CA-MetaCOBOL™ +

String Manipulation Language

Release 1.1

R203M+11DRP

-- PROPRIETARY AND CONFIDENTIAL INFORMATION --

This material contains, and is part of a computer software program which is, proprietary and confidential information owned by Computer Associates International, Inc. The program, including this material, may not be duplicated, disclosed or reproduced in whole or in part for any purpose without the express written authorization of Computer Associates. All authorized reproductions must be marked with this legend.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the United States Government ("the Government") is subject to restrictions as set forth in A) subparagraph (c) (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and/or B) subparagraphs (c) (1) (ii) of the Rights in Technical Data and Computer Software clause of DFAR 252.227-7013. This software is distributed to the Government by:

Computer Associates International, Inc.
One Computer Associates Plaza
Islandia, NY 11788-7000

Unpublished copyrighted work - all rights reserved under the copyright laws of the United States.

This material may be reproduced by or for the United States Government pursuant to the copyright license under the clause at DFAR 252.227-7013 (October 1988).

Release 1.1, January, 1992

Copyright © 1992 Computer Associates International, Inc.
All rights reserved.

CA DATACOM/DB® is a registered trademark of Computer Associates, Inc.
CA-MetaCOBOL™ + is a trademark of Computer Associates, Inc.

All product names referenced herein are trademarks of their respective companies.

Contents

1. About This Manual

This manual provides the description, syntax, code samples, and other supporting information for all CA-MetaCOBOL+ String Manipulation Language statements.

1.1 Purpose

The CA-MetaCOBOL+ String Manipulation Language (SML) is a COBOL language extension that provides string handling and inspection capability unavailable in COBOL. Like other CA-MetaCOBOL+ programs, programs written with the SML are translated through the CA-MetaCOBOL+ Translator. The CA-MetaCOBOL+ Translator produces conventional COBOL programs.

1.2 Organization

The *String Manipulation Reference* is organized as follows:

Chapter	Description
1	Discusses the purpose of the manual, gives a list of CA-MetaCOBOL+ documentation, and explains notation conventions for CA-MetaCOBOL+.
2	Gives an overview of the String Manipulation Language.
3	Provides descriptions, functions, and syntax for the SML Statements.
4	Explains RETURN-CODES and Function Pointers.
5	Provides examples of SML Statements before and after translation.

1.3 Publications

This manual assumes familiarity with the COBOL language and working knowledge of CA-MetaCOBOL+. You may want to refer to the following documentation supplied with CA-MetaCOBOL+. All manuals are updated as required. Instructions accompany each update packet.

Title	Contents
Introduction to CA-MetaCOBOL+	Introduces the CA-MetaCOBOL+ Work Bench, Structured Programming Facility, Quality Assurance Facility, CA DATACOM/DB Facility, Macro Facility, Panel Definition Facility, the Online Programming Language and the Chart Writer Language.
Installation Guide - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment.
CA ACTIVATOR Installation Supplement - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment using CA ACTIVATOR
Installation Guide - VSE	Explains how to install CA-MetaCOBOL+ in the VSE environment.
Installation Guide - CMS	Explains how to install CA-MetaCOBOL+ in the VM environment.
Structured Programming Guide	Introduces the Structured Programming Facility. Includes information on creating, testing, and maintaining structured programs.
Macro Facility Tutorial	Introduces the Macro Facility. Includes information on writing basic macros, model programming, macro writing techniques, and debugging.
Macro Facility Reference	Includes detailed information on the program flow of the CA-MetaCOBOL+ macro translator, macro format, definition of comments, macro nesting, macro prototypes, symbolic words, and model programming.
Quality Assurance Guide	Introduces the Quality Assurance Facility. Includes all the necessary information to perform quality assurance testing on COBOL source programs.
Program Development Guide CA DATACOM/DB	Includes all the information necessary to develop programs that make full use of the functions and features of the CA DATACOM/DB environment.
Program Development Reference CA DATACOM/DB	Contains all CA DATACOM/DB Facility constructs and statements.
Panel Definition Facility Command Reference	Contains all Panel Definition Facility commands.
Panel Definition Facility User's Guide	Includes all the information necessary to create, edit, duplicate, rename, delete, index, and print panel definitions and members. Also describes how to generate BMS source.

Title	Contents
Introduction to CA-MetaCOBOL+	Introduces the CA-MetaCOBOL+ Work Bench, Structured Programming Facility, Quality Assurance Facility, CA DATACOM/DB Facility, Macro Facility, Panel Definition Facility, the Online Programming Language and the Chart Writer Language.
Installation Guide - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment.
CA ACTIVATOR Installation Supplement - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment using CA ACTIVATOR
Installation Guide - VSE	Explains how to install CA-MetaCOBOL+ in the VSE environment.
Installation Guide - CMS	Explains how to install CA-MetaCOBOL+ in the VM environment.
Online Programming Language Reference	Contains all Online Programming Language statements.
Online Programming Language Guide	Contains further instructions for using the Online Programming Language.
PC User Guide	Explains how to customize, get started, and use CA-MetaCOBOL+/PC. Includes information on keyword expansion, the CA-MetaCOBOL+/PC translator, and CA macro sets and programs.
Program Development Guide CA DATACOM/PC	Describes how to use CA-MetaCOBOL+/PC with CA DATACOM/PC.
String Manipulation Language Guide	Introduces the String Manipulation Language, which provides string handling and inspection capabilities unavailable in COBOL.

All manuals are updated as required. Instructions accompany each update package.

