor will try to detect queries field types and sizes. Some specific field types may not be detected properly and may result in data truncation or failure during data retrieval. SQL can be edited after query is added. If new fields are added to SQL statement (SQL property) and should be retrieved, you should add them under the query using “Add” and “Field” menu option, and map them to output.

Each additional map query will use resources and take extra time to run. Therefore for faster map execution we recommend having fewer queries with more fields rather than many queries with few fields.



Major query properties.

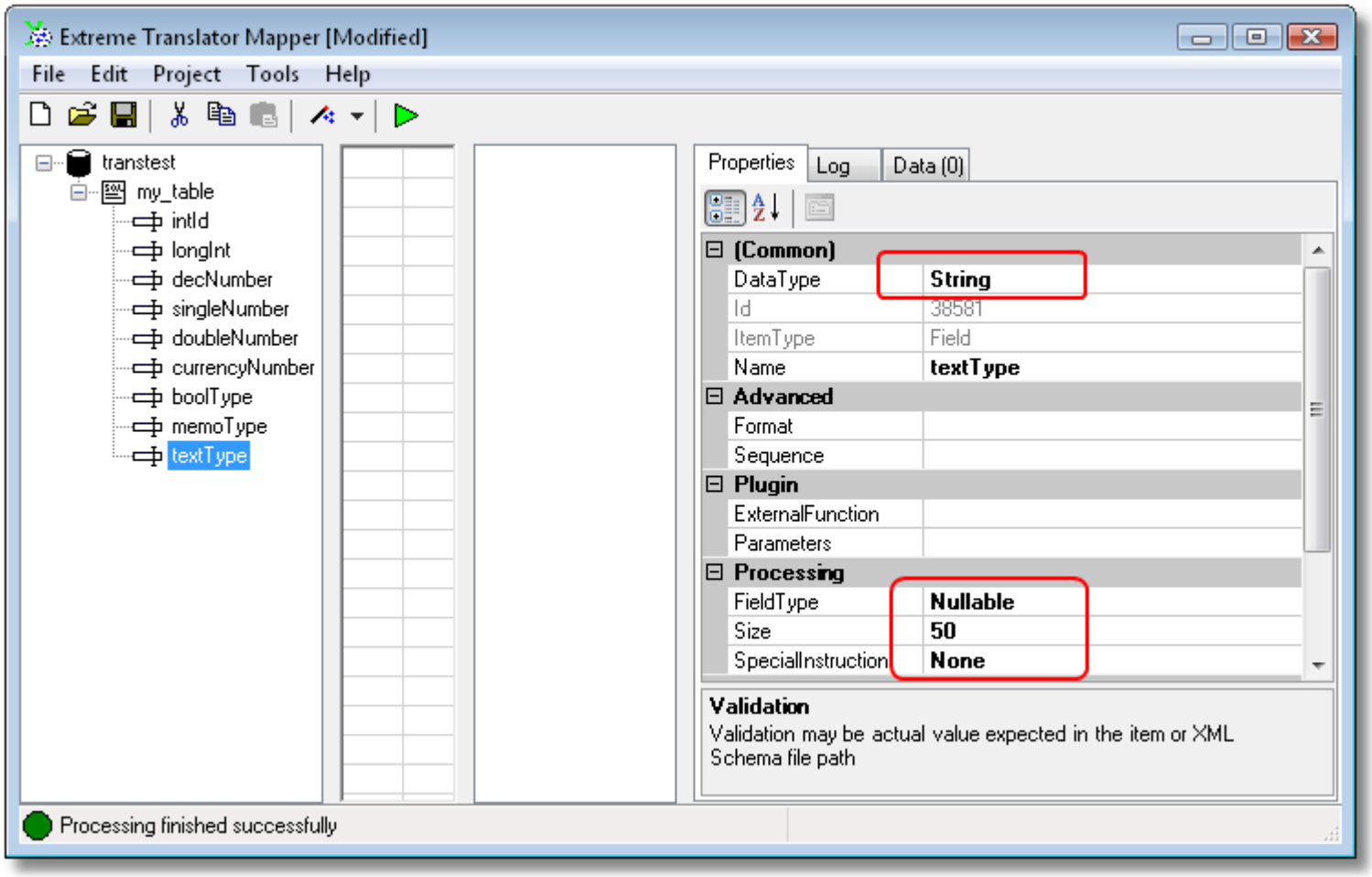
Modify SQL query to select specific rows from the database. If your database tables contain all the data and you want to extract only subset of that data use WHERE clause to select specific rows.

