1. Identification of the proposed change

1.1. Title

PRODUCE library function

1.2. MDC Proposer and Sponsor

This proposal originates from Ed de Moel.
Motions regarding the status of this document will be made by Taskgroup 2 (String Handling) of Subcommittee 13 (Data Management and Manipulation).
Ed de Moel can be reached at:

· 3950 Mahaila Avenue, apartment K12, San Diego, California 92122
· home phone: 619 455 7107
· office phone: 619 535 7566
· telefax: 619 535 7627
· email: demoel@fwva.saic.com

1.3. Motion

No motion; final MDC Type A write-up.

1.4. History of MDC actions

Date	Document	Action
October 1995	X11/95-111	Final MDC Type A write-up.
June 1995	X11/SC13/95-4	Document reformatted. Presented for elevation to MDC Type A. Passed 25:3:10.
January 1995	X11/SC13/94-41	No modifications. Presented for elevation to MDC Type A. Reformatting suggested, not presented for elevation.
June 1994	X11/SC13/94-29	No modifications. Presented for elevation to SC#13 Type A. Passed 20:0:5
February 1994	X11/SC13/94-17	Editorial inconsistencies corrected. Presented to replace SC#13 Type B. Accepted 18:0:3.
February 1994	X11/SC13/94-4	Document brought in the format defined after the October meeting. Presented for elevation to MDC Type A. Editorial inconsistencies detected. Document not voted on.
October 1993	X11/SC13/93-53	Presented for promotion to MDC Type A. Vote postponed until document has been brought in concordance with the "format for library proposals" that is to be defined at a later time during this meeting.
June 1993	X11/SC13/93-33	Presented for promotion to SC#13 Type A. Accepted 13:3:5. Only con raised: libraries not yet formalized.
February 1993	X11/SC13/93-5	History section corrected. Submitted for promotion to SC#13 Type A. Proposal modified in taskgroup, and presented as a replacement Type B. Accepted 16:1:3.
October 1992	X11/SC13/92-63	Corrections applied as discussed in taskgroup. Accepted as a replacement for a SC#13 Type B (17:0:2).
October 1992	X11/SC13/92-37	Correction applied as discussed and passed during previous meeting. Presented for promotion to SC#13 Type A, but modified in taskgroup.
June 1992	X11/SC13/92-13	Corrections applied as discussed during previous meeting. Proposal not accepted as SC#13 Type A.

February 1992	X11/SC13/91-14	History section put in correct order. Syntax changed to be a library function per subcommittee guidance. Presented for promotion to SC#13 Type B.
October 1991	X11/SC8/91-9	MDC#52 (St. Louis) Proposal modified to work without exceeding the portable subscript limitations. Substantive change, presented for promotion to Type B proposal of SC#13.
August 1991	no number	Discussion about the subject through Forum/InterNet.
January 1990	X11/SC8/89-17	MDC#44 (San Francisco), updated proposal for MDC, subcommittee #8. Proposal proposed for promotion to a type A proposal of subcommittee #8, but not promoted. The possibility that the function may not terminate needs to be dealt with explicitly.
October 1989	X11/SC8/89-3	MDC#43 (Andover), formal proposal for MDC, subcommittee #8. Proposal accepted as a type B proposal of subcommittee #8.
June 1989	...	MDCC-E#24 (Barcelona), extrinsic function approach approved. Proposal to be split up into two proposals to be submitted to MDC SC#8 in the document-format for that subcommittee.
March 1989	...	MDCC-E#23 (Frankfurt), re-evaluation of current proposal. Two separate functionalities found for proposed new function. Slight preference for extrinsic versus intrinsic approach.
November 1988		X11/SC1/89-30MDC#40 (Washington DC), formal proposal
February 1988	...	MDCCE#20 (Utrecht), ordering of diacritical signs resolved, proposal made simpler
January 1988	X11/SC1/88-20	MDC#37 (San Diego), first attempt for a formal proposal, problems with ordering of diacritical signs
November 1987		...MDCCE#19 (Vienna), further discussion, creation European task-group
November 1987		...MDC#36a (Washington DC), separate initial proposals by Richard Walters (MDC) and Alfons Puig (MDCC-E)

1.5. Dependencies

MUMPS Library Specification.

