

# **MLTS**

## **(Multi Location Translation System)**

### **2.0**

06.02.07

Meinolf Amekudzi

T-Systems Multimedia Solutions GmbH  
Dongleware Verlags GmbH

## Index

MLTS .....	1
(Multi Location Translation System) .....	1
2.0.....	1
1 Introduction .....	4
2 Language and Location Codes .....	4
3 Date, Time, Timezones and Daylight Saving Rules .....	5
3.1 ZoneInfo .....	5
4 Application Coded Data and Application Runtime Data .....	5
4.1 Application Coded Data .....	5
4.2 Application Runtime Data.....	5
4.2.1 Database Tables and COLUMNS .....	5
5 Text Phrases .....	6
5.1 Internal or External Text Phrase Resources .....	6
5.1.1 Internal Text Resource Mechanism.....	6
5.2 MLTS Text Phrase Format.....	7
5.2.1 Key Generation and Rules for Source Code Text Phrases.....	8
5.3 Information Tags .....	8
5.3.1 <sense/> Tag .....	8
5.3.1.1 Standard <sense/> <i>info</i> Attribute.....	8
5.3.1.1.1 <i>info</i> Attribute <i>city</i> .....	8
5.3.1.1.2 <i>info</i> Attribute <i>language</i> .....	8
5.3.1.1.3 <i>info</i> Attribute <i>country</i> .....	9
5.3.1.1.4 <i>info</i> Attribute <i>federalstate</i> .....	9
5.3.2 <lid/> Tag .....	9
5.4 Placeholder Tags .....	9
5.4.1 Static Placeholder Tags .....	9
5.4.1.1 Standard Placeholder Tags .....	9
5.4.2 Dynamic Placeholder Tags.....	10
5.4.2.1 <i>if</i> Attribute .....	11
5.4.2.1.1 <i>Equal To if</i> -Definitions.....	11
5.4.2.1.2 <i>Equal or Greater Than if</i> -Definitions .....	11
5.4.2.1.3 <i>Equal or Less Than if</i> -Definitions.....	11
5.4.2.1.4 <i>Not Equal To if</i> -Definitions.....	12
5.4.2.1.5 <i>Multiple Values if</i> -Definitions .....	12
5.4.2.2 <i>format</i> Attribute.....	12
5.4.2.2.1 <i>int</i> Formattype .....	12
5.4.2.2.2 <i>float</i> Formattype .....	13
5.4.2.2.3 <i>string</i> Formats .....	13
5.4.2.2.4 <i>language</i> Formats .....	13
5.4.2.2.5 <i>datetime</i> Formats .....	13
5.4.2.2.6 <i>gmtdatetime</i> Formats .....	13
5.4.2.2.7 <i>currency</i> Formats.....	13
5.4.2.2.8 <i>unit</i> Formats.....	13
5.4.3 Placeholder Macros .....	13
6 Pictures.....	14
7 Files .....	14
8 Best Language Function.....	14
9 Customized Translations .....	14

10	Database Conversions .....	14
	Appendix 1 - Example PHP Implementation of Central MLTS Function .....	15

## 1 Introduction

MLTS (Multi Location Translation System) is a system to provide an application with functions to use it in different languages and locations. The system encloses functions for translation of dynamic text phrases with all types of included units, dates and measurements as well as functions to translate pictures and files.

This are the main benefits of MLTS:

- On-the-fly translations into multiple languages at runtime.
- Support of external resource systems.
- Support of internal resource systems (e.g. text phrases are included in source code).
- XML like definition of text phrases: easy to learn and to read.
- Dynamic placeholders for grammar etc..
- Excluding of formatting functions from source code, including time conversions with daylight saving rules and measurement or currency calculations.
- Support of mixed source codes (different languages within a source code) and any auto conversions to any language or location.
- Easy possibility of im- and export to or from translation tools.
- Support of picture and file management and selections.
- Support of application- and customer-translation-layers to realize easy customizations of applications (text phrase translation overload).
- Old translations are available after changes to make it easier for translation editors to adapt changes. Changes could be recognized by an automatic system.