Scenario: you have a table with multiple rows of data and want to process them in batches of fixed number of rows.

One typical trick is to add additional field to an existing table. Let's call it "flag". Make all "flag" values default to 0. Then set the "flag" field to value 1 on the rows you want to extract. Then modify your WHERE clause to select rows where flag=1. After you run the map, reset all flag=1 values to 2 using SQL UPDATE.

So the whole process is like this: before processing rows start with flag=0. Then you set flag=1 on certain rows and run the map with clause WHERE flag=1. After successful map execution you set flag=2 on processed rows. That way rows in the table go though states unprocessed-processing-done.

This is just one example of solving "batching" question.



Major field properties.

Date, time and datetime fields accept data in a special format.

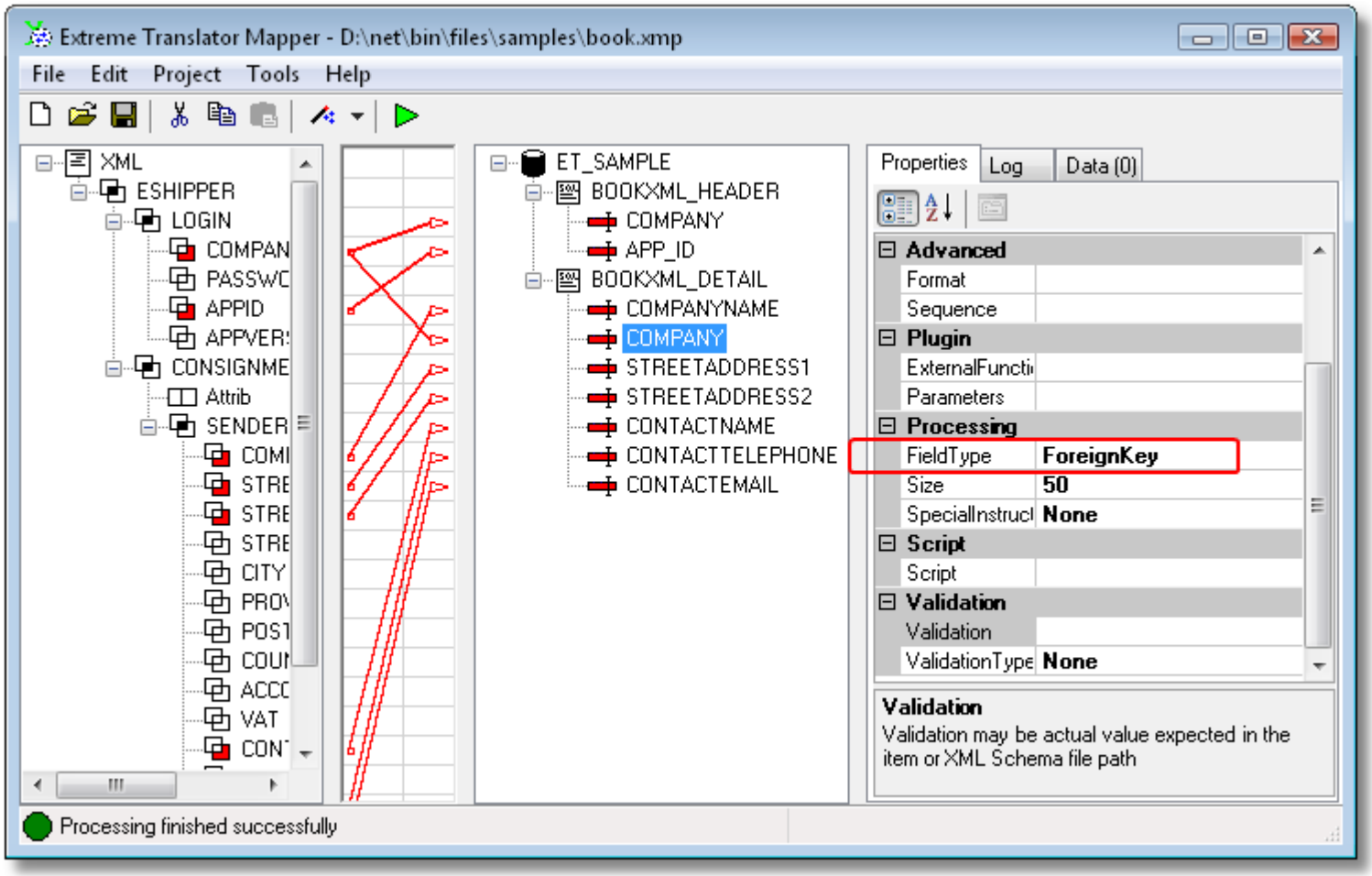
DataType property	Data format to be used
Date	yyyymmdd
Time	hhmmss
Datetime	yyyymmddhhmmss