1.4 Notation Conventions

The following conventions are used in the command formats throughout this manual:

UPPERCASE	is used to display commands or keywords you must code exactly as shown.
lowercase italic	is used to display information you must supply. For example, DASD space parameters may appear as *xxxxxxx xxxxxxx xxxxxxx*.
Underscores	show a default value.
Brackets [ ]	mean that you can select one of the items enclosed by the brackets; none of the enclosed items is required.
Braces { }	mean that you must select one of the items enclosed by the braces.
Vertical Bar |	separates options. One vertical bar separates two options, two vertical bars separate three options, and so on. You must select one of the options.
Ellipsis . . .	means that you can repeat the word or clause that immediately precedes the ellipsis.

2. How to Get Started

The COBOL String Manipulation Language (COBOL/SML) is an extension to the COBOL programming language providing string handling and inspection capabilities unavailable in COBOL.

The string manipulation capability is provided by a Support Module invoked by the COBOL program, and a set of high-level statements facilitating requests of the Support Module's functions. The program containing COBOL/SML statements is passed through a CA-MetaCOBOL+ translation where these statements are replaced with a series of MOVE and CALL statements. The compiled program may be link-edited with the string manipulation Support Module, or if compiled with the DYNAM option, the Support Module can be dynamically loaded at program execution.

Some of these functions can be achieved using COBOL language statements such as STRING, UNSTRING, INSPECT, and TRANSFORM. Unfortunately, TRANSFORM is a statement discontinued in VS COBOL II. Also, with some COBOL compilers, INSPECT, STRING, and UNSTRING require the use of operating system services, prohibiting their use in CICS programs. The string manipulation Support Module does not require the use of operating system services. In addition, it is re-entrant - all data manipulation takes place in areas supplied by the calling program.

Provides Consistency

The syntax for COBOL string manipulation statements mentioned above is not consistent between any two statements. The COBOL/SML language has the consistency that each statement has a STARTING AT and FOR LENGTH OF clause available to it, making them easier to use than their COBOL counterpart. The presence of these two clauses in each of the COBOL/SML statements allows for the manipulation to be confined to specific parts of fields--a capability not found in all COBOL string manipulation statements.

RETURN-CODES and Function Pointers

RETURN-CODES are issued by the Support Module. A RETURN-CODE of zero indicates that the string manipulation request was performed successfully. The Support Module also sets the Function Pointer if the RETURN-CODE is zero.

2.1 COBOL String Manipulation Language

The COBOL String Manipulation Language (COBOL/SML) provides a high-level statement corresponding to each of the string manipulation functions. These statements simplify the task of invoking the Support Module by providing the WORKING-STORAGE definitions for some calling parameters and initializing the appropriate parameters with function request information. The COBOL/SML statement syntax resembles English, providing the additional benefit of being self-documenting.

The COBOL/SML statements are an adaptation of the COBOL "MOVE" statement (except for the statement providing the UPCASE function). There is no conditional code generated in their expansion, allowing them to be placed anywhere in the PROCEDURE DIVISION without impacting the surrounding logic.

All fields named as either a source or target field for string manipulation or inspection can be qualified by a starting position and/or length. When a source or target field is not qualified by either a starting position or a length, the starting position defaults to 1, and the length defaults to the difference between the starting position and the end of the field.

Note: COBOL/SML does not support fields subordinate to an OCCURS clause nor fields using the "OF" or "IN" qualification.

2.2 The Support Module

The Support module provides the following string manipulation functions:

- Finds the location of a substring within a string
- Moves a piece of a field to another location
- Makes a string by concatenating a series of source fields into the target location
- Finds the location of the first occurrence in the source string of any character found in the target
- Finds the location of the first occurrence in the source string of any character other than those found in the target
- Determines whether the source string conforms to a specified naming convention
- Changes any lower case characters to upper case

The Support Module will perform string manipulation or inspection for a maximum length of 256 bytes. The fields on which the Support Module performs this activity may be defined longer than this 256 byte maximum, but only 256 bytes of data movement or inspection can be done in a single function request. It has run-time checking to prevent it from doing any data manipulation beyond the defined boundaries of the target field. The Support Module treats all source and target fields as alphanumeric regardless of their actual definition.

2.3 Translating COBOL/SML Statements

A sample input to the Translator is shown below. You must supply the options and source program indicated. Refer to the CA-MetaCOBOL+ User Guide for more information. If you are using CA-MetaCOBOL+/PC, refer to the CA-MetaCOBOL+/PC User Guide for equivalent SET statements.

[OPTION options]

```
*$COPY SML          from standard library
*$LIBED SML         from CA LIBRARIAN master
```

source program

2.4 Link-Editing Requirements

The COBOL program using String Manipulation Language functions must be link-edited with the CA supplied object module MCTXSML. The MCTXSML component is provided with CA-MetaCOBOL+ or CA-MetaCOBOL+/PC at installation time.

3. Command Descriptions and Syntax

Functions

The following functions are performed at COBOL-program execution time as a result of the translation of String Manipulation Language statements. Sample SML statements, before and after translation are given in Chapter 5.

LOCATE STRING
Find the location of a substring within a string.

SUBSTRING
Move a piece of a field to another location.

CONCATENATE
Make a string by concatenating a series of source fields into the target location.

LOCATE CHARACTER
Find the location of the first occurrence in the source field of any character found in the target.

LOCATE ABSENCE OF CHARACTER
Find the location of the first occurrence in the source field of any character other than those found in the target.

CONFORM
Determine whether the source string conforms to a specific naming convention. Choices are COBOL data-name and PDS member name.

UPCASE
Change any lowercase characters to uppercase characters.

Functional Groupings

LOCATE STRING, SUBSTRING, and CONCATENATE are "string" oriented functions in that each recognizes contiguous strings of characters in the fields.

LOCATE CHARACTER, LOCATE ABSENCE OF CHARACTER, CONFORM, and UPCASE are "character" oriented functions; each recognizes characters regardless of the character's position within the fields.

Unlike the COBOL MOVE statement, which pads a target field with blanks when the source is shorter, the SUBSTRING and CONCATENATE functions will not affect data in the target field preceding or succeeding the designated limits of the operation.