2. Justification of Proposed Change

2.1. Needs

While dealing with other languages than English, the user of a computer encounters discrepancies between the schemes for:

- encoding characters
- collating (sorting) strings
- printing text

These discrepancies have been extensively described in X11/SC1/88-20.
These schemes bear a stronger relation to the (human) language and equipment used, than to the programming language. However, the user of the MUMPS programming language needs a tool for the conversion of strings from one representation into the other in order to be able to use them for the various purposes.

The main problem with a functionality like REPLACE or PRODUCE is how to specify which text is to be replaced by which.

Just to bring those recipients up to date, who didn't see the original proposals:
- $TRANSLATE(TEXT,FROM,TO)
  translates characters in TEXT, replacing those characters that occur in FROM by the corresponding characters in TO

One problem with $TRANSLATE is that it translates one character into one (or zero) other characters.
If, for instance, one wishes to replace all occurrences of "ng" in a string with the phonetic symbol "η", this function can not be used.
Of course, the MUMPS code to do this is fairly straightforward:
```
SET OUT=IN FOR  QUIT:OUT'[FROM  DO
. SET OUT=$PIECE(OUT,FROM,1)_TO_$PIECE(OUT,FROM,1,$LENGTH(OUT))
. QUIT
```
but this would have to be done for each substring that is to be replaced.

The proposed library function $%REPLACE^STRING(TEXT,SPEC) would accept a text and an array of replacement-specifications as parameters. The specification-array would look like:

SPEC(FROM)=TO

meaning that all occurrences of FROM would be replaced by TO.

Advantages of this way of specifying the transformation are:
- immediately obvious to a person reading the program
- easy to add or delete a specific transformation to or from the array

The main disadvantage of this approach is
- current restrictions on subscript affect the set of values that FROM may have, especially: control-characters can not be translated

Of course, there are other possibilities to specify sets of FROM/TO values:
- SPEC(1,1)=FROM1
  SPEC(1,2)=TO1
  SPEC(2,1)=FROM2
  SPEC(2,2)=TO2
  ...
- SPEC(1)=FROM1_separator_TO1
  SPEC(2)=FROM2_separator_TO2
  ...

Finally, there is the problem of 'order'.
Suppose that we were REPLACEing:

SPEC("ab")="PQ"
SPEC("abc")="XYZ"

The original proposal for $%REPLACE^STRING made the function walk through the SPEC array using $ORDER(SPEC(FROM),-1), so that the "abc" would be found before the "ab". At the time it was thought that this approach took care of overlapping occurrences in such a way that 'the most complex' would be found first. This appears to be not completely true, but later it was agreed that for lack of a complete solution to this problem, the $ORDER(...,-1) could be regarded as the 'closest' we could ever get.

One advantage of the SPEC(1,1)=... and the SPEC(1)=FROM_x_TO approaches is that there is a pre-established order, which may be used to establish a rule of precedence for the replacement of overlapping substrings, e.g.:

```
SPEC(1,1)="abcab"
SPEC(1,2)="111"
SPEC(2,1)="ab"
SPEC(2,2)="222"
SPEC(3,1)="abcde"
SPEC(3,2)="333"
```

Whatever way we choose to proceed with these functions, the following needs to be kept in mind:

- eventually we will standardize on a characterset that involves more than 128 characters.
- Current limitations make the SPEC(FROM)=TO approach undesirable, since this would dis-allow replacement of control-characters in portable code.
- Codes that represent printable characters in one character-set, may represent control characters in another (e.g. $CHAR(128) is capital C with cedilla in MS-DOS, but a control-character in ISO-10646). This phenomenon would make it difficult to write a program that converts a text that was written in MS-DOS code to be processed in an environment that uses ISO-10646 code.

Looking back at matters, the author thinks that the SPEC(i)=from_separator_to approach and the SPEC(1,1)=FROM approach tackle two problems:

- it resolves the control-character problem without attacking the subscript restrictions
- it resolves the precedence problem in the case of overlapping substrings

The SPEC(i)=from_separator_to approach creates a problem regarding the separator to choose. An array may be constructed using one separator, and later additions may turn that choice invalid.
Hence, this proposal expresses a preference for the SPEC(1,1)=FROM approach.