Inserting data into database

Table has to have at least two fields for inserts to work.
FieldType property drives inserts. If you want to change the way records are inserted modify property for table fields. FieldType should match database field type in most cases. However there could be exceptions if you would like to reorder fields in some special way. You may have FieldType setup to NotNull or PrimaryKey even if actual field in the database table is Nullable.

Basic concepts built into the translator:
1. PrimaryKey field is like a leader field and should lead record ordering. There should be at least one PrimaryKey field in the table. All fields that come in the input stream before PrimaryKey should be marked as ForeignKey or Optional, and all fields that come in the input after PrimaryKey should be marked as NotNull or Nullable.
2. NotNull fields should always have data in them. If there is no data in the input, NotNull fields will “borrow” data from values in previous record.
3. Nullable fields can have they values discarded. If there is only one Key value and two Nullable field values, first value will be lost (overridden) with second value coming into Nullable field. Nullable values should come after PrimaryKey value in the input side.

4. Optional field is like Nullable field with only difference that it's value can come from input data that is above first value that comes to PrimaryKey.
5. ForeignKey is used to mark field that is used to form relationship for PrimaryKey on other table.
Example: there are two tables Header and Detail. Header has PrimaryKey and some Nullable fields. Detail will have one PrimaryKey, some Nullable fields and one ForeignKey. Detail ForeignKey value will come from the same item on input side where Header PrimaryKey data is coming from. Look into XML-to-Database sample for more details. Using these concepts you should be able to achieve almost any record ordering.



FieldType property drives record ordering.

SQLType property for database table can be set to Insert, InsertOrUpdate or Update flag. Translator will use first PrimaryKey field in the database to perform Insert, InsertOrUpdate and Update operations. InsertOrUpdate is a combination of possible operations on the database where translator will select row from the database based on PrimaryKey, and if it exists translator will perform update otherwise insert will be performed.

