INTERNATIONAL STANDARD

ISO/IEC 1539-2

First edition
1994-12-15

Information technology — Programming languages — Fortran —

Part 2: Varying length character strings

Technologies de l'information — Langages de programmation — Fortran —

Partie 2: Chaînes de caractères de longueur variable

Reference number
ISO/IEC 1539-2:1994(E)

Contents

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standard ISO/IEC 1539-2 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*.

ISO/IEC 1539 consists of the following parts, under the general title *Information technology — Programming languages — Fortran*:

— *Part 1: Core Language Fortran*

— *Part 2: Varying length character strings*

Annexes A and B of this part of ISO/IEC 1539 are for information only.

Introduction

This part of ISO/IEC 1539 has been prepared by ISO/IEC JTC1/SC22/WG5, the technical working group for the Fortran language. This part of ISO/IEC 1539 is an auxiliary standard to ISO/IEC 1539 : 1991, which defines the latest revision of the Fortran language, and is the first part of the multipart Fortran family of standards; this part of ISO/IEC 1539 is the second part. The revised language defined by the above standard is informally known as Fortran 90.

This part of ISO/IEC 1539 defines the interface and semantics for a module that provides facilities for the manipulation of character strings of arbitrary and dynamically variable length. Annex A includes a possible implementation, in Fortran 90, of a module that conforms to this part of ISO/IEC 1539. It should be noted, however, that this is purely for purposes of demonstrating the feasibility and portability of this standard. The actual code shown in this annex is not intended in any way to prescribe the method of implementation, nor is there any implication that this is in any way an optimal portable implementation. The module is merely a fairly straightforward demonstration that a portable implementation is possible.

Information technology — Programming languages — Fortran — Part 2:

Varying length character strings

Section 1: General

1.1 Scope

This part of ISO/IEC 1539 defines facilities for use in Fortran for the manipulation of character strings of dynamically variable length. This part of ISO/IEC 1539 provides an auxiliary standard for the version of the Fortran language informally known as Fortran 90. The International Standard defining this revision of the Fortran language is

- ISO/IEC 1539 : 1991 "Programming Language Fortran"

This part of ISO/IEC 1539 is an auxiliary standard to that defining Fortran 90 in that it defines additional facilities to those defined intrinsically in the primary language standard. A processor conforming to the Fortran 90 standard is not required to also conform to this part of ISO/IEC 1539. However, conformance to this part of ISO/IEC 1539 assumes conformance to the primary Fortran 90 standard.

This part of ISO/IEC 1539 prescribes the name of a Fortran module, the name of a derived data type to be used to represent varying-length strings, the interfaces for the procedures and operators to be provided to manipulate objects of this type, and the semantics that are required for each of the entities made accessible by this module.

This part of ISO/IEC 1539 does not prescribe the details of any implementation. Neither the method used to represent the data entities of the defined type nor the algorithms used to implement the procedures or operators whose interfaces are defined by this part of ISO/IEC 1539 are prescribed. A conformant implementation may use any representation and any algorithms, subject only to the requirement that the publicly accessible names and interfaces conform to this part of ISO/IEC 1539, and that the semantics are as required by this part of ISO/IEC 1539 and those of ISO/IEC 1539 : 1991.

It should be noted that a processor is not required to implement this part of ISO/IEC 1539 in order to be a standard conforming Fortran processor, but if a processor implements facilities for manipulating varying length character strings, it is recommended that this be done in a manner that is conformant with this part of ISO/IEC 1539.

A processor conforming to this part of ISO/IEC 1539 may extend the facilities provided for the manipulation of varying length character strings as long as such extensions do not conflict with this part of ISO/IEC 1539 or with ISO/IEC 1539 : 1991.

A module, written in standard conforming Fortran, is included in Annex A. This module illustrates one way in which the facilities described in this part of ISO/IEC 1539 could be provided. This module is both conformant with the requirements of this part of ISO/IEC 1539 and, because it is written in

standard conforming Fortran, it provides a portable implementation of the required facilities. This module is included for information only and is not intended to constrain implementations in any way. This module is a demonstration that at least one implementation, in standard conforming and hence portable Fortran, is possible.