It still may seem to be awkward to add or delete specific entries once the array has been created. In practice, such modifications do not seem to be required so often that it is a problem to explicitly renumber the elements.

2.2. Existing Practice in Area of the Proposed Change

Currently no standardized technique for the required functionality exists.

3. Description of the proposed change

3.1. General Description of the Proposed Change

This proposal is one of a series defining two new library-functions to be used for the replacement of substrings within strings. A difference with the currently available intrinsic function $TRANSLATE is that these functions will be able to change "one or more characters" into "zero or more characters", whereas $TRANSLATE is only able to change "one character" into "zero or one character". The "translation-specification" can be of arbitrary length and is stored in an array rather than in one single value.

3.2. Annotated Examples of Use[1]

The MUMPS program lines

```
 SET IN="abcdefghijklmnopqrstuvwxyz"
SET OUT="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  SET RESULT=$TRANSLATE(X,IN,OUT)
```

would create the specification for the translation of lower-case into upper-case characters in ASCII-code.

```
 SET IN="αβγδεζηθικλμνξοπρστυφχψω"
SET OUT="ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ"
```

could be used for the same purpose with the Greek alphabet.

When matching patterns we are not so much interested in the actual characters themselves as in the groups of characters that are to be discerned.

[1] Since the library functions PRODUCE and REPLACE are so strongly related, this document contains examples for both the PRODUCE and the REPLACE functions, and highlights the differences. The 'annotated examples' sections in the proposal for the REPLACE function refers to this document.

If one would want all digits plus some relevant extra characters to match as "numerics" one could use a function-call like:

```
   $TRANSLATE(string,"0123456789+-E.","00000000000000")?.N
```

which would indeed yield a true value for a string equal to "+1.23E45".
In the same way one could use this function on other languages:

```
 SET INL="αβγδεζηθικλμνξοπρστυφχψω"
SET INU="ΑΒΓΔΕΖΗΘΙΚΛΜΝΞΟΠΡΣΤΥΦΧΨΩ"
 SET OUTL="aaaaaaaaaaaaaaaaaaaaaaaa"
 SET OUTU="AAAAAAAAAAAAAAAAAAAAAAAA"
 $TRANSLATE(X,INL_INU,OUTL_OUTU)?...
```

would correctly match any pattern on Greek characters as far as being upper-case or lower-case is concerned.

But now the new functions:

The first function (***described in another proposal***) works more or less analoguously to $TRANSLATE and will be called $%REPLACE^STRING.

Suppose that it is necessary to print German umlauted characters on a printer that is not able to produce such characters, in such a case the Umlaut on the character is usually replaced by the vowel "e" following the character. Hence:

```
             KILL SPEC
SET SPEC(1,1)="ä",SPEC(1,2)="ae"
SET SPEC(2,1)="ö",SPEC(2,2)="oe"
SET SPEC(3,1)="ü",SPEC(3,2)="ue"
SET SPEC(4,1)="Ä",SPEC(4,2)="AE"
SET SPEC(5,1)="Ö",SPEC(5,2)="OE"
SET SPEC(6,1)="Ü",SPEC(6,2)="UE"
 WRITE $%REPLACE^STRING(X,.SPEC)
```

would write the text in X, with all umlauted characters translated into two characters.

The same function could be used for simplifying output-strings containing device-dependent functions:

```
                       KILL SPEC
  SET SPEC(1,1)="<UP-ARROW>",SPEC(1,2)=$CHAR(27)_"[A"
 SET SPEC(2,1)="<DOWN-ARROW>",SPEC(2,2)=$CHAR(27)_"[B"
SET SPEC(3,1)="<RIGHT-ARROW>",SPEC(3,2)=$CHAR(27)_"[C"
 SET SPEC(4,1)="<LEFT-ARROW>",SPEC(4,2)=$CHAR(27)_"[D"
     SET SPEC(5,1)="<ESCAPE>",SPEC(5,2)=$CHAR(27)
   SET SPEC(6,1)="<LINE-FEED>",SPEC(6,2)=$CHAR(10)
         SET X=$%REPLACE^STRING(INPUT,.SPEC)
```

would produce a value in X with some control-sequences translated into a form fit for people to understand.