This page is intentionally blank so that the command syntax diagrams and their explanations appear on facing pages.

3.1 LOCATE STRING Function

The LOCATE STRING function finds the location of a substring within a string. The COBOL/SML statement providing the LOCATE STRING function is:

```
MOVE  POSITION  FOUND  IN  { data-name-1    }
                           { alpha-literal-1 }

                              [ STARTING AT  { data-name-2 } ]
                                             { num-literal-1 }

                              [ FOR LENGTH OF { data-name-3 } ]
                                              { num-literal-2 }

                          OF  STRING

                              { data-name-4    }
                              { alpha-literal-2 }

                              [ STARTING  AT { data-name-5 } ]
                                             { num-literal-3 }

                              [ FOR LENGTH OF { data-name-6 } ]
                                              { num-literal-4 }

                           TO  data-name-7
```

data-name-x

is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2, 3, and 5-7 must be numeric. Data-names 1 and 4 must be alphanumeric.

dataname-7

is the dataname into which the Function Pointer is to be moved.

alpha-literal-x

is a character string bounded by apostrophes. The character string may include any character in the computer's character set. To represent a single quotation mark character within an alphanumeric literal, you must use two contiguous apostrophes.

num-literal-x

is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

RETURN-CODE

If the RETURN-CODE is not equal to zero, the request has been rejected. Refer to Chapter 4 for information about RETURN-CODES.

Function Pointer

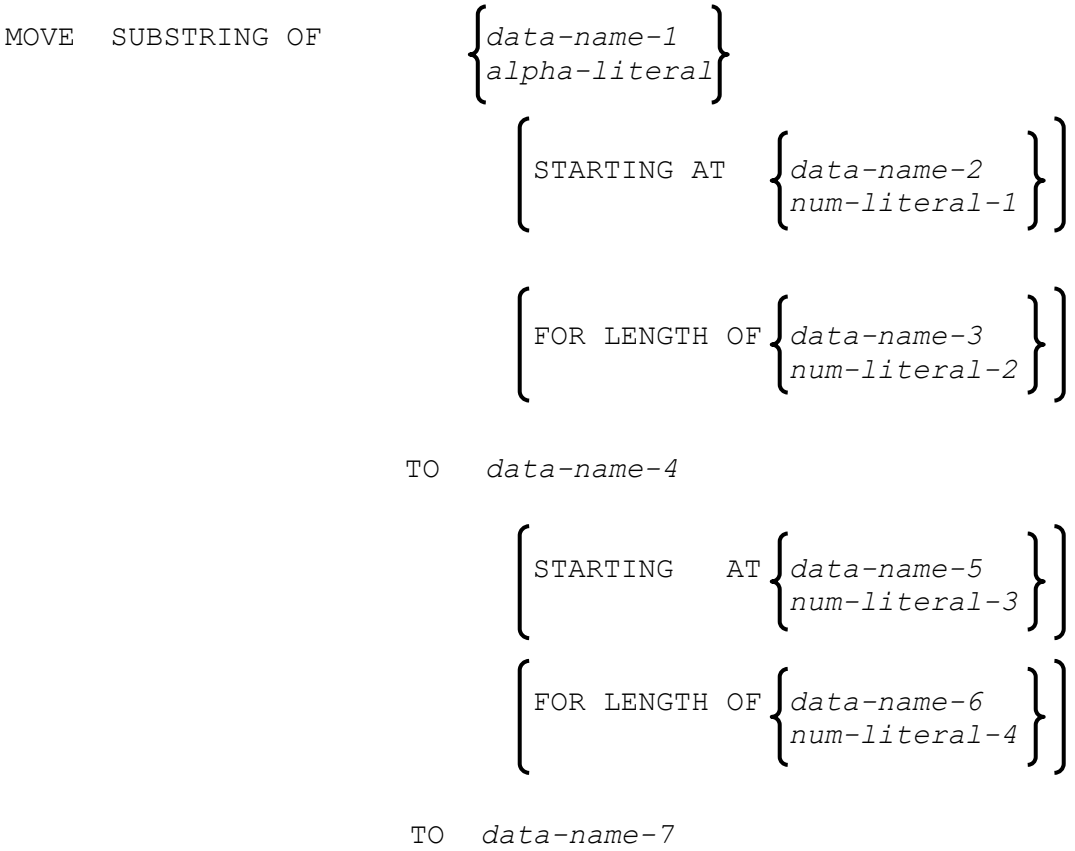
The Function Pointer is a dataname defined by the SML Language with the name MCT-FNCT-FUNCTION-POINTER. It is set only when the RETURN-CODE for the requested function is zero. For LOCATE STRING, the Function Pointer value provides the following information:

0	substring not found in larger string.
n	substring begins at position n in larger string.

For more information about Function Pointers, refer to Section 4.3, "Function Pointers."

3.2 SUBSTRING Function

The SUBSTRING Function moves a piece of a field to another location.



data-name-x
Is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2, 3, 5, and 6 must be numeric. Data-name 4 must be alphanumeric.

alpha-literal
Is a character string bounded by apostrophes. The character string may include any character in the computer's character set. To represent a single quotation mark character within an alphanumeric literal, you must use two contiguous apostrophes.