It should be noted that this part of ISO/IEC 1539 defines facilities for dynamically varying length strings of characters of default kind only. Throughout this part of ISO/IEC 1539 all references to intrinsic type **CHARACTER** should be read as meaning characters of default kind. Similar facilities could be defined for non-default kind characters by a separate, if similar, module for each such character kind.

This part of ISO/IEC 1539 has been designed, as far as is reasonable, to provide for varying length character strings the facilities that are available for intrinsic fixed length character strings. All the intrinsic operations and functions that apply to fixed length character strings have extended meanings defined by this part of ISO/IEC 1539 for varying length character strings. Also a small number of additional facilities are defined that are appropriate because of the essential differences between the intrinsic type and the varying length derived data type.

1.2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 1539. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this part of ISO/IEC 1539 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 646:1991, *Information technology — ISO 7-bit coded character set for information interchange.*

ISO/IEC 1539:1991, *Information technology — Programming languages — Fortran.*

Section 2 : Requirements

2.1 The Name of the Module

The name of the module shall be

```
ISO_VARYING_STRING
```

Programs shall be able to access the facilities defined by this part of ISO/IEC 1539 by the inclusion of `USE` statements of the form

```
USE ISO_VARYING_STRING
```

2.2 The Type

The type shall have the name

```
VARYING_STRING
```

Entities of this type shall represent values that are strings of characters of default kind. These character strings may be of any non-negative length and this length may vary dynamically during the execution of a program. There shall be no arbitrary upper length limit other than that imposed by the size of the processor and the complexity of the programs it is able to process. The characters representing the value of the string have positions 1,2,...,N, where N is the length of the string. The internal structure of the type shall be `PRIVATE` to the module.

2.3 Extended Meanings for Intrinsic Operators

The meanings for the intrinsic operators of:

assignment	`=`
concatenation	`//`
comparisons	`==, /=, <, <=, >=, >`

shall be extended to accept any combination of scalar operands of type `VARYING_STRING` and type `CHARACTER`. Note that the equivalent comparison operator forms `.EQ.`, `.NE.`, `.LT.`, `.LE.`, `.GE.`, and `.GT.` also have their meanings extended in this manner.

2.3.1 Assignment: An assignment of the form

```
var = expr
```

shall be defined for scalars with the following type combinations:

```
VARYING_STRING = VARYING_STRING
VARYING_STRING = CHARACTER
     CHARACTER = VARYING_STRING
```

Action: The characters that are the value of the expression `expr` become the value of the variable `var`. There are two cases:

Case(i) : Where the variable is of type `VARYING_STRING`, the length of the variable becomes that of the expression.

Case(ii) : Where the variable is of type `CHARACTER`, the rules of intrinsic assignment to a Fortran character variable apply. Namely, if the expression string is longer than the declared length of the character variable, only the left-most characters are assigned. If the character variable is longer than that of the string expression, it is padded on the right with blanks.

2.3.2 Concatenation: The concatenation operation

```
string_a // string_b
```

shall be defined for scalars with the following type combinations:

```
VARYING_STRING // VARYING_STRING
VARYING_STRING // CHARACTER
     CHARACTER // VARYING_STRING
```

The values of the operands are unchanged by the operation.

Result Attributes: scalar of type `VARYING_STRING`.

Result Value: The result value is a new string whose characters are the same as those produced by concatenating the operand character strings in the order given.

2.3.3 Comparisons: Comparisons of the form

```
string_a .OP. string_b
```

where `.OP.` represents any of the operators `==`, `/=`, `<`, `<=`, `>=`, or `>` shall be defined for scalar operands with the following type combinations:

`VARYING_STRING .OP. VARYING_STRING`,

`VARYING_STRING .OP. CHARACTER`, or

`CHARACTER .OP. VARYING_STRING`.

The values of the operands are unchanged by the operation.

Note that the equivalent operator forms `.EQ.`, `.NE.`, `.LT.`, `.LE.`, `.GE.`, and `.GT.` also have their meanings extended in this manner.

Result Attributes: scalar of type default `LOGICAL`.

Result Value: The result value is true if `string_a` stands in the indicated relation to `string_b` and is false otherwise. The collating sequence used for the inequality comparisons is that defined by the processor for characters of default kind. If `string_a` and `string_b` are of different lengths, the comparison is done as if the shorter string were padded on the right with blanks.