## 2 Language and Location Codes

Two values will be needed to specify a translation: the language and the location. Therefore ISO 639-1 will be used to define the language or a combination of ISO 639-1 and ISO 3166 will be used to define a language variation based on a special location. The combination form is represented by lowercase language code and an uppercase country code (e.g.. “enGB” for british english).

For national regions the extended ISO definition is used where an additional two character code indicate the federal state or region within a country.

Examples:

DEBW => Germany, Baden-Württemberg  
USCA => USA, California

Because there are problems by using names for columns in databases no minus character is used in MLTS.

### 3 Date, Time, Timezones and Daylight Saving Rules

All times and dates are represents as a floating point number given all seconds after 1.1.1970 0:00:00 UTC. The fraction of this value defines the location of time (*ZoneInfo*, see chapter 3.1). A value of zero means that time is given in real UTC, all other values defines locations of time. The value is an enumeration (multiply by 10000) of a list of *ZoneInfo* (see chapter 3.1). In this way the corresponding timezone and daylight saving rule can be calculated.

#### 3.1 ZoneInfo

The term *ZoneInfo* defines a national region within all information about timezones and daylight saving rules including all dynamic rules in past and future. E.g. same countries and federal states changed timezone and/or daylight saving rules in the past. With a given *ZoneInfo* it possible to calculate a correct local time for each UTC time in past and future.

A *ZoneInfo* is identified by continent or ocean and then by the name of the location, which is typically the largest city within the region. This definition is used by most Unix systems.

### 4 Application Coded Data and Application Runtime Data

There is a different between data within an application and data which is produced and used while an application is running. All *Application Coded Data* will be created by a programmer while developing an application, Application Runtime Data will be created by an user while an application is running.

#### 4.1 Application Coded Data

Handling of Application Coded Data is described in chapter 5.1.

#### 4.2 Application Runtime Data

##### 4.2.1 Database Tables and COLUMNS

To indicate columns and tables in databases for several languages a prefix of a language/location is recommend. In this way it is possible to create universal methodes for automatic translation workflows. In MLTS a prefix of type *MLTSllcc* is used where *ll* indicates the language and optional *cc* indicates the country (ISO code).

Example (columns of a news database):

```
MLTSenGB_modification_datetime  
MLTSenGB_news_teaser  
MLTSenGB_news_text  
MLTSdeGB_modification_datetime  
MLTSdeGB_news_teaser  
MLTSdeGB_news_text
```

## 5 Text Phrases

The most important part of MLTS are mechanisms and functions to handle text phrases. With MLTS it is possible to translate text phrases on the fly at runtime into multiple languages.

### 5.1 Internal or External Text Phrase Resources

There are two fundamental programming models to handle text phrases. An application can use internal or external resources. If internal resources are used, all text phrases are included in the source code of an application. If external resources are used, all text phrases are excluded in a separated file for each language and/or location. In this case the source code includes only references to the external resources.

Most multilingual systems are based on the resource programming model where text phrases are separated from source code and objects (windows, controls build via IDE etc.). This type of programming is not very intuitive, application outputs and corresponding source code positions are not easy to find, the programming process is interrupted continually by managing external resources. An other problem at resource based systems is to recognize changes at resources while developing a program.

MLTS can used at both types of programming models, but there is a special mechanism in MLTS to support internal resources to prevent problems that occurred at internal resource based systems.

#### 5.1.1 Internal Text Resource Mechanism

At a system which works with MLTS all text phrases can be placed within the source code and controls of a program. The text phrases in source code are enclosed with a function which handels translations at runtime. A special MLTS parser can recognize the enclosed text phrases in source code and synchronize them with a database. The collected text phrases can then translated by an editor or automaticly to any language and/or location. After database changes a special MLTS component generates files (e.g. at php a associative array) for any language so the central translation function can convert the enclosed text phrases easy at runtime.

The original text phase in source code is also the key to indicate the translation of a text phrase. (In php it will be the key of the created associative array.)

A simple php example:

The program line looks like this...

```
For($i=1;$i<=10;$++)MLT("This is a translation example!");
```

After MTLTS parser run the entry of our example database looks like this (at a systems which are configurated for english and german language) ...