The difference between REPLACE and PRODUCE is that PRODUCE operates more like a set of "production-rules" and REPLACE is more a straightforward translation.

```
                       KILL SPEC
            SET SPEC(1,1)="aa",SPEC(1,2)="a"
        SET SPEC(2,1)="pqr",SPEC(2,2)="alabama"
             SET SPEC(3,1)="b",SPEC(3,2)=""
                SET X="aaaaaaapqraaaaaaa"
   $%REPLACE^STRING(X,.SPEC) returns "aaaaalabamaaaaa"
        $%PRODUCE^STRING(X,.SPEC) returns "alama"
```

The results are explained as follows (blanks inserted for clarity):

```
REPLACE:          aa aa aa a pqr aa aa aa a
                  a a a a alabama a a a a

PRODUCE repeats:  aa aa ala b ama aa aa
                  a a ala ama a a
                  aa al aa m aa a
                  a al a m a a
                  aa lam aa
                  a lam a
```

The proposed library element may be used for exhaustive replacement of substrings. The function could be used to reduce multiple separators to single ones:

```
KILL SPEC
SET SPEC(1,1)="␣␣",SPEC(1,2)="␣" ; two blanks will be reduced to one
; translate all relevant separators into blanks
SET X=$TRANSLATE(X,"!?,.;:()","␣␣␣␣␣␣␣␣")
; and reduce multiple blanks to singletons
SET X=$%PRODUCE^STRING(X,.SPEC)
```

A more practical example of this functionality would be found in the implementation of soundex algorithms. E.g. a soundex for English names:

1. Remove non-alpha and convert lower case to upper case
2. Examine first letters
   PH → F
   CE → S
   KN → N
   WR → R
   C → K

3. Ignore all other vowels (A, E, I, O and U)
4. Then replace
   - MP → M
   - ST → S
   - DG → D
   - TCH → CH
   - GHT → HT
5. Then make the following equivalent
   - B, F, P and V
   - C, G, J, K, Q, S, X and Z
   - D and T
   - M and N
6. Finally, remove multiple consecutive occurences of the same character

In current-day MUMPS code:

```
SOUNDEX(A)  NEW B,UP,LOW
            SET UP="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
            SET LOW="abcdefghijklmnopqrstuvwxyz"
            SET A=$TRANSLATE(A,LOW_"-., ()'",UP)
            SET B=$EXTRACT(A,1,2),C=$FIND("PH.CE.KN.WR",B)
            IF C SET $EXTRACT(A,1,2)=$EXTRACT("FSNR",C\3)
            IF $EXTRACT(A,1)="C" SET $EXTRACT(A,1,2)="K"
            SET A=$TRANSLATE(A,"AEIOUY")
            DO RE("MP","M")
            DO RE("ST","S")
            DO RE("DG","D")
            DO RE("TCH","CH")
            DO RE("GHT","HT")
            SET A=$TRANSLATE(A,"BFPVCGJKQSXZDTMN","BBBBCCCCCCCCDDMM")
            FOR B=$LENGTH(A):-1:1 DO
            . IF $EXTRACT(A,B)=$EXTRACT(A,B-1) SET $EXTRACT(A,B)=""
            . QUIT
            QUIT A
            ;
RE(b,c)     FOR  QUIT:A'[b  SET A=$PIECE(A,b)_c_$PIECE(A,b,2,999)
            QUIT
```

