TFF

Text Formatting Functions

The TFF string function provides functions for formatting text. The general syntax is:

```
TFF (string-value [,optional-arguments,…], option-string
[,ERR=line-ref|,ERC=error-code])
```

The text string to be formatted is specified by *string-value* and the function is specified by *option-string*. The number of *optional-arguments* depends on the function. The available functions are described in the following sections.

String Search and Replace functions

This TFF string function performs search and replace functions on a string.

SYNTAX

```
TFF (string-value, search-for, replace-with [,separator], option-string
[,ERR=line-ref|,ERC=error-code])
```

string-value	is the string to be searched.
search-for	is a string specifying one or more values to search for.
replace-with	is a string specifying one or more replacement values.
separator	is an optional separator character.
option-string	is a string specifying the option code.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

OPTION CODES

This TFF function searches the *string-valu*e replacing each matching *search-for* value with the corresponding *replace-with* value. If the optional separator character is included, multiple search and replace pairs are processed in a single function call. Appending a plus sign to the *option-string* causes leading and trailing spaces and tabs to be removed from each *replace-with* value.

"**S**"	Replace all matching search strings.
"**s**"	Replace only the first matching search string.

EXAMPLES

```
LET U$ = TFF("Your code is %C", "%C", "1234", "s" )
```

U$ will contain “Your code is 1234”

```
LET S$ = TFF("Dear %NAME;", "%NAME", #UTCUST.CUST-NAME, "s+")
```

S$ contains “Dear Warren Baseball Club;” with trailing spaces removed from CUST-NAME.

```
LET SEP$="|";

LET A$="[[sessionid]]" + SEP$ + "[[format]]";

LET B$=SESSION$ + SEP$ + FORMAT$;

LET HTML$ = TFF(HTML$,A$,B$,SEP$,"S")
```

SEP$ is the character used to separate search and replace strings. Every occurrence of "[[sessionid]]" in HTML$ will be replaced with the contents of SESSION$ and every occurrence of "[[format]]" in HTML$ will be replaced with the contents of FORMAT$. If there will be only one match for each search argument, a lower case **"s"** *option-string* should be used to improve performance.

Internet Uniform Resource Locator functions

This TFF string function implements encoding and decoding of Internet URL strings.

```
TFF (string-value, option-string [,ERR=line-ref|,ERC=error-code])
```

string-value	is the string to be encoded or decoded.
option-string	is a string specifying the option code.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

OPTION CODES

URL encoding converts all but specific ASCII characters into a %HH notation consisting of a percent sign character and two hexadecimal characters. Alphanumeric characters 0-9, A-Z, and a-z, and the special characters !()*-._~ are not encoded. Decoding converts an encoded string back to its original form.

"E"	Encode *string-value*.
"D"	Decode *string-value*.

"**E+**"	Encode string-value and replace space characters with a plus sign.
"**D+**"	Decode string-value and replace plus sign characters with a space.

EXAMPLES

```
LET U$ = TFF("This is a simple & short test","E")
```

U$ will contain “This%20is%20a%20simple%20%26%20short%20test”

```
LET U$ = TFF("This is a simple & short test","E+")
```

U$ will contain “This+is+a+simple+%26+short+test”

```
PRINT TFF(U$,"D")
```

If U$ contains the result of the first example, “This is a simple & short test” will be printed.

```
PRINT TFF(U$,"D+")
```

If U$ contains the result of the second example, “This is a simple & short test” will be printed.

Encrypt a String function

This TFF string function uses internal cipher routines to encrypt a string.

```
TFF (string-value, key-string, option-string
[,ERR=line-ref|,ERC=error-code])
```

string-value	is the string to be encrypted or decrypted.
key-string	is a string specifying an encryption key or password. Minimum size is 4 characters. Maximum size is 8 characters.
option-string	is a string with the value **"C"** to select this function.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

REMARKS

This TFF function uses internal cipher routines with the specified *key-string* to produce a random eight-byte string. The XOR string function is then called to reverse random bits in *string-value* to produce the result string. When *string-value* is the result of a previous encryption and the original *key-string* is provided, the new result will be the original *string-value*.

This function produces strings containing binary characters that could be misinterpreted by applications, such as the Basic Field Separator ($8A$), the Basic Escape ($1B$), ASCII control characters such as TAB (09), and String Terminators (00 and 24). The HTA string function may be useful to convert an encrypted string to ASCII format.

Since a particular *key-string* will always produce the same random string, *key-strings* should be changed periodically.

EXAMPLES

```
LET E$ = TFF("123-45-6789","SSNKEY","C")
```

E$ will contain $F59D6CB0DC97540CF39766$

```
LET A$="123 Old Lake Shore Road", P$="PASSWORD";
LET B$=TFF(A$, P$, "C"), C$=TFF(B$, P$, "C")
```

B$ will contain the encrypted result string and C$ will contain the same value as the original A$.