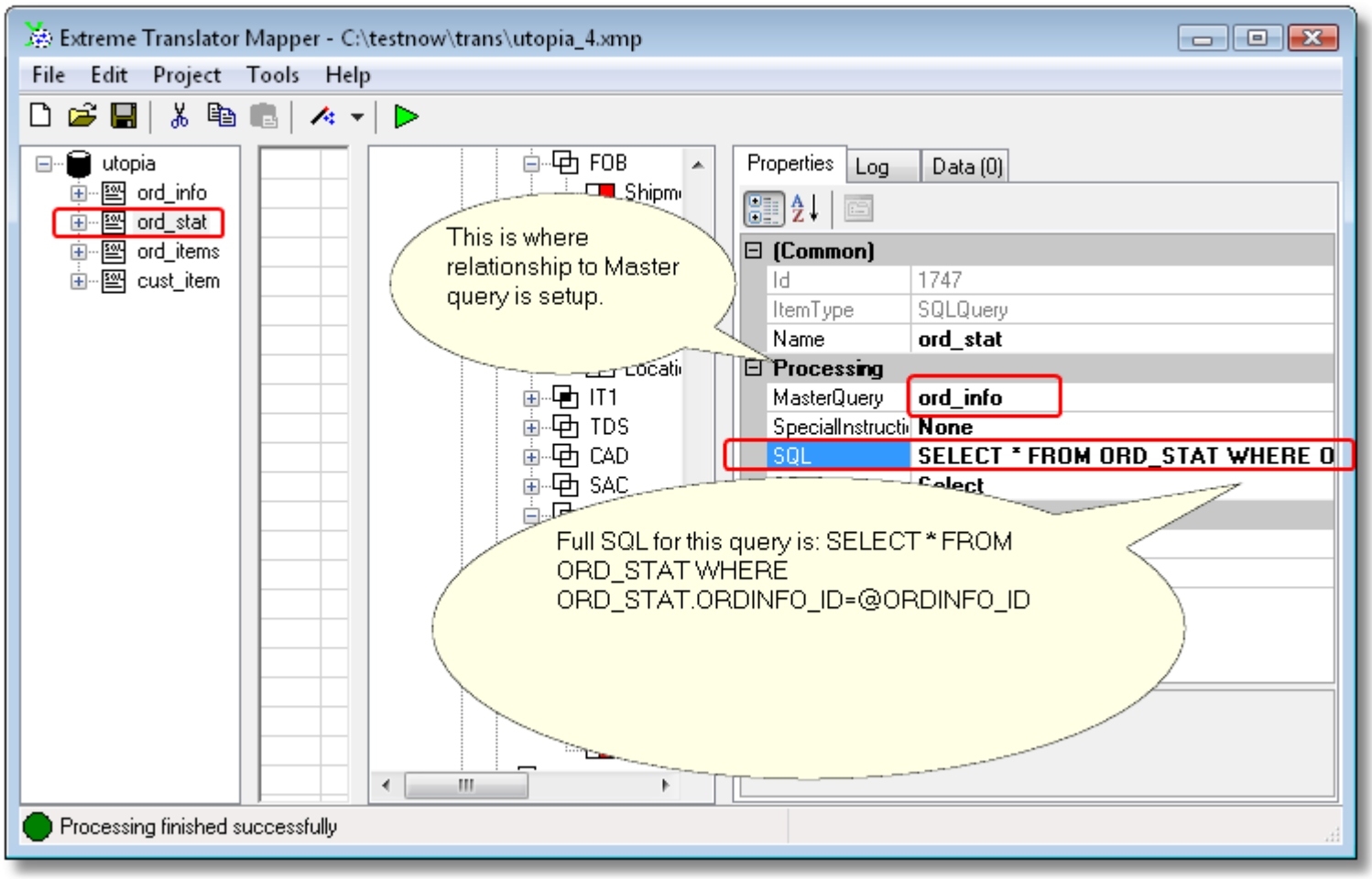
Whenever new value comes into the field that has FieldType=PrimaryKey data gets inserted into the database.
If you do not set any field to FieldType=PrimaryKey on the database table then no values will get inserted into the table.

Selecting data from database

Translator supports master-detail queries. Master-detail queries execute in sequence and retrieve detail rows for each master row.

In simplest scenario you would have two queries, one would provide data for header record, let say purchase order, and another one would provide data for detail record, say purchase order line item.

One query would be something like “*SELECT* PO_ID, PO_AMOUNT *FROM* PR_ORDERS”, second one would be “*SELECT* PO_ID, PO_LINE_NO, PO_ITEM_NAME *FROM* PR_LINE”. Both queries can be tied using master-detail relationship. You can add both queries using “Add SQL Query” menu item and set MasterQuery property on PR_LINE to point to PR_ORDERS, also use field PO_ID to make a relation. Detail query should be modified to “*SELECT* PO_ID, PO_LINE_NO, PO_ITEM_NAME *FROM* PR_LINE PO_ID = @PO_ID”. Symbol “@” means that field from master query should be used to fetch results from this query.
Parameter names and field names are case sensitive. It is possible to nest queries 4 levels deep. That is header, detail1, detail2, trailer with each query passing parameters to the one nested below.



Example of detail query attached to the master. There ord_info is the MasterQuery while ord_stat is the detail. For each record of the master, detail is executed and ORDINFO_ID is passed into it.