Using the REPLACE function

```
SOUNDEX(A)  NEW B,C,UP,LOW
            SET UP="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
            SET LOW="abcdefghijklmnopqrstuvwxyz"
            SET A=$TRANSLATE(A,LOW_"-., ()'",UP)
            SET C(1,1)="PH",C(1,2)="F"
            SET C(2,1)="CE",C(2,2)="S"
            SET C(3,1)="C",C(3,2)="K"
            SET C(4,1)="KN",C(4,2)="N"
            SET C(5,1)="WR",C(5,2)="R"
            SET $EXTRACT(A,1,2)=$%REPLACE^STRING($EXTRACT(A,1,2),.C)
            SET A=$TRANSLATE(A,"AEIOUY")
```

```
              SET C(1,1)="MP",C(1,2)="M"
              SET C(2,1)="ST",C(2,2)="S"
              SET C(3,1)="DG",C(3,2)="D"
              SET C(4,1)="TCH",C(4,2)="CH"
              SET C(5,1)="GHT",C(5,2)="HT"
              SET A=$%REPLACE^STRING(A,.C)
              SET A=$TRANSLATE(A,"BFPVCGJKQSXZDTMN","BBBBCCCCCCCCDDMM")
              FOR B=$LENGTH(A):-1:1 DO
              . IF $EXTRACT(A,B)=$EXTRACT(A,B-1) SET $EXTRACT(A,B)=""
              . QUIT
              QUIT A
```

And using the PRODUCE function:

```
SOUNDEX(A)    NEW B,C,UP,LOW,I
              SET UP="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
              SET LOW="abcdefghijklmnopqrstuvwxyz"
              SET A=$TRANSLATE(A,LOW_"-., ()'",UP)
              FOR I=1:1:5 SET C(I,1)=$PIECE("PH C CE KN WR"," ",I)
              FOR I=1:1:5 SET C(I,2)=$PIECE("F. K S. N. R."," ",I)
              SET $EXTRACT(A,1,2)=$%REPLACE^STRING($EXTRACT(A,1,2),.C)
              SET A=$TRANSLATE(A,"AEIOUY.")
              FOR I=1:1:5 SET C(I,1)=$PIECE("MP ST DG TCH GHT"," ",I)
              FOR I=1:1:5 SET C(I,2)=$PIECE("M. S. D. CH. HT." ",I)
              SET A=$%REPLACE^STRING(A,.C)
              SET A=$TRANSLATE(A,"BFPVCGJKQSXZDTMN.","BBBBCCCCCCCCDDMM")
              FOR I=1:1:26 SET C(I,1)=$CHAR(64+I,64+I),C(I,2)=$CHAR(64+I)
              SET A=$%PRODUCE^STRING(A,.C)
              QUIT A
```

It is possible that a translation-specification like

```
SPEC(I,1)="x",SPEC(I,2)="x"
```

would be defined in array `SPEC`. In such a case, the function will loop infinitely.
Since this problem is caused by bad programming and really not different from a programming error like

```
FOR I=1:1:10 SET I=5
```

this type of error need not be dealt with in this proposal.
Using the optional third parameter, though, a limit can be established for the number of iterations that would be peformed.

3.3. Formalization

In the MUMPS Library Specification, add the definition of the PRODUCE library function (at the time that this document is written, no document number or section numbering is known for the MUMPS Library Specification).

3.3.1. Library Element Description

String handling function; substring replacement.

The function scans a string for the occurrence of certain substrings and replace all such occurrences by another substring. This process is repeated until none of these substrings can be found anymore. The resulting string will be passed back to the caller as the function-value.

The function has two required parameters, a string-value (IN) and a translation-specification array (SPEC).
The function converts the value of `IN`, according to the specification in `SPEC`. If the optional parameter `MAX` is specified, its value is used as the maximum number of iterations allowed to perform the intended conversion.

For the purpose of this discussion, the string-value will be called `IN` and the translation-specifications will be called `SPEC(I,1)=FIND`, `SPEC(I,2)=OUT`, `FIND` being a substring to be located and `OUT` being the substring to be put in its place.

For each element of the form `SPEC(I,1)`, the function will scan whether `IN` contains the substring `FIND`. If such a substring occurs, it is replaced by `OUT`, which is the value of `SPEC(I,2)`. After the replacement has been made, `IN` is scanned again for the occurrence of `FIND`. This process continues until the substring is no longer found. When `IN` no longer contains any instance of `FIND`, the next translation-specification is tried.