XML output function

This TFF string function produces an XML formatted string from a FORMAT.

```
TFF (format-name [,data-names], option-string
[,ERR=line-ref|,ERC=error-code])
```

format-name	is a string specifying the name of a format that has been loaded into memory using the FORMAT INCLUDE directive and populated with data.
data-names	is an optional string specifying data elements to be selected from *format-name*.
option-string	is a string specifying the option codes.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

OPTION CODES

This function creates an XML formatted string from all or selected elements of a format in memory. The *option-string* begins with "XO" and may be followed by any of the optional codes listed below. See the remarks section for complete descriptions. Commas, spaces and tabs may be included to improve readability.

"XO"	Selects the XML Output function.
"C"	Include Century in dates.

"D"	Use data element descriptions.
"R"	Special character replacement.
"E"	Special character encoding.
"e"	Special character encoding without semicolons.
"Y"	Include empty fields.

REMARKS

The optional *data-names* argument is used to select data elements to be output. The string contains a list of 20 character data names. Non-matching data names are ignored. All elements in the format are output if *data-names* is omitted.

Each selected data element is formatted with an XML Start-Tag, the data element's contents converted to text, and an XML End-Tag. Dashes in the XML tags are replaced with underscores. The optional codes invoke additional formatting as needed.

Five markup delimiters &, <, >, ', and " are prohibited in XML tags. These characters are automatically encoded with &amp;, &lt;, &gt;, &apos;, and &quot;. The "E" option requests that this encoding be applied to the formatted text. The "e" option does not produce the semicolons.

The "D" option requests that data descriptions contained in *format-name* be used to create the XML tags. A language selection must have been present in the #IDSV system format when the FORMAT INCLUDE for *format-name* was executed. This option is ignored if descriptions are not available.

The "R" option specifies character replacement. This option requires arguments in the form: DFDTD, where D is a user selected delimiter character, F is one or more characters to be replaced, and T is zero or more replacement characters. The readability characters are considered normal characters in these arguments. Characters from the F list found in the formatted text are replaced with corresponding characters in the T list. When the F list is longer than the T list, the extra F list characters are removed from the text.

The "C" option requests that the century be output if included in a date element. The default is two year digits.

The "Y" option requests that the XML tags be output when a data element is considered empty. The default is to skip empty elements.

EXAMPLES

For the examples the following code is used to FORMAT INCLUDE and populate a FORMAT. Line Feeds are inserted in the output for clarity.

```
FORMAT INCLUDE #TFFORDER
LET #TFFORDER.ITEM-CODE="2008RC"
LET #TFFORDER.DESCRIPTION="Rocking Chair"
LET #TFFORDER.ORDER-QUANTITY=2
LET #TFFORDER.ORDER-DATE=DTN("060708","MMDDYY")
```

This is an example of the default XML formatting. Dashes have been replaced with underscores.

```
LET XML$=TFF("#TFFORDER","XO"); PRINT XML$

<ITEM_CODE>2008RC</ITEM_CODE>
<DESCRIPTION>Rocking Chair</DESCRIPTION>
<ORDER_QUANTITY>2</ORDER_QUANTITY>
<ORDER_DATE>06/07/08</ORDER_DATE>
```

This example shows how the **"D"** option code uses Data Descriptions to create XML tags. Spaces in the descriptions are replaced with underscores.

```
LET XML$=TFF("#TFFORDER","XOD"); PRINT XML$

<Item_Code>2008RC</Item_Code>
<Item_Description>Rocking Chair</Item_Description>
<Quantity_Ordered>2</Quantity_Ordered>
<Order_Date>06/07/08</Order_Date>
```

This example selects a single Data Element and uses the **"C"** option code to include century in a date. Complete formatting of dates can be accomplished by using the DM= Valid Value option in the Format.

```
LET XML$=TFF("#TFFORDER",PAD("ORDER-DATE",20),"XOC"); PRINT XML$

<ORDER_DATE>06/07/2008</ORDER_DATE>
```

This example shows how to select and control the order of Data Elements.

```
LET SEL$=PAD("ORDER-DATE",20)+PAD("ITEM-CODE",20)
LET XML$=TFF("#TFFORDER",SEL$,"XOD"); PRINT XML$

<Order_Date>06/07/08</Order_Date>
<Item_Code>2008RC</Item_Code>
```

XML input function

This TFF string function uses an XML formatted string to update a FORMAT.

```
TFF (format-name, xml-string [,data-names], option-string
[,ERR=line-ref|,ERC=error-code])
```

format-name	is a string specifying the name of a format that has been loaded into memory using the FORMAT INCLUDE directive.
xml-string	is a string containing XML formatted data.
data-names	is an optional string specifying data elements to be selected from *format-name*.
option-string	is a string specifying the option codes.
line-ref	is the program line number or label to branch to if an error is produced by this function.
error-code	is a programmer-defined error code. Valid values are positive or negative whole numbers.