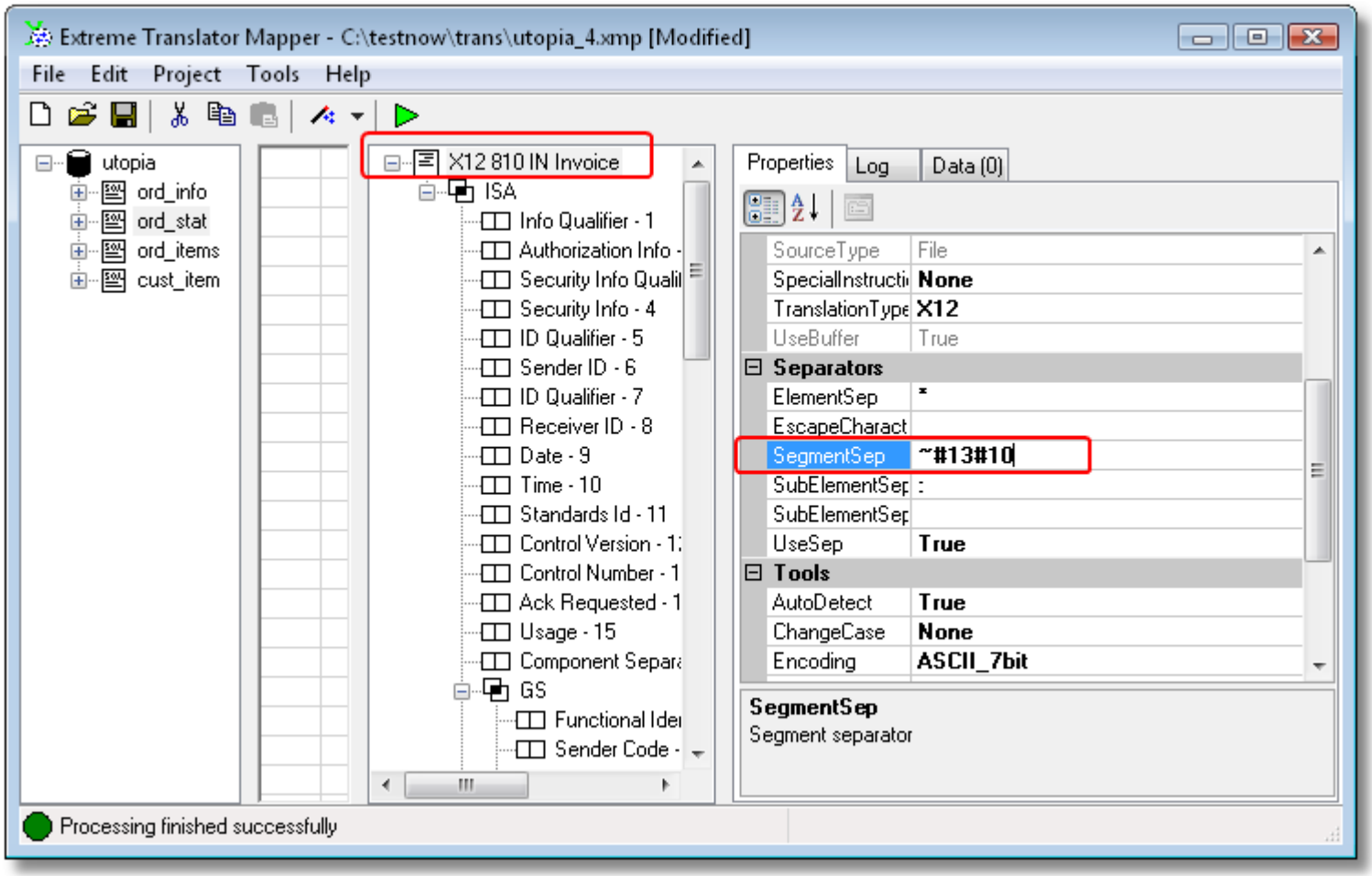
Properties Tab

Properties Tab shows all the properties of the currently selected map item. You can edit any properties that are displayed in black.

Property list can be displayed in groups: Common, Advanced, Separators, etc., or it can be sorted by property name. Some properties allow entering decimal codes in cases when they are not displayable characters and cannot be entered using the keyboard.

One of most common cases is CRLF, carriage return and line feed, character sequence, it is also called new line. You can enter decimal characters #13#10 and they will be treated as CRLF. If your data is separated with just LF you can use #10 only. “#” should be entered before any decimal code.

The properties where you can enter decimal codes are: *StartTag*, *EndTag*, *LoopEndTag*, *SegmentSep*, *ElementSep*, *SubElementSep, EscapeCharacter, Filter*.