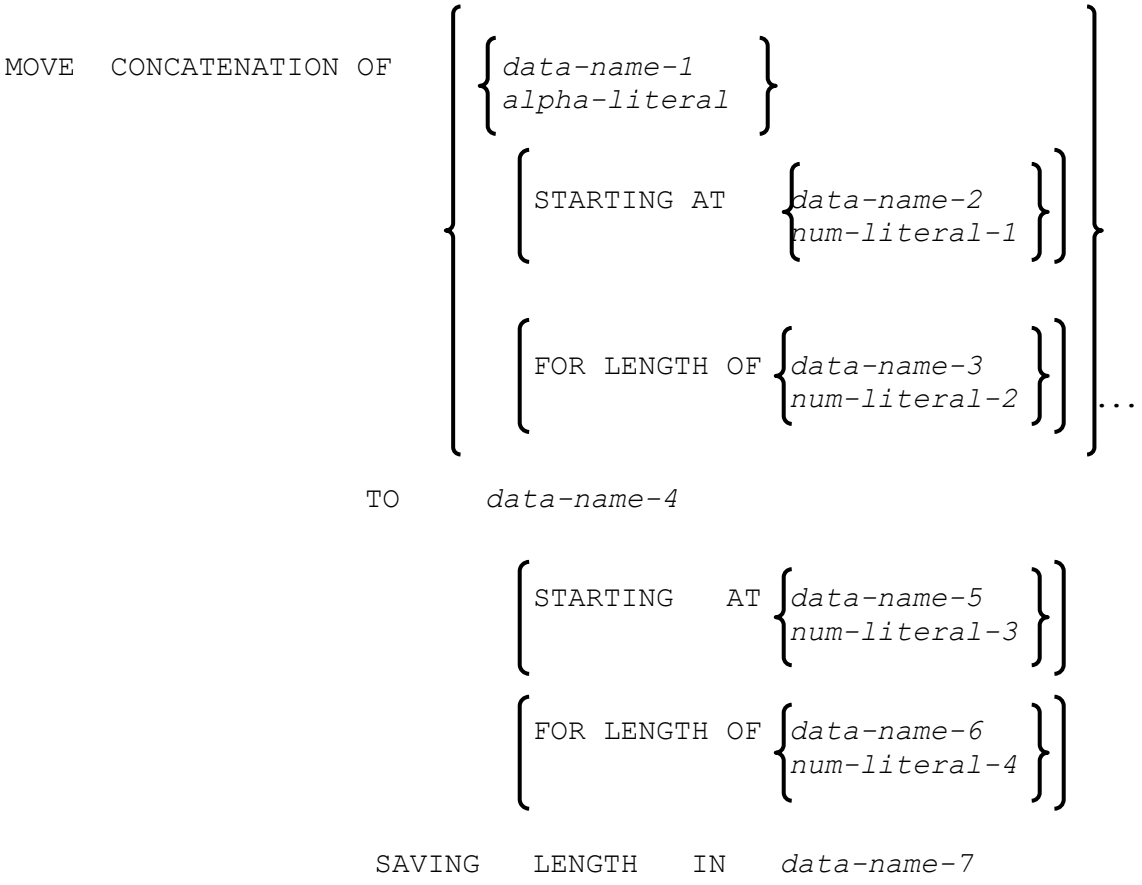
num-literal-x
Is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

RETURN-CODE
If the RETURN-CODE is not equal to zero, the request has been rejected. Refer to Chapter 4 for information about RETURN-CODES.

Function Pointer
The SUBSTRING Function does not have a Function Pointer.

3.3 CONCATENATE Function

The CONCATENATE Function makes a string by concatenating a series of source fields into the target location. The COBOL/SML statement providing the CONCATENATE function:



data-name-X

Is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2, 3, and 5-7 must be numeric. Data-names 1 and 4 must be alphanumeric.

dataname-7

is the dataname into which the Function Pointer is to be moved.

alpha-literal

Is a character string bounded by apostrophes. The character string may include any character in the computer's character set. To represent a single quotation mark character within an alphanumeric literal, you must use two contiguous apostrophes.

num-literal-x

Is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

Note: Up to four source fields may be specified for one concatenation function.

RETURN-CODE

If the RETURN-CODE is not equal to zero, the request has been rejected. Refer to Chapter 4 for information about RETURN-CODES.

Function Pointer

The Function Pointer is a dataname defined by the SML Language with the name MCT-FNCT-FUNCTION-POINTER. It is set only when the RETURN-CODE for the requested function is zero. For CONCATENATE, the Function Pointer value provides the following information:

n position of last target character modified relative to the first defined byte of the target

For more information about Function Pointers, refer to Section 4.3, "Function Pointers.

3.4 LOCATE ABSENCE OF CHARACTER Function

The LOCATE ABSENCE OF CHARACTER Function finds the location of the first occurrence in the source field of any character other than those found in the target. The COBOL/SML statement providing the LOCATE ABSENCE OF CHARACTER function:

```
MOVE  POSITION  FOUND  IN  { data-name-1    }
                           { alpha-literal-1 }

                              [ STARTING AT    { data-name-2 } ]
                              [                { num-literal-1 } ]

                              [ FOR LENGTH OF  { data-name-3 } ]
                              [                { num-literal-2 } ]

                        OF    { CHARACTERS } NOT
                              { CHARACTER  }

                              { data-name-4    }
                              { alpha-literal-2 }

                              [ STARTING   AT  { data-name-5 } ]
                              [                { num-literal-3 } ]

                              [ FOR LENGTH OF  { data-name-6 } ]
                              [                { num-literal-4 } ]

                       SAVING   LENGTH   IN   data-name-7
```

Note: The syntactical difference between the LOCATE ABSENCE OF CHARACTER and LOCATE CHARACTER functions is the presence or absence of the word "NOT".

data-name-x

Is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2, 3, and 5-7 must be numeric. Data-names 1 and 4 must be alphanumeric.

dataname-7

is the dataname into which the Function Pointer is to be moved.

alpha-literal

Is a character string bounded by apostrophes. The character string may include any character in the computer's character set. To represent a single quotation mark character within an alphanumeric literal, you must use two contiguous apostrophes.

num-literal-x

Is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

CHARACTERS
CHARACTER

Enter either CHARACTER or CHARACTERS. One of these must be entered. Using the proper singular or plural semantics is not required here.

RETURN-CODE

If the RETURN-CODE is not equal to zero, the request has been rejected. Refer to Chapter 4 for information about RETURN-CODES.

