MLIS-5016 TransAccount

WP 2
Form Based Parser

Specification
Version 1.0 from 16/05/2001
Deliverable D2a

ALPNET Technology GmbH

AT internal

TITLE:	Form Based Parser
PROJECT:	TransAccount
AUTHOR:	Waldhör
ESTABLISHMENT:	AT
ISSUE DATE:	09/05/2001
DOCUMENT ID:	TransAccount_01/01
VERSION:	1.0
STATUS:	initial
DISTRIBUTION:	TransAccount
LOCATION:	formbasedparser.doc
KEYWORDS:	TransAccount, translation technology

Date:	Version:	Author:	Comments:
16.05.2001	1.0	Waldhör Klemens	

Contents

1 Changes to previous version

1.1 Version 1.0

New Version

Form Based Parser

2 SUMMARY

This document describes the form based parser which allows the transformation of account tables from a source language into a target language. Language in this context means different accounting systems. Parsing process is based on a rules file which contains the descriptions of the transformations to be applied.

3 Structure of the Document

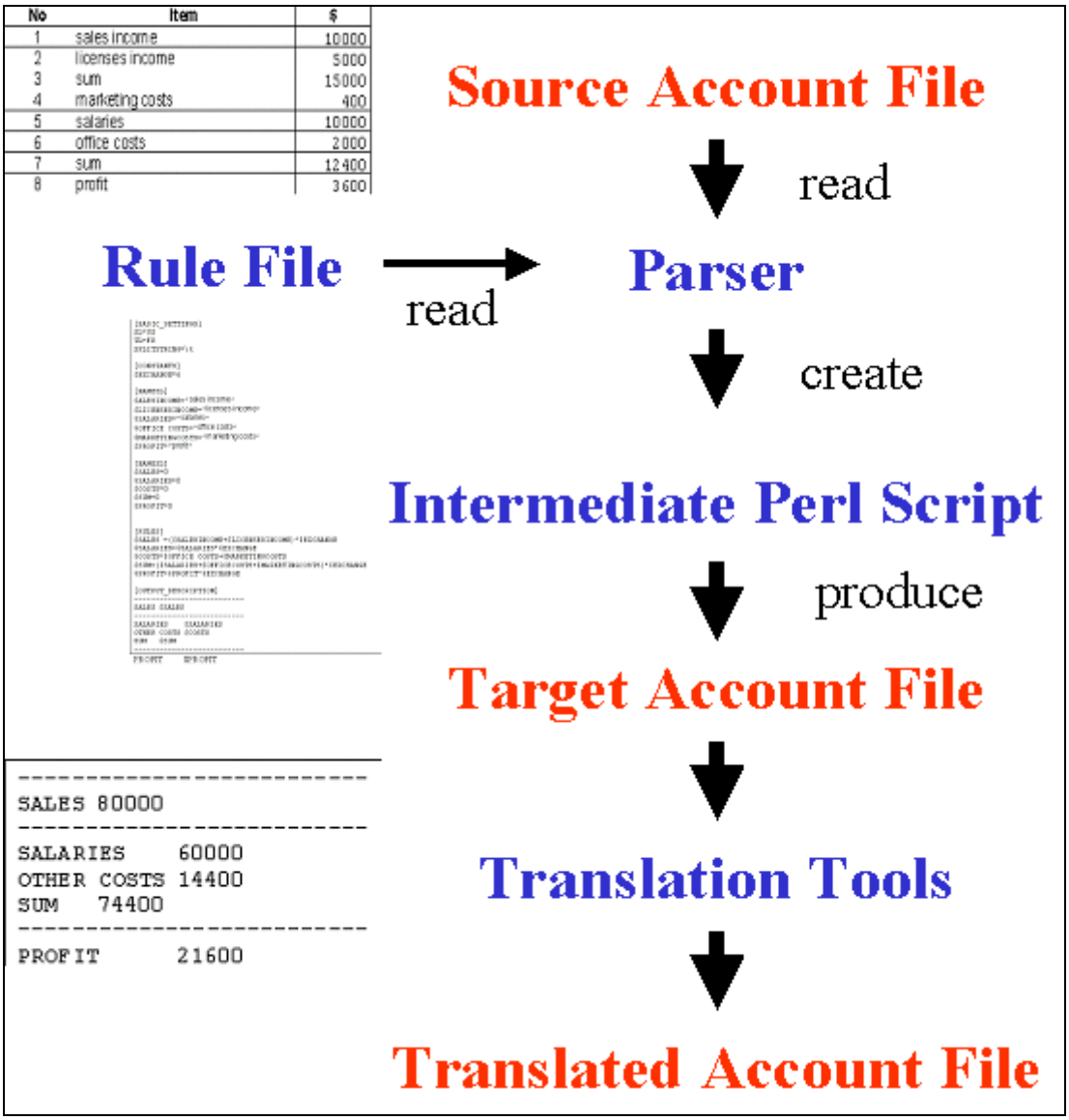
Chapter 4 explains the basic ideas, while chapter 5 explains how the contents of a rules file. An example is given there too.

4 Form Based Parser – Basic Idea

The form based parser is realised as a distinct step in the translation process which just applies to the tabular parts of the accounts. It is run before any other translation tool is applied and transforms a given table into a standardised (target) form based on a rule file. This form (document) produced can then be translated by various translation tools.

The parser supports in its current form simple tab separated files only. By adding an additional pre-transformation step other formats can be read too.

The following picture shows this in diagram form.



The parser itself works in the following steps.