<b>creation_date_src</b>	01.11.2006 11:11
<b>state_src</b>	used
<b>src</b>	This a translation example!
<b>modification_date_en</b>	01.11.2006 11:11
<b>state_en</b>	ok_by_source
<b>en</b>	This a translation example!
<b>modification_date_de</b>	01.11.2006 11:11
<b>state_de</b>	not_translated
<b>de</b>	This a translation example!

After an editor has translated the new text phrase, the database entry looks like this...

<b>creation_date_src</b>	01.11.2006 11:11
<b>state_src</b>	used
<b>src</b>	This a translation example!
<b>modification_date_en</b>	01.11.2006 11:11
<b>state_en</b>	ok_by_source
<b>en</b>	This a translation example!
<b>modification_date_de</b>	02.11.2006 11:11
<b>state_de</b>	manually_translated
<b>de</b>	Dies ist ein Übersetzungsbeispiel!

Now it is possible to generate a german associative array (in this example it will be manually created)...

```
$MLTS=array();
$MLTs["This is a translation example!"]=
    "Dies ist ein Übersetzungsbeispiel!";
file_put_contents("example.dat",serialize($MLTs));
```

...and uses it for on-the-fly-translations at runtime of program...

```
$MLTs=unserialize(file_get_contents("example.dat"));
function MLT($_txt){
global $MLTs;
    if(isset($MLTs[$_txt]){
        return $MLTs[$_txt];
    }else{
        return $_txt;
    };
};
For($i=1;$i<=10;$i++)MLT("This is a translation example!");
```

## 5.2 MLTS Text Phrase Format

To get more dynamic and better performance, a xml-based tag-system is defined to MLTS. So it is possible to include hidden descriptions, static and/or dynamic placeholders, grammar information, gender information, internationalization information, measurement units and many more functions into text phrases.

### 5.2.1 Key Generation and Rules for Source Code Text Phrases

There are some rules to keep while creating text phrases. It is not allowed to include control characters or multiple spaces into a text phrase. This is necessary because the system should be able to generate clear and unique keys for arrays. Another reason is that it should be possible to export/import MLTS databases to/from a standard clean xml format.

## 5.3 Information Tags

Information tags are tags without any direct translation function and will be removed at runtime by translation function. This type of tags gives additional information of text phrase for system and translation editors.

### 5.3.1 `<sense/>` Tag

With the help of this tags editors get more information about the context of a text phrase. An other import function of this type of tags is to prevent wrong translations based on double meaning text phrases. An attribute info is recommend to define a description of text phrase context.

#### 5.3.1.1 Standard `<sense/>` *info* Attribute

To describe sense and context of a text phrase the attribute info is used.

Example:

```
Power<sense info="physical; power of a car"/>
```

For several employments standard info values are defined:

##### 5.3.1.1.1 *info* Attribute *city*

This information can help to automate the translation of city names. E.g. an expression like `Munich<sense info="city"/>` could help to make an automatic translation of a given city. Therefore the system just have to add `<sense info="city"/>` after a given city name and a translation function call.

Example:

```
Munich<sense info="city"/>
```

##### 5.3.1.1.2 *info* Attribute *language*

This information can help to automate the translation of city names. E.g. an expression like

Example:

```
German<sense info="language"/>
```



#### 5.3.1.1.3 *info* Attribute *country*

This information can help to automate the translation of city names. E.g. an expression like

Example:

```
USA<sense info="country"/>
```

#### 5.3.1.1.4 *info* Attribute *federalstate*

This information can help to automate the translation of city names. E.g. an expression like

Example:

```
California<sense info="ferdederalstate" country="us" id="usCA"/>
```

### 5.3.2 <lid/> Tag

The lid tag is used to indicate the language/location of a text phrase. With this tag it is possible to use mixed language documents and controls in same projects or files.

Example:

```
<lid id="de"/>Dies ist eine deutsche Textphrase.
```

## 5.4 Placeholder Tags

There are two types of placeholder tags defined to MLTS: Static and dynamic placeholders. In both cases the name of the placeholder is used to define a tag with same name.

```
$whatIsIt="Place-Holder";  
echo MLT("This is a <placeholder/> example!",array("placeholder"=>$whatIsIt));
```

This will print in english...

This is a Place-Holder example!

...or in german...

Dies ist ein Place-Holder-Beispiel!