Function Pointer

The Function Pointer is a dataname defined by the SML Language with the name MCT-FNCT-FUNCTION-POINTER. It is set only when the RETURN-CODE for the requested function is zero. For LOCATE ABSENCE Of CHARACTER, the Function Pointer value provides the following information:

0	all characters in source found somewhere in target.
n	position of first source character not found in target.

For more information about Function Pointers, refer to Section 4.3, "Function Pointers."

3.5 LOCATE CHARACTER Function

The LOCATE CHARACTER Function finds the location of the first occurrence in the source field of any character found in the target. The COBOL/SML statement providing the LOCATE CHARACTER function:

```
MOVE  POSITION  FOUND  IN  { data-name-1   }
                           { alpha-literal-1}

                              [ STARTING AT   { data-name-2  } ]
                              [               { num-literal-1} ]

                              [ FOR LENGTH OF { data-name-3  } ]
                              [               { num-literal-2} ]

                        OF    { CHARACTERS }
                              { CHARACTER  }

                              { data-name-4   }
                              { alpha-literal-2}

                              [ STARTING   AT { data-name-5  } ]
                              [               { num-literal-3} ]

                              [ FOR LENGTH OF { data-name-6  } ]
                              [               { num-literal-4} ]

                         TO   data-name-7
```

Note: The syntactical difference between the LOCATE ABSENCE OF CHARACTER and LOCATE CHARACTER functions is the presence or absence of the word "NOT".

data-name-x

Is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2, 3, and 5-7 must be numeric. Data-names 1 and 4 must be alphanumeric.

dataname-7

is the dataname into which the Function Pointer is to be moved.

alpha-literal

Is a character string bounded by apostrophes. The character string may include any character in the computer's character set. To represent a single quotation mark character within an alphanumeric literal, you must use two contiguous apostrophes.

num-literal-x

Is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

CHARACTERS
CHARACTER

Enter either CHARACTER or CHARACTERS. One of these must be entered. Using the proper singular or plural semantics is not required here.

RETURN-CODE

If the RETURN-CODE is not equal to zero, the request has been rejected. Refer to Chapter 4 for information about RETURN-CODES.

Function Pointer

The Function Pointer is a dataname defined by the SML Language with the name MCT-FNCT-FUNCTION-POINTER. It is set only when the RETURN-CODE for the requested function is zero. For LOCATE CHARACTER, the Function Pointer value provides the following information:

0	no characters in source found anywhere in target.
n	position of first source character also found in target.

For more information about Function Pointers, refer to Section 4.3, "Function Pointers."

3.6 CONFORM Function

The CONFORM Function determines whether the source string conforms to a specific naming convention. Choices are COBOL data-name and PDS member name. The COBOL/SML statement providing the CONFORM function:

```
MOVE  POSITION  FOUND  IN   data-name-1

                              [ STARTING AT   { data-name-2   } ]
                              [               { num-literal-1 } ]

                              [ FOR LENGTH OF { data-name-3   } ]
                              [               { num-literal-2 } ]

                        OF    { CHARACTERS } { COBOL  }
                              { CHARACTER  } { MEMBER }

                         TO   data-name-4
```

Note: The CONFORM function will ignore trailing blanks.

data-name-x

Is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2, 3, and 4 must be numeric. Data-name 1 must be alphanumeric.

dataname-4

is the dataname into which the Function Pointer is to be moved.

alpha-literal

Is a character string bounded by apostrophes. The character string may include any character in the computer's character set. To represent a single quotation mark character within an alphanumeric literal, you must use two contiguous apostrophes.

num-literal-x

Is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

CHARACTERS
CHARACTER

Enter either CHARACTER or CHARACTERS. One of these must be entered. Using the proper singular or plural semantics is not required here.

COBOL
MEMBER

Enter one of these to verify conformity to either the COBOL or MEMBER naming convention.

Under COBOL naming conventions, *data-name-1* must contain a word that can be spelled as a 30-character alphanumeric data-name. This data-name must have at least one alphabetic character. It may contain hyphens, but the first and last characters must not be hyphens.

Under MEMBER naming conventions, *data-name-1* must be 1 to 8 alphanumeric characters in length, and the first character must be alphabetic.

If you are checking for conformity to COBOL naming conventions, enter COBOL.

If you are checking for conformity to Member naming conventions, enter MEMBER.

RETURN-CODE

If the RETURN-CODE is not equal to zero, the request has been rejected. Refer to Chapter 4 for information about RETURN-CODES.

Function Pointer

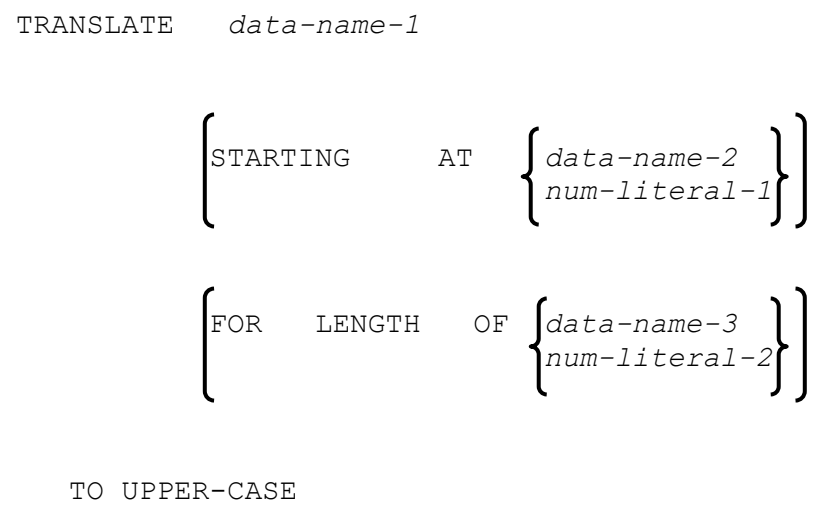
The Function Pointer is a dataname defined by the SML Language with the name MCT-FNCT-FUNCTION-POINTER. It is set only when the RETURN-CODE for the requested function is zero. For CONFORM, the Function Pointer value provides the following information:

0 all source characters conform to specified naming convention.
n position of first source character at which specified naming convention is violated.