OPTION CODES

This function uses data extracted from an XML formatted string to update elements of a format in memory. The *option-string* begins with "XI" and may be followed by any of the optional codes listed below. See the remarks section for complete descriptions. Commas, spaces and tabs may be included to improve readability.

"XI"	Selects the XML Input function.
"D"	Use data element descriptions.
"R"	Special character replacement.
"E"	Special character decoding.
"e"	Special character decoding without semicolons.
"L"	Produce a list of data elements that were updated.

REMARKS

This function uses XML tags in *xml-string* to select data elements in *format-name*. Non-matching tags are ignored. The output of this function is an XML formatted list of errors that prevented a successful update. Each error entry is formatted <ERR>NNNTag</ERR>, where NNN is a Basic error number from –99 to 999 and Tag is the XML Start-Tag in *xml-string*.

The optional *data-names* argument is used to select data elements to be updated. The string contains a list of 20 character data names. Data names matching XML tags but missing from *data-names* will be skipped.

Five markup delimiters &amp;, &lt;, &gt;, &apos;, and &quot; are automatically decoded in XML tags as &, <, >, ’, and ". The "E" option requests that this decoding be applied to the formatted text. The "e" option does not require the semicolons.

The "D" option requests that data descriptions contained in *format-name* be used to match XML tags. A language selection must have been present in the #IDSV system format when the FORMAT INCLUDE for *format-name* was executed. This option is ignored if data descriptions are not available. When the *data-names* argument is included, a data name selected by matching a description must also exist in *data-names*.

The "R" option specifies character replacement. This option requires arguments in the form: DFDTD, where D is a user selected delimiter character, F is one or more characters to be replaced, and T is zero or more replacement characters. The readability characters are considered normal characters in these arguments. Characters from the F list found in the XML formatted text are replaced with corresponding characters in the T list. When the F list is longer than the T list, the extra F list characters are removed from the text.

The "L" option produces a list of data names that were successfully updated. The list follows the list of errors in the output string.

EXAMPLES

The examples refer to a FORMAT named #TFFORDER defining four Data Elements shown in the table below. A successful FORMAT INCLUDE #TFFORDER has been executed and language **"EN"** has been set in the #IDSV system format.

Element Name	Size	Type	EN Description
ITEM-CODE	6	Character	Item Code
DESCRIPTION	20	Character	Item Description
ORDER-QUANTITY	3.0	Integer	Quantity Ordered
ORDER-DATE	6.0	SQL Date	Order Date

In the example below the formatted string in XML$ is used to set all four fields in FORMAT #TFFORDER. Underscores are automatically converted to dashes.

```
XML$="<ITEM_CODE>2008RC</ITEM_CODE>
      <DESCRIPTION>Rocking Chair</DESCRIPTION>
      <ORDER_QUANTITY>2</ORDER_QUANTITY>
      <ORDER_DATE>06/07/08</ORDER_DATE>"

LET ERR$=TFF("#TFFORDER",XML$,"XI")
```

In the example below the **"D"** option code indicates Data Descriptions are used in the XML string in place of Element Names. The order of Data Elements does not need to match the XML data. Underscores are automatically converted to spaces.

```
XML$="<Quantity_Ordered>2</Quantity_Ordered>
      <Item_Code>2008RC</Item_Code>
      <Item_Description>Rocking Chair</Item_Description>
      <Order_Date>06/07/08</Order_Date>"

LET ERR$=TFF("#TFFORDER",XML$,"XID")
```

This example shows how DM= can be used in an XML Tag to specify the format of a date in the XML data.

```
XML$="<ORDER_DATE DM=YYYYMMDD>20080706</ORDER_DATE>"

LET ERR$=TFF("#TFFORDER",XML$,"XI")
```

This example shows an update list produced by the **"L"** option code. The example also shows how to use a *data-names* list to select specific XML Tags. The *data-names* list must be in the same order as the XML data for the update list to match.

```
XML$="<ITEM_CODE>2008RC</ITEM_CODE>
      <DESCRIPTION>Rocking Chair</DESCRIPTION>
      <ORDER_QUANTITY>2</ORDER_QUANTITY>
      <ORDER_DATE>06/07/08</ORDER_DATE>"

LET SEL$=PAD("ORDER-QUANTITY",20)+PAD("ORDER-DATE",20)
LET LST$=TFF("#TFFORDER",XML$,SEL$,"XIL"); PRINT LST$
<updatelist>ORDER-QUANTITY      ORDER-DATE          </updatelist>
```

This example output from the TFF() function shows that ORDER-DATE was updated before the ERR=167 was detected in the data for ORDER-QUANTITY. XML Tags are used in the error list and Data Element names are used in the update list.

```
XML$="<ORDER_DATE>06/07/08</ORDER_DATE>
      <ORDER_QUANTITY>1234</ORDER_QUANTITY>"

LET ERR$=TFF("#TFFORDER",XML$,"XIL"); PRINT ERR$
<ERR>167ORDER_QUANTITY</ERR><updatelist>ORDER-DATE        </updatelist>
```