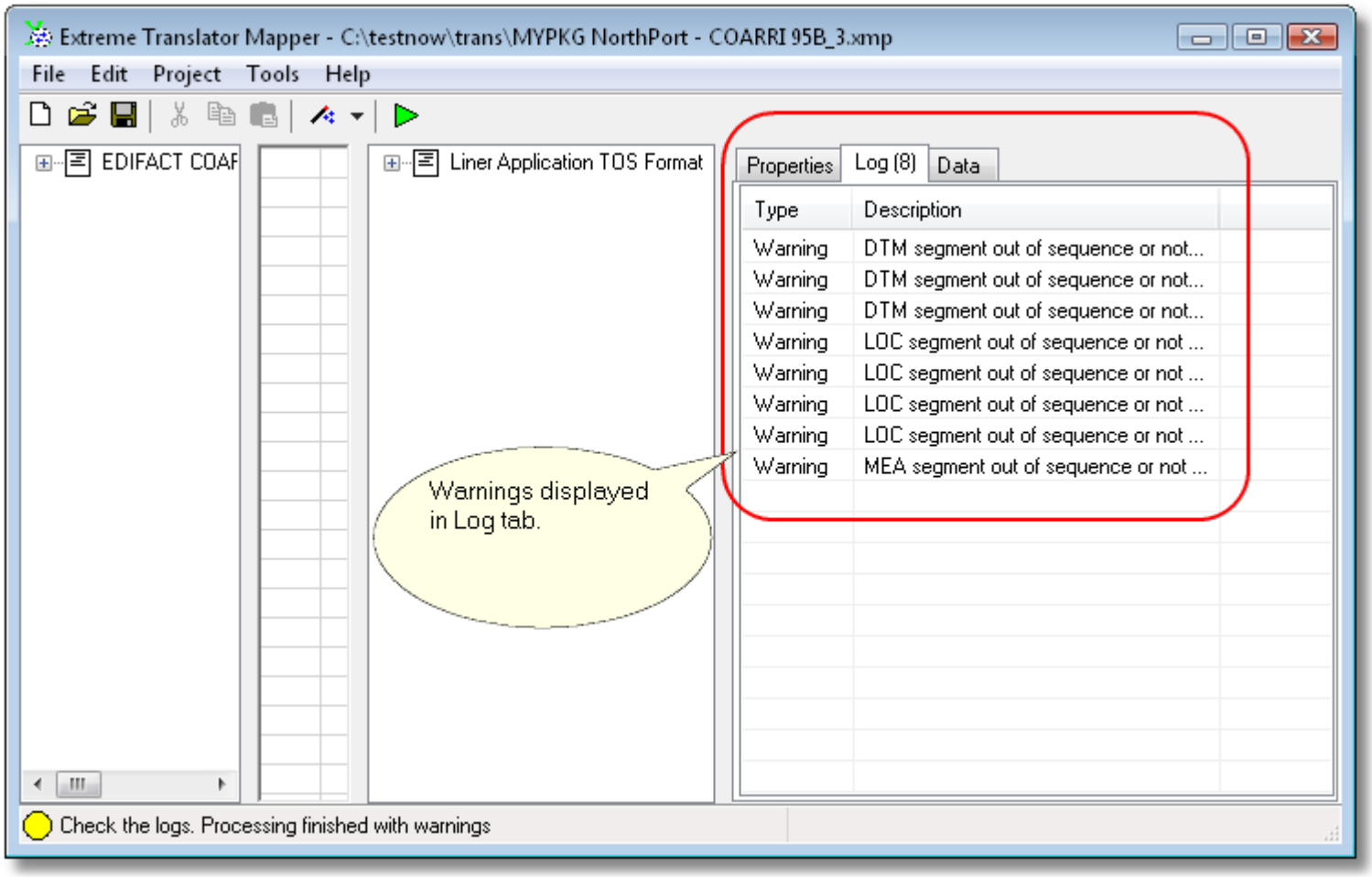


There is how to use non printable characters in properties. In this example carriage return and line feed is placed in the output file.

Log Tab

Log Tab may display warnings and non-critical errors in a list format. That means processing has finished with warnings but output might be produced anyway. Most warnings are data validation errors:

1. Map contains segments with Condition/ConditionType set and incoming segments do not match the Conditions set. All unmatched segments will result in warnings. If unmatched segments also start loop then nested segments will be in the list of warnings.
2. Map segments at specific loop level do not match actual incoming data segments. Unmatched segments will result in warnings.
3. Incoming data segments do not match map at all. Typical examples are when map has been executed on wrong input files destined to be processed by other map.