For more information about Function Pointers, refer to Section 4.3, "Function Pointers."

3.7 UPCASE Function

The UPCASE Function changes any lowercase characters to uppercase characters. The COBOL/SML statement providing the UPCASE Function:



data-name-x
Is any unqualified data-name in the Data Division. Qualifications will not be accommodated.

In this syntax diagram, data-names 2 and 3 must be numeric. Data-name 1 must be alphanumeric.

num-literal-x
Is a literal composed of one or more numeric characters that may also contain a positive algebraic sign. The positive algebraic sign, if present, must be the leftmost character.

Function Pointer

There is no Function Pointer for the UPCASE Function.

4. RETURN-CODES and Function Pointers

The RETURN-CODE for the invocation of the Support Module is available to the COBOL program in the RETURN-CODE special register.

If the RETURN-CODE is not zero, the Support Module was unable to satisfy the string manipulation request, and no string manipulation was performed. The number assigned to a RETURN-CODE conforms to the Series Codes in the table below.

Series Codes

1000 series
: Indicates run-time problems with the lengths and/or starting positions provided for fields used in the function request.

2000 series
: Designates an invalid number of parameters provided for the specific function requested.

greater than 2000
: Indicates a compile-time problem, or a problem introduced by a ZAP being incorrectly applied at the site.

A table of RETURN-CODES appears on the next page.

4.1 Table of RETURN-CODES

The RETURN-CODES listed below are issued by the Support Module. The following RETURN-CODES apply to all functions: 4001, 3001, 10n1, 10n2, 10n3, 10n4, and 0000.

Code	Reason for Failure
4001	Too many parameters accompanied the function request.
3601	The qualifier of the CONFORM function name is not recognized. (It must be "COBOL" or "MEMBER").
3001	The function name is not recognized.
2701	The UPCASE function request did not specify only one target field.
2601	The CONFORM function request did not specify only one target and only one source field.
2501	The LOCATE CHARACTER function request did not specify only one target and only one source field.
2401	The LOCATE ABSENCE OF CHARACTER function request did not specify only one target and only one source field.
2301	The CONCATENATION function request specified more than four source fields.
2201	The SUBSTRING function request did not specify only one target and only one source field.
2101	The LOCATE STRING Function request did not specify only one target and only one source field.
1101	The LOCATE STRING Function designates a target string longer than the source field in which it is to search.
10n1	The nth field has a requested length less than 1.
10n2	The nth field has a requested length greater than 256.
10n3	The nth field has a requested starting position less than 1.
10n4	The nth field has a requested starting position and requested length which together extend beyond the defined boundaries of the field.
0000	No errors. The function request ran to completion, and a Function Pointer value has been provided to caller.

Proceed to the next page for instructions and an example for checking the RETURN-CODES and fixing errors.

4.2 Interpreting RETURN-CODES 10n1 through 10n4

When you code a String Manipulation Statement using one of the functions of the String Manipulation Language, the CA-MetaCOBOL+ Translator replaces your original MOVE Statement with several MOVE and CALL Statements at translate-time. If an error is detected at runtime, then the CALL Statement will receive a RETURN-CODE other than zero.

Remember to include logic for checking errors based on the RETURN-CODES, as shown in the example below.

For RETURN-CODES 10n1 thru 10n4, to determine the field in error in the CALL Statement, add 4 to n. Then count to the nth field (after the USING Clause) in the CALL Statement. Each field is separated by a space.

Example

Assume we coded the statement below:

```
MOVE ZERO TO BINARY-COUNTER-FIELD

MOVE SUBSTRING OF 62-BYTE-SCRATCH-FIELD
        FOR LENGTH OF BINARY-COUNTER-FIELD
    TO 13-BYTE-SCRATCH-FIELD
        STARTING AT 2.
.
.
.
IF RETURN-CODE IS NOT EQUAL TO ZERO THEN...
```

A CALL Statement and a series of MOVE Statements is generated by the Translator. The CALL Statement following the MOVE Statements is:

```
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS 13-BYTE-SCRATCH-FIELD
   62-BYTE-SCRATCH-FIELD.
```

Assume the RETURN-CODE returned for our MOVE Command was 1021. Remember, the field with an error is determined by adding 4 to n. Here, n = 2. Because 4 + 2 = 6, the sixth field is in error. The sixth field is 62-BYTE-SCRATCH-FIELD. The error in this field is that its length was set to zero, as shown in the original code above.

4.3 Function Pointer

The Function Pointer is a dataname defined by the SML Language with the name MCT-FNCT-FUNCTION-POINTER. It is set only when the RETURN-CODE for the requested function is zero. The Function Pointer value provides the following information:

For LOCATE STRING

- 0 substring not found in larger string.
- n substring begins at position n in larger string.

For CONCATENATION

- n position of last target character modified relative to the first defined byte of the target.

For LOCATE ABSENCE OF CHARACTER

- 0 all characters in source found somewhere in target.
- n position of first source character not found in target.

For LOCATE CHARACTER

- 0 no characters in source found anywhere in target.
- n position of first source character also found in target.

For CONFORM

- 0 all source characters conform to specified naming convention
- n position of first source character at which specified naming convention is violated.

Trailing blanks are ignored in the CONFORM function.

Note: When the Source Field for the CONFORM Function has a length designation greater than target conformity will allow (30 for a COBOL data-name, and 8 for PDS member name), then no conformity checking is done; the violation occurs at the first byte beyond the maximum allowable length. The Function Pointer is set to this position; 31 for COBOL, and 9 for MEMBER.