2.4 Extended Meanings for Generic Intrinsic Procedures

The generic intrinsic procedures `LEN`, `CHAR`, `ICHAR`, `IACHAR`, `TRIM`, `LEN_TRIM`, `ADJUSTL`, `ADJUSTR`, `REPEAT`, `LLT`, `LLE`, `LGE`, `LGT`, `INDEX`, `SCAN`, and `VERIFY` shall have their meanings extended to include the appropriate scalar argument type combinations involving `VARYING_STRING` and `CHARACTER`. The results produced in each case are also scalar.

2.4.1 The LEN procedure: The generic function reference of the form

```
LEN(string)
```

shall be added.

Description: returns the length of a character string.

Argument: `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by the procedure.

Result Attributes: scalar of type default `INTEGER`.

Result Value: The result value is the number of characters in `string`.

2.4.2 The CHAR procedure: The generic function references of the form

```
CHAR(string)
CHAR(string,length)
```

shall be added.

Description: converts a varying string value to default character.

Arguments:

string - is of type **VARYING_STRING**

length - is of type default **INTEGER**.

The arguments are scalars and are unchanged by the procedure.

Result Attributes: scalar of type default **CHARACTER**. If **length** is absent, the result has the same length as **string**. If **length** is present, the result has the length specified by the argument **length**.

Result Value:

Case(i) : If **length** is absent, the result is a copy of the characters in the argument **string**.

Case(ii) : If **length** is present, the result is a copy of the characters in the argument **string** that may have been truncated or padded. If **string** is longer than **length**, the result is truncated on the right. If **string** is shorter than **length**, the result is padded on the right with blanks. If **length** is less than one, the result is of zero length.

2.4.3 The ICHAR procedure: The generic function reference of the form

ICHAR(c)

shall be added.

Description: returns the position of a character in the processor defined collating sequence.

Argument: **c** is a scalar of type **VARYING_STRING** and of length exactly one. The argument is unchanged by the procedure.

Result Attributes: scalar of type default **INTEGER**.

Result Value: The result value is the position of the character **c** in the processor defined collating sequence for default characters. That is, the result value is **ICHAR(CHAR(c))**.

2.4.4 The IACHAR procedure: The generic function reference of the form

IACHAR(c)

shall be added.

Description: returns the position of a character in the collating sequence defined by the International Standard ISO 646 : 1991.

Argument: **c** is a scalar of type **VARYING_STRING** and of length exactly one. The argument is unchanged by the procedure.

Result Attributes: scalar of type default **INTEGER**.

Result Value: The result value is the position of the character **c** in the collating sequence defined by the International Standard ISO 646 : 1991 for default characters. If the character **c** is not defined in the standard set, the result is processor dependent but is always equal to **IACHAR(CHAR(c))**.

2.4.5 The TRIM procedure: The generic function reference of the form

TRIM(string)

shall be added.

Description: removes trailing blanks from a string.

Argument: **string** is a scalar of type **VARYING_STRING**. The argument is unchanged by the procedure.

Result Attributes: scalar of type **VARYING_STRING**.

Result Value: The result value is the same as **string** except that any trailing blanks have been deleted. If the argument **string** contains only blank characters or is of zero length, the result is a zero-length string.

2.4.6 The LEN_TRIM procedure: The generic function reference of the form

```
LEN_TRIM(string)
```

shall be added.

Description: returns the length of a string not counting any trailing blanks.

Argument: `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by the procedure.

Result Attributes: scalar of type default `INTEGER`.

Result Value: The result value is the position of the last non-blank character in `string`. If the argument `string` contains only blank characters or is of zero length, the result is zero.

2.4.7 The ADJUSTL procedure: The generic function reference of the form

```
ADJUSTL(string)
```

shall be added.

Description: adjusts to the left, removing any leading blanks and inserting trailing blanks.

Argument: `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by the procedure.

Result Attributes: scalar of type `VARYING_STRING`.

Result Value: The result value is the same as `string` except that any leading blanks have been deleted and the same number of trailing blanks inserted.

2.4.8 The ADJUSTR procedure: The generic function reference of the form

```
ADJUSTR(string)
```

shall be added.

Description: adjusts to the right, removing any trailing blanks and