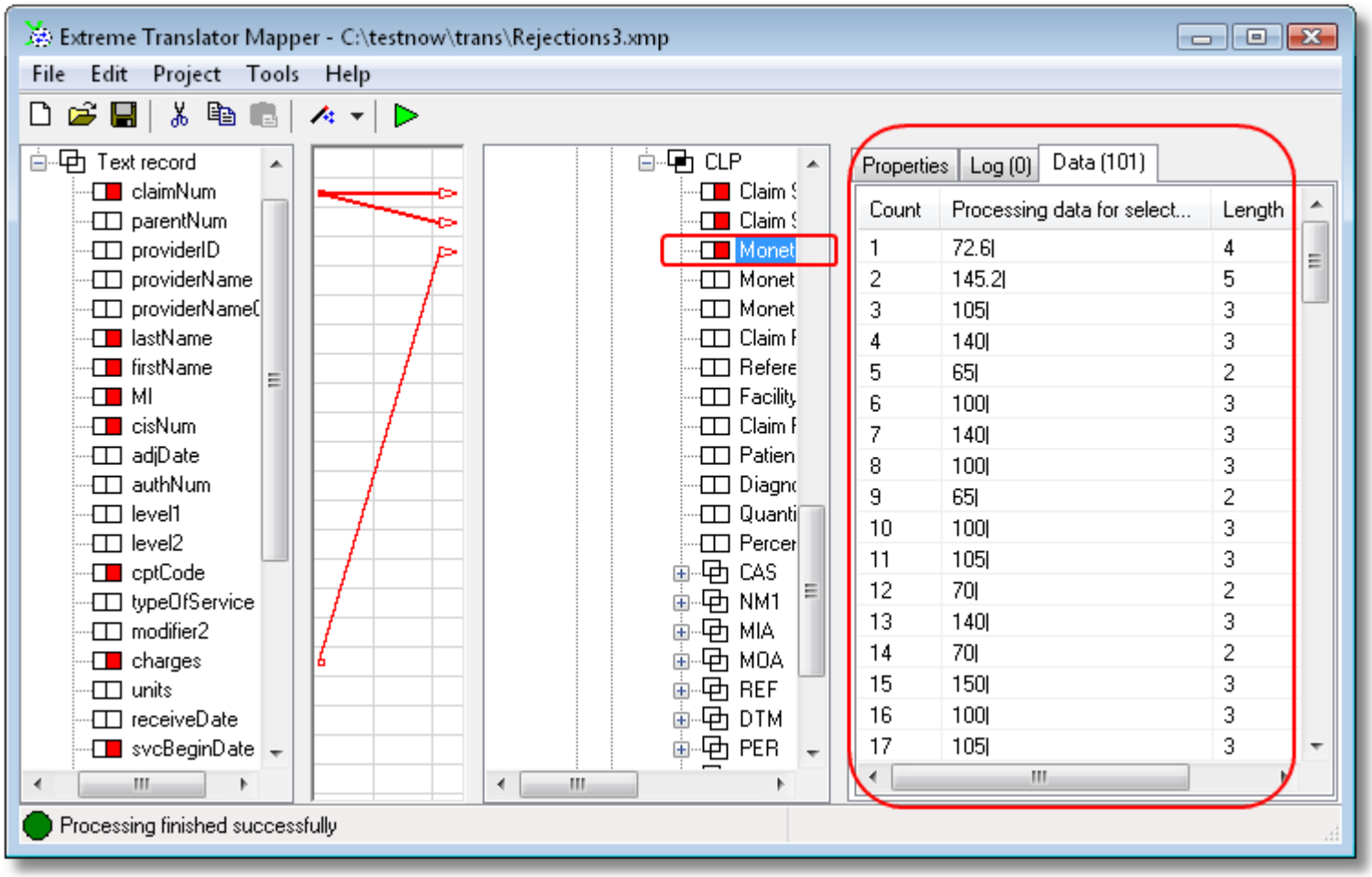
Warnings produced during execution.

Data Tab

It displays processing data attached to the map item. After processing each item in the map has data associated with it. That is whatever was retrieved from the file or written to the file gets displayed in Data Tab when you select map item.

Each item can have 0-to-many data lines attached to it. This is because most of the map items can loop. Each loop instance results in new data line.

Navigating Data Tab is easy way to find what data has been extracted from input and produced in the output.