Step	Action
1	Read the source account table and the rules file.
2	Apply the rules file to the source account table. This will produce a intermediate Perl script
3	Execute the intermediate Perl script
4	Return the transformed source account file

5 Form Based Parser Language

The description of the form based parser contains different sections which specify various functions.

The following sections are available:

```
BASIC_SETTINGS
NAMESSL
NAMESTL
CONSTANTS
RULES
OUTPUT_DESCRIPTION
```

5.1 Comments

Comments start with a "#" character:

Example:

```
# This is a comment
```

5.2 BASIC_SETTINGS

Here one defines general parameters for the parser process.

[BASIC_SETTINGS]	**Comment**
SL=<sourcelanguage>	This defines the source language
TL=<targetlanguage>	This defines the target language
SPLITSTRING=<string>	This identifies the split string used in the account form table.

Example: the following indicates that the fields in the form are separated by tabs.

```
[BASIC_SETTINGS]
SPLITSTRING=\t
SL=EN
TL=FR
```

5.3 NAMESSL

This section declares the variables to be used and how they are instantiated in the account table for the source language. This can be used to store the relationship between generic variables and actual names in a given form.

[NAMESSL]	Comment
$varname=value	The string appearing in the account table
$varname=regular expression	Here one can define a Perl regular expression to make matching easier and more general

Example:

```
[NAMESSL]
$PROFIT=Profit
$SALARY=Salar.*
```

5.4 NAMESTL

This section declares the variables to be used for the target language (form). A variable always starts with the $ sign. This can be used to store the relationship between generic variables and actual names in a given form.

[NAMESTL]	Comment
$varname=value	The string will appear in the transformed account table

Example:

```
[NAMESTL]
$PROFIT=0
```

5.5 CONSTANTS

This section allows the definitions of constants which can be used in expressions, e.g. for currency computations. A variable always starts with the $ sign.

[CONSTANTS]	Comment
$variable=value	Associates value with variable

Example:

```
[CONSTANTS]
$DOLLAR_TO_FF=6
$DM_TO_EURO=1.95583
```

5.6 RULES

The MAP rules define how variables are mapped from the source language to the target language. This can involve any valid Perl expression which results in a value, normally this will be a numerical value.

[RULES]	Comment
$varname=Perl expression	This defines a transformation of a source variable based expression into a target variable. The source expression may contain several source variable and constants which have to be defined in the

	appropriate sections. Note that variables and constants always have to start with a $.

Examples:

```
$loss=$losses
$sumlosses=$lossespart1+$lossespart2
$depreciations=($depreciation1+=$depreciation2)*0.35
$depreciationnew=($depreciation1+=$depreciation2)*$depreciationfactor
```

5.7 OUTPUT_DESCRIPTION

The output description describes for each target variable how to output it. In addition one might add constant text too. This enables one to re-arrange lines.

[OUTPUT_DESCRIPTION]	**Comment**
expression	A valid Perl string

Example :

```
[OUTPUT_DESCRIPTION]
SALES sales
--------------------------
SALARIES    $salaries
OTHER COSTS $costs
SUM   $sum
--------------------------
PROFIT      $profit
```

5.8 Example

Consider the following input table:

No	Item	$
1	sales income	10000
2	licenses income	5000
3	sum	15000
4	marketing costs	400
5	salaries	10000
6	office costs	2000
7	sum	12400
8	profit	3600

Let's assume for the target form to be generated marketing cost and office costs are always shown in just one number. In addition we only have one sales number. This can be achieved by the rule file shown later.

The desired output form should be:

No	Item	FF
1	$SALES	80000
2	$SUM	80000
3	$SALARIES	60000
4	$OTHERCOSTS	14400
5	$SUM	74400
6	$PROFIT	21600

The target table shows that lines 1 and 2 have been collapsed into one line 1 in the target table, lines 5 and 6 from source have been collapsed into line 4 of the target table. The values have been converted from $ to FF.

The following rule file will produce this result:

```
[BASIC_SETTINGS]
SL=US
TL=FR
SPLITSTRING=\t

[CONSTANTS]
$EXCHANGE=6

[NAMESL]
$ALESINCOME=”sales income”
$LICENSESINCOME=”licenses income”
$SALARIES=”salaries”
$OFFICE COSTS=”office costs”
$MARKETINGCOSTS=”marketing costs”
$PROFIT=”profit”

[NAMESL]
$SALES=0
$SALARIES=0
$COSTS=0
$SUM=0
$PROFIT=0


[RULES]
$SALES =($SALESINCOME+$LICENSESINCOME)*$EXCHANGE
$SALARIES=$SALARIES*$EXCHANGE
$COSTS=$OFFICE COSTS+$MARKETINGCOSTS
$SUM=($SALARIES+$OFFICECOSTS+$MARKETINGCOSTS)*$EXCHANGE
$PROFIT=$PROFIT*$EXCHANGE

[OUTPUT_DESCRIPTION]
-------------------------
SALES $SALES
-------------------------
SALARIES    $SALARIES
OTHER COSTS $COSTS
SUM   $SUM
-------------------------
PROFIT      $PROFIT
```

The transformed output file produced will be:

```
-------------------------
SALES 80000
-------------------------
SALARIES    60000
OTHER COSTS 14400
SUM   74400
-------------------------
PROFIT      21600
```

Note that there are two options now for getting a translation for the transformed file.

a) one can give the translation of strings directly in the `OUTPUT_DESCRIPTION` part.
b) Or one could just insert symbolic strings or source language strings which will later be translated by other translation tools.