### 5.4.1 Static Placeholder Tags

Static placeholder tags are tags where only a given text will be simply replaced. This should be a string given by the call of the translation function via a associative array or standard placeholders.

#### 5.4.1.1 Standard Placeholder Tags

Standard placeholder tags are pre defined by system and could be overload by user placeholder tags.

`<cr/>`

Includes a cr-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<n/>`

Includes a cr-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<r/>`

Includes a cr-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<tab/>`

Includes a tabulator character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<br/>`

Includes a cr-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<space/>`

Includes a space-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<nbsp;/>`

Includes a non break space-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<newline/>`

Includes a cr-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<lt/>`

Includes a <-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

`<gt/>`

Includes a <-character into text phrase.

Optional Attribute *noof* (default value 1): number of returned characters.

slashnewline

## 5.4.2 Dynamic Placeholder Tags

A useful benefit are dynamic placehodler tags. In this type of tags additional attributes defines the formatting of the returned placeholder. There is also a mechanism to get conditional depended translations (e.g. in depended of grammer aspects).

### 5.4.2.1 *if* Attribute

In applications it is often necessary to output text phrases dynamically based on placeholder values. Therefore the special attribute *if* is defined by MLTS. The text which is included in a placeholder tag within an *if*-attribute is only returned if the condition is true.

```
<myplaceholder tag if="condition"/>
```

The standard condition of the *if*-attribute value is *value is equal to placeholder*.

Example:

```
<noof if="0">no apples</noof>
```

...if placeholder value of *noof* is zero the words *no apple* will be returned.

For more flexibility there are more condition syntax definitions (see following chapters).

It is possible to make combinations with different conditions and enumerations. Example:

```
<noof if="0">There are no apples.</noof><noof if="1">There is one  
apple.</noof><noof if="2|3+">There are <noof/> apples.</noof>
```

Value could be numerics or strings. If strings are used, the alphabetic order will be calculated.

#### 5.4.2.1.1 *Equal To if-Definitions*

A subsequent “=”-character defines a *equal to* condition. This is the standard if condition, so it could be left out.

Example:

```
<noof if="2=">apples</noof>
```

...if value of *noof* is 2 the word *apples* will be returned.

The example above is the same as this:

```
<noof if="2">apples</noof>
```

#### 5.4.2.1.2 *Equal or Greater Than if-Definitions*

A subsequent “+”-character defines a *greater than* condition.

Example:

```
<noof if="2+">apples</noof>
```

...if value of *noof* is equal or greater than 2 the word *apples* will be returned.

#### 5.4.2.1.3 *Equal or Less Than if-Definitions*

A subsequent “-”-character defines a *greater than* condition.

Example:

```
<noof if="0-">apples</noof>
```

...if value of *noof* is equal or less than 0 the word *apples* will be returned.

#### 5.4.2.1.4 *Not Equal To if-Definitions*

A subsequent “!”-character defines a *not equal to* condition.

Example:

```
<noof if="0!">apples</noof>
```

...if value of *noof* is any value except zero the word *apples* will be returned.

#### 5.4.2.1.5 *Multiple Values if-Definitions*

‘|’-characters are used to define multiple values to an *if*-definition (logical OR).

Example:

```
<noof if="2|3|4">apples</noof>
```

...if value of *noof* is 2, 3 or 4 then the word *apples* will be returned.

### 5.4.2.2 *format Attribute*

The format Attribute is used to define formatting of the placeholder value. The general syntax is:

```
<xxx format="formattype:[languageid]:[options:]formatparameters"/>
```

The format Attribute is used to define formatting of the placeholder value. The general syntax

#### 5.4.2.2.1 *int Formattype*

The int Formattype is used to format integer values.

```
<num format="int:[signs]:[zeros]:[thousands_sep]"/>
```

signs

[+][-][ ]

indicates whether a sign should be set as prefix. Default is “-“.

zeros

[number][ ]

indicates how many zeros should be set as prefix. Default is “” (not more than needed).

thousands\_sep

[char][ ]

indicates whether a sign should be set as prefix. Default is “.”.