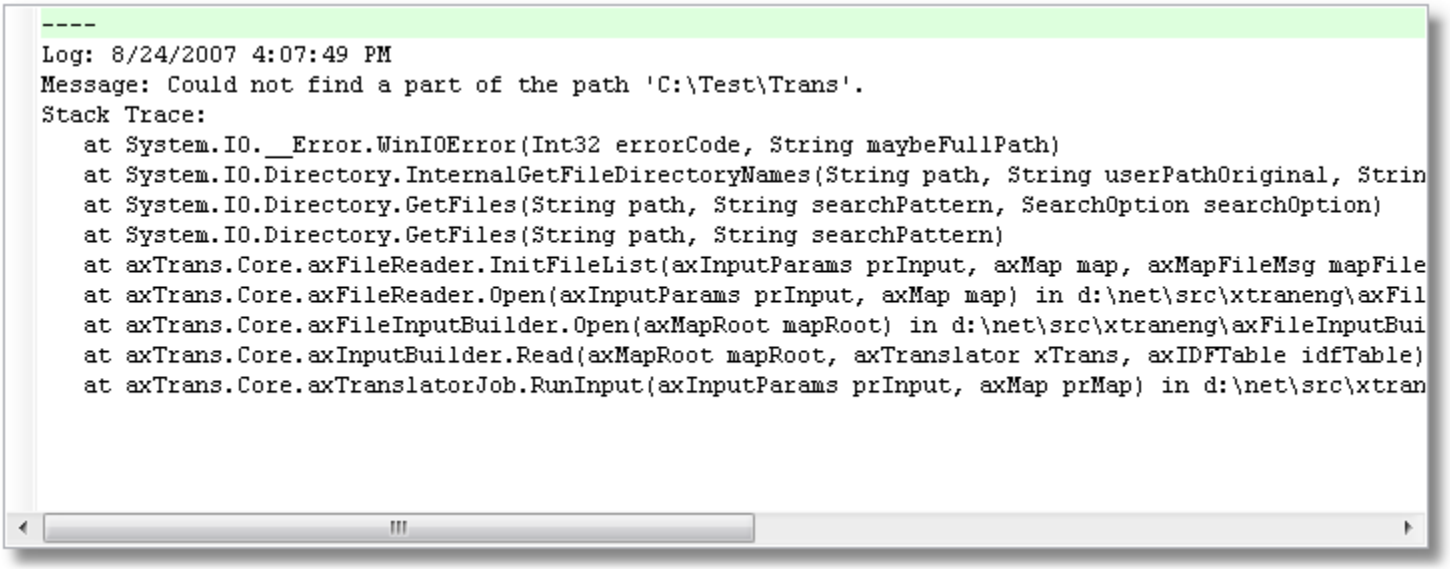


Data attached to and processed for each map item after execution. Data tab also shows item's length and marks end of item with "|". "|" character is handy if item has spaces at the end of data so you can see them.

Technical Support

When critical errors happen in the code during processing of the map translator writes log file. It contains instructions and function names that had failed. This log may not be useful for the end user but is excellent source of information for technical support.

Log file name starts with original map file name where error happened and ends with ".log".



```
----
Log: 8/24/2007 4:07:49 PM
Message: Could not find a part of the path 'C:\Test\Trans'.
Stack Trace:
   at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
   at System.IO.Directory.InternalGetFileDirectoryNames(String path, String userPathOriginal, Strin
   at System.IO.Directory.GetFiles(String path, String searchPattern, SearchOption searchOption)
   at System.IO.Directory.GetFiles(String path, String searchPattern)
   at axTrans.Core.axFileReader.InitFileList(axInputParams prInput, axMap map, axMapFileMsg mapFile
   at axTrans.Core.axFileReader.Open(axInputParams prInput, axMap map) in d:\net\src\xtraneng\axFil
   at axTrans.Core.axFileInputBuilder.Open(axMapRoot mapRoot) in d:\net\src\xtraneng\axFileInputBui
   at axTrans.Core.axInputBuilder.Read(axMapRoot mapRoot, axTranslator xTrans, axIDFTable idfTable)
   at axTrans.Core.axTranslatorJob.RunInput(axInputParams prInput, axMap prMap) in d:\net\src\xtran
```

There are sample log file contents.

Please visit “Support” page on our website for more information.