NOTE: The array `SPEC` may contain overlapping find-strings, e.g. `SPEC(1,1)="ABCDE"` and `SPEC(2,1)="ABC"`. Since the array `SPEC` is scanned using $ORDER on the first subscript, the longer substring will be replaced, before the shorter one is attempted. If the opposite behaviour is required, the order of the values in `SPEC` should be reversed: `SPEC(1,1)="ABC"` and `SPEC(2,1)="ABCDE"`.

Since any translation may cause a substring to be translated to be inserted again, the above process will be repeated until for no specification in the array `SPEC` a matching substring could be found.

3.3.2. Definition
`PRODUCE^STRING(IN,.SPEC,MAX:INTEGER:0)`

This function is computationally equivalent to:

```
PRODUCE(IN,SPEC,MAX) ;
      NEW VALUE,AGAIN,P1,P2,I,COUNT
      SET VALUE=IN,COUNT=0
      FOR  DO  QUIT:'AGAIN
      . SET AGAIN=0
      . SET I=""
      . FOR  SET I=$ORDER(SPEC(I)) QUIT:I=""  DO  QUIT:COUNT<0
      . . QUIT:$GET(SPEC(I,1))=""
      . . QUIT:'($DATA(SPEC(I,2))#2)
      . . FOR  QUIT:VALUE'[SPEC(I,1)  DO  QUIT:COUNT<0
      . . . SET P1=$PIECE(VALUE,SPEC(I,1),1)
      . . . SET P2=$PIECE(VALUE,SPEC(I,1),2,$LENGTH(VALUE))
      . . . SET VALUE=P1_SPEC(I,2)_P2,AGAIN=1
      . . . SET COUNT=COUNT+1
      . . . IF $DATA(MAX),COUNT>MAX SET COUNT=-1,AGAIN=0
      . . . QUIT
      . . QUIT
      . QUIT
      QUIT VALUE
```

3.3.3. Domain
Subscripts in the array SPEC must conform to the portability requirement on subscripts.

3.3.4. Range
Standard.

3.3.5. Side Effects
None.

3.3.6. MUMPS code to implement
See 3.3.2.

4. Implementation impacts

4.1. Impact on Existing User Practices and Investments
Since no standardized technique for this purpose currently exists, no upward compatibility issues exist.

4.2. Impact on Existing Vendor Practices and Investments
None.

4.3. Techniques and Costs for Compliance Verification
Create the following program:

```
      KILL SPEC
      SET SPEC(1,1)="aa",SPEC(1,2)="a"
      SET SPEC(2,1)="pqr",SPEC(2,2)="alabama"
      SET SPEC(3,1)="b",SPEC(3,2)=""
      SET X="aaaaaaapqraaaaaaa"
      WRITE !,$%PRODUCE^STRING(X,.SPEC)
      QUIT
```

An implementation that implements the function correctly would print `"alama"`.

4.4. Legal considerations
None.

5. Closely related standards activities

5.1. Other X11 Proposals (Type A or Type B) Under Consideration
The functions PRODUCE and REPLACE are very strongly related. Their functionality is sufficiently different to make them different functions, though.

5.2. Other Related Standards Efforts
None.

5.3. Recommendations for Co-ordinating Liaison
None.

6. List of Associated Documents

X11/SC1/88-20: Natural language handling.

7. Issues, Pros and Cons, and Discussion

7.1. June 1993, Washington DC
The only con raised was that the Library Specification Document is not yet available.

7.2. October 1993, Dublin Ireland
Formal vote postponed.

7.3. February 1994, Houston Texas
No cons raised.

7.4. June 1994, Reno Nevada
No cons raised.

7.5. January 1995, Albuquerque New Mexico
No cons raised. Formalism re-organized. Descriptive text moved to 3.3.1; M[UMPS] code moved to 3.3.2. Changed all occurrences of $& to $%. Added a footnote that explains why examples for the REPLACE function occur in this document.

7.6. June 1995, Chicago Illinois
Pro: 1. Useful functionality; 2. General solution to many problems; 3. Almost no cost to implementors.
Con: 1: Rarely required, best done by application [4]; 2. Poor cost/benefit for community [1]