Example

Format="int::5:," => "-10,000"

#### 5.4.2.2.2 *float* Formattype

The float Formattype is used to format floating point values.

```
<num format="float::"/>
```

#### 5.4.2.2.3 *string* Formats

The int Formattype is used to format integer values.

```
<num format="string:[lower|upper|capital]"/>
```

#### 5.4.2.2.4 *language* Formats

MLT('This text is written in <writtenlanguage format="language::lower"/>.',  
Array('writtenlanguage =>\$mylanguagecode);

#### 5.4.2.2.5 *datetime* Formats

```
<num format="datetime:[languageid]:[location]:formatinfo"/>
```

#### 5.4.2.2.6 *gmtdatetime* Formats

```
<num format="datetime:[languageid]:[location]:formatinfo"/>
```

MLT('My birthday is <mydate format="datetime:en:America/New York:l dS of F Y"/>',  
Array('mydate'=>'gmtmktime()+\$curUser->timeLocal);  
MLT('My birthday is <mydate format="datetime::l dS of F Y"/>',  
Array('mydate'=>'gmtmktime()+\$curUser->timeLocal);

#### 5.4.2.2.7 *currency* Formats

#### 5.4.2.2.8 *unit* Formats

sdsdsd

### Recommended Placeholder Tag Names and Use

```
<genderOfxxxx/>
```

...

### 5.4.3 Placeholder Macros

Placeholder macros have to be programmed explicit by functions.

Example:

```
MLT('Current date is <mydate/>',  
    Array('mydate'=>MLT_ShortDate(gmtmktime()+$curUser->timeLocal));  
  
Function MLT_ShortDay($_date){  
    return MLT('<shortdate format="datetime:en::l dS of F Y"/><sense  
    info="general formatting of short day representation"/>',  
        Array('shortdate'=>$_date);  
};
```

## 6 Pictures

To handle pictures for different locations and languages a function MLTPic is used. This function checks whether a located picture is available in file system and returns the pathfilename of best available picture. \$\_picturepathfilename specified the base pathfilename of the picture.

```
MLTPic($_picturepathfilename)
```

```
MLTPicture(APP_BASE."/test.jpg");  
.../test_en.jpg  
.../test_de.jpg
```

## 7 Files

In the same way as pictures are handled  
MLTFile(APP\_BASE."/test.jpg");

## 8 Best Language Function

If there are multiple translation available a function will be needed to get the best language.

## 9 Customized Translations

At many standard applications there is a demand to support customize translations. E.g. a ticket system should send e-mails adapt to a company. There is a simple way to prevent customize translations in MLTS. Just a second database is used to define all customized text phrases. All standard text phrases will then overload with customized text phrases.

## 10 Database Conversions

Because text phrases are stored in a classic databases, many format conversions are available, such XML- or an EXCEL-sheet expor, to support infaces to serveral translation tools. There are no problems with double spaces or special control characters.



## Appendix 1 - Example PHP Implementation of Central MLTS Function

```

function MLTNew($_text,$_placeholders=array(),
    $_replacePlaceholders=true,$_doNotTranslate=false){
    global $mlts,$MLT_RESERVED,$LANGUAGES,$PreWorkMLTtoRpls,$PreWorkMLTrplTos;
    // cleaning text to get translation key for associative array...
    $_text=trim($_text);
    $_text=str_replace($PreWorkMLTtoRpls,$PreWorkMLTrplTos,$_text);
    // make translation via associative array...
    //if(isset($mlts[$_text])&&!$_doNotTranslate)$_text=$mlts[$_text];
    if($_replacePlaceholders){
        $_text='<mtls>'.$_text.'</mtls>';
        // parse placeholder tags...
        if(preg_match_all("/([^\>\<]*)<([\A-Za-z0123456789_]*)([^\>]*)>([^\>\<]*)/m"
            ,$_text,$matches,PREG_SET_ORDER)){
            $level=0;
            $levelremoves=array(false);
            $c=count($matches);
            for($i=0;$i<$c;$i++){
                // matches: 0=not used, 1=not used, 2=tag, 3=attributes, 4=value(text)
                $count=1;
                $isBeginTag=true;
                $isEndTag=false;
                $tag=trim($matches[$i][2]);
                if($tag{0}=='/'){
                    $tag=substr($tag,1);
                    $isEndTag=true;
                    $isBeginTag=false;
                }
                $paratxt=trim($matches[$i][3]);
                if($paratxt!=''){
                    // parse and handle placeholder attributes...
                    if($paratxt{strlen($paratxt)-1}=='/'){
                        $paratxt=substr($paratxt,0,strlen($paratxt)-1);
                        $isBeginTag=false;
                    }
                }
                if($isBeginTag){
                    $level++;
                    $levelremoves[$level]=$levelremoves[$level-1];
                }
                if($isEndTag){
                    $level--;
                }
            }
        }
    }
}
  