The activity of the Support Module is restricted to only providing a Function Pointer value when the function name is LOCATE STRING, LOCATE ABSENCE OF CHARACTER, LOCATE CHARACTER, or CONFORM. No data in either the source or target fields is affected for these functions.

5. Expanded COBOL SML Statements

This section contains an example of each expanded COBOL SML Statement. The Identification, Environment, and Data Divisions referenced by each example appear in Section 5.1.

In each example, the original code is marked **Input**. The generated MOVE and CALL Statments are given below the original code and are marked **Output**.

Function Pointer

In each example that has a Function Pointer, the Function Pointer is moved to the target dataname with the following statement:

```
MOVE MCT-FNCT-FUNCTION-POINTER TO dataname-x.
```

5.1 Sample Identification, Environment, and Data Divisions

The divisions below are to be used as the Identification, Environment, and Data Divisions for all examples in this chapter. All data-names referenced in the examples can be found here.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. ABCDEFGH.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  CONSTANTS.
    02 ALPHABETIC-CHARACTERS    PIC X(29) VALUE

'@#$ABCDEFGHIJKLMNOPQRSTUVWXYZ'
    02 BINARY-DIGITS            PIC X(02) VALUE '01'.
    02 OCTAL-DIGITS             PIC X(08) VALUE '01234567'.
    02 HEXIDECIMAL-DIGITS       PIC X(16) VALUE
                                '0123456789ABCDEF'.

01  SCRATCH-AREA.
    02 NUMERIC-FIELDS.
       03 BINARY-COUNTER-FIELD  PIC S9(04) COMP.
       03 PACKED-COUNTER-FIELD  PIC S9(05) COMP-3.
       03 NUMERIC-DISPLAY-COUNTER-FIELD PIC S9(06) DISPLAY.
    02 13-BYTE-SCRATCH-FIELD    PIC X(013).
    02 35-BYTE-SCRATCH-FIELD    PIC X(035).
    02 62-BYTE-SCRATCH-FIELD    PIC X(062).
    02 197-BYTE-SCRATCH-FIELD   PIC X(197).
*
* The remainder of WORKING-STORAGE is generated by COBOL/SML
*
01  MCT-FNCT-WORKAREA.
    02 MCTXSML                  PIC X(08) VALUE 'MCTXSML'.
    02 MCT-FNCT-CALL-PARAMETERS.
       03 MCT-FNCT-TRIPLETS.
          04 MCT-FNCT-POINTERS  OCCURS 6 TIMES.
             05 MCT-FNCT-FIELD-DEFINED-LENGTH PIC S9(4) COMP.
             05 MCT-FNCT-FIELD-REQUESTD-LENGTH PIC S9(4) COMP.
             05 MCT-FNCT-FIELD-START-POSITION PIC S9(4) COMP.
       03 MCT-FNCT-FUNCTION-POINTER PIC S9(4) COMP.
    02 MCT-FNCT-FUNCTION-CODE   PIC X(8).
    02 MCT-FNCT-SCRATCH-PAD     PIC X(256).
```

```
  02 MCT-FNCT-LITERAL-1       PIC X(4) VALUE 'DSN='.
  02 MCT-FNCT-LITERAL-2       PIC X(1) VALUE '('.
  02 MCT-FNCT-LITERAL-3       PIC X(1) VALUE ')'.
```

Note: The fields MCT-FNCT-LITERAL-n are generated by COBOL/SML Statements when a source or target specified in the statement is a literal.

5.1.1 Example 1 - LOCATE ABSENCE OF CHARACTER

Input:

```
MOVE POSITION FOUND IN 62-BYTE-SCRATCH-FIELD
                         STARTING AT
                           NUMERIC-DISPLAY-COUNTER-FIELD
                     OF CHARACTER NOT OCTAL-DIGITS
                     TO PACKED-COUNTER-FIELD.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 8 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (1)
MOVE 8 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (1)
MOVE 62 TO MCT-FNCT-FIELD-DEFINED-LENGTH (2)
MOVE NUMERIC-DISPLAY-COUNTER-FIELD TO
   MCT-FNCT-FIELD-START-POSITION (2)
COMPUTE MCT-FNCT-FIELD-REQUESTD-LENGTH (2) EQUAL 62 -
   NUMERIC-DISPLAY-COUNTER-FIELD + 1
MOVE 'VERIFY' TO MCT-FNCT-FUNCTION-CODE
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS OCTAL-DIGITS 62-BYTE-SCRATCH-FIELD
MOVE MCT-FNCT-FUNCTION-POINTER TO PACKED-COUNTER-FIELD.
```

5.1.2 Example 2 - LOCATE CHARACTER

Input:

```
MOVE POSITION FOUND IN 197-BYTE-SCRATCH-FIELD
                         STARTING AT PACKED-COUNTER-FIELD
                         FOR LENGTH OF BINARY-COUNTER-FIELD
                     OF CHARACTER ALPHABETIC-CHARACTERS
                         STARTING AT 1
                         FOR LENGTH OF 3
                     TO PACKED-COUNTER-FIELD.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 29 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (1)
MOVE 3 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (1)
MOVE 197 TO MCT-FNCT-FIELD-DEFINED-LENGTH (2)
MOVE PACKED-COUNTER-FIELD TO MCT-FNCT-FIELD-START-POSITION
   (2)
MOVE BINARY-COUNTER-FIELD TO MCT-FNCT-FIELD-REQUESTD-LENGTH
   (2)
MOVE 'TRT' TO MCT-FNCT-FUNCTION-CODE
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS ALPHABETIC-CHARACTERS
   197-BYTE-SCRATCH-FIELD
MOVE MCT-FNCT-FUNCTION-POINTER TO PACKED-COUNTER-FIELD.
```

5.1.3 Example 3 - CONFORM

Input:

```
MOVE POSITION FOUND IN 62-BYTE-SCRATCH-FIELD
                    OF CHARACTER NOT COBOL
                    TO PACKED-COUNTER-FIELD.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 'COBOL' TO MCT-FNCT-SCRATCH-PAD
MOVE 256 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (1)
MOVE 256 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (1)
MOVE 62 TO MCT-FNCT-FIELD-DEFINED-LENGTH (2)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (2)
MOVE 62 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (2)
MOVE 'CONFORM' TO MCT-FNCT-FUNCTION-CODE
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS MCT-FNCT-SCRATCH-PAD
   62-BYTE-SCRATCH-FIELD
MOVE MCT-FNCT-FUNCTION-POINTER TO PACKED-COUNTER-FIELD.
```

5.1.4 Example 4 - LOCATE STRING

Input:

```
MOVE POSITION FOUND IN 35-BYTE-SCRATCH-FIELD
                    OF STRING 'DSN='
                    TO PACKED-COUNTER-FIELD.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 4 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (1)
MOVE 4 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (1)
MOVE 35 TO MCT-FNCT-FIELD-DEFINED-LENGTH (2)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (2)
MOVE 35 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (2)
MOVE 'INDEX' TO MCT-FNCT-FUNCTION-CODE
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS MCT-FNCT-LITERAL-1
   35-BYTE-SCRATCH-FIELD
MOVE MCT-FNCT-FUNCTION-POINTER TO PACKED-COUNTER-FIELD.
```

5.1.5 Example 5 - SUBSTRING

Input:

```
MOVE SUBSTRING OF 62-BYTE-SCRATCH-FIELD
                    FOR LENGTH OF BINARY-COUNTER-FIELD
               TO 13-BYTE-SCRATCH-FIELD
                    STARTING AT 2.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 13 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 2 TO MCT-FNCT-FIELD-START-POSITION (1)
COMPUTE MCT-FNCT-FIELD-REQUESTD-LENGTH (1) EQUAL 13 - 2 + 1
MOVE 62 TO MCT-FNCT-FIELD-DEFINED-LENGTH (2)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (2)
MOVE BINARY-COUNTER-FIELD TO MCT-FNCT-FIELD-REQUESTD-LENGTH
   (2)
MOVE 'SUBSTR' TO MCT-FNCT-FUNCTION-CODE
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS 13-BYTE-SCRATCH-FIELD
   62-BYTE-SCRATCH-FIELD.
```

5.1.6 Example 6 - UPCASE

Input:

```
TRANSLATE 62-BYTE-SCRATCH-FIELD TO UPPER-CASE.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 62 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (1)
MOVE 62 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (1)
MOVE 'UPCASE' TO MCT-FNCT-FUNCTION-CODE
CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
   MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
   MCT-FNCT-TRIPLETS 62-BYTE-SCRATCH-FIELD.
```

5.1.7 Example 7 - CONCATENATE

Input:

```
MOVE CONCATENATION OF 197-BYTE-SCRATCH-FIELD
                        STARTING AT PACKED-COUNTER-FIELD
                        FOR LENGTH OF BINARY-COUNTER-FIELD
                     '('
                     35-BYTE-SCRATCH-FIELD
                        STARTING AT
                           NUMERIC-DISPLAY-COUNTER-FIELD
                     ')'
                   TO 62-BYTE-SCRATCH-FIELD
                        STARTING AT 10
                   SAVING LENGTH IN BINARY-COUNTER-FIELD.
```

Output:

```
MOVE LOW-VALUES TO MCT-FNCT-CALL-PARAMETERS
MOVE 62 TO MCT-FNCT-FIELD-DEFINED-LENGTH (1)
MOVE 10 TO MCT-FNCT-FIELD-START-POSITION (1)
COMPUTE MCT-FNCT-FIELD-REQUESTD-LENGTH (1) EQUAL 62 - 10 + 1
MOVE 197 TO MCT-FNCT-FIELD-DEFINED-LENGTH (2)
MOVE PACKED-COUNTER-FIELD TO MCT-FNCT-FIELD-START-POSITION (2)
MOVE BINARY-COUNTER-FIELD TO MCT-FNCT-FIELD-REQUESTD-LENGTH(2)
MOVE 1 TO MCT-FNCT-FIELD-DEFINED-LENGTH (3)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (3)
MOVE 1 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (3)
MOVE 35 TO MCT-FNCT-FIELD-DEFINED-LENGTH (4)
MOVE NUMERIC-DISPLAY-COUNTER-FIELD TO
     MCT-FNCT-FIELD-START-POSITION (4)
COMPUTE MCT-FNCT-FIELD-REQUESTD-LENGTH (4) EQUAL 35 -
   NUMERIC-DISPLAY-COUNTER-FIELD + 1
MOVE 1 TO MCT-FNCT-FIELD-DEFINED-LENGTH (5)
MOVE 1 TO MCT-FNCT-FIELD-START-POSITION (5)
MOVE 1 TO MCT-FNCT-FIELD-REQUESTD-LENGTH (5)
MOVE 'CONCAT' TO MCT-FNCT-FUNCTION-CODE
  CALL 'MCTXSML' USING MCT-FNCT-FUNCTION-CODE
     MCT-FNCT-FUNCTION-POINTER MCT-FNCT-SCRATCH-PAD
     MCT-FNCT-TRIPLETS 62-BYTE-SCRATCH-FIELD
     197-BYTE-SCRATCH-FIELD MCT-FNCT-LITERAL-2
     35-BYTE-SCRATCH-FIELD MCT-FNCT-LITERAL-3
  MOVE MCT-FNCT-FUNCTION-POINTER TO BINARY-COUNTER-FIELD.
```

Index

S

T

W

Z