```

```

};
if(!$levelremoves[$level]){
  if($isEndTag){
    $placeholdervalue='';
  }else{
    $placeholdervalue=$_placeholders[$tag];
  };
  // parse attributes of tag...
  $attrtxt=$paratxt;
  while($attrtxt!=''){
    $attrtxt=ltrim($attrtxt);
    $attrname=strnextcut(' =',$attrtxt,$char);
    $attrvalue='';
    if($char=='='){
      $attrvalue=strnextcut('"',$attrtxt,$char);
      $attrvalue=strnextcut("'",$attrtxt,$char);
    };
    switch($attrname){
      case'format':
        $e=explode(':', $attrvalue);
        switch($e[0]){
          case'language':
            switch($e[1]){
              case'longLower':
                $placeholdervalue=strtolower($LANGUAGES[$placeholdervalue]);
                break;
              case'longCapital':
                $placeholdervalue=$LANGUAGES[$placeholdervalue];
                break;
            };
            break;
          case'gmtdatetime':
            break;
          case'gmttime':
            break;
          case'localdatetime':
            break;
          case'localtime':
            break;
          case'count':
            break;
          case'float':
            break;
        };
        break;
      case'info':
        break;
      case'count':
        $count=$attrvalue+0;
        break;
      case'if':
        // check multiple conditions separated by |-characters...
        $ifs=explode('|', $attrvalue);
        $levelshow=false;
        foreach($ifs as $if){
          // greater than or less than operators present?...
          switch($if{strlen($if)-1}){
            case'+':
              $levelshow|=( $placeholdervalue>=substr($if,0,strlen($if)-1));
              break;
            case'-':
              $levelshow|=( $placeholdervalue<=substr($if,0,strlen($if)-1));
              break;
            case'!':
              $levelshow|=( $if!=$placeholdervalue);
            case '=':
              $levelshow|=( $if==$placeholdervalue);
            default:
              $levelshow|=( $if==$placeholdervalue);
          };
        };
        $levelremoves[$level]=!$levelshow;
        $placeholdervalue='';
        break;
      };
    };
  };
}

```



```

    };
    }else{
        $placeholdervalue='';
    };
    }else{
        if($tag{strlen($tag)-1}=='/'){
            $tag=substr($tag,0,strlen($tag)-1);
            $isBeginTag=false;
        };
        if($isBeginTag){
            $level++;
            $levelremoves[$level]=$levelremoves[$level-1];
        };
        if($isEndTag){
            $level--;
        };
        if($isEndTag){
            $placeholdervalue='';
        }else{
            $placeholdervalue=$_placeholders[$tag];
        };
    };
    if($levelremoves[$level]){
        $tag='';
        $placeholdervalue='';
        $matches[$i][4]='';
        $matches[$i][2]='';
    };
    switch($tag){
        case'mlts':
            $matches[$i][2]='';
            break;
        case'sense':
            $matches[$i][2]='';
            break;
        case'lid':
            $matches[$i][2]='';
            break;
        case'newline':
            $matches[$i][2]="\n";
            break;
        case'nbsp':
            $matches[$i][2]=str_repeat('&nbsp;',$count);
            break;
        case'space':
            $matches[$i][2]=str_repeat(' ', $count);
            break;
        case'br':
            $matches[$i][2]='<br/>';
            break;
        case'tab':
            $matches[$i][2]="\t";
            break;
        case'gt':
            $matches[$i][2]='>';
            break;
        case'lt':
            $matches[$i][2]='<';
            break;
        default:
            $matches[$i][2]=$placeholdervalue;
            break;
    };
    };
    // assemble final translated text including all calculated placeholders..
    $_text='';
    for($i=0;$i<$c;$i++)$_text.=$matches[$i][2].$matches[$i][4];
    };
    };
    return $_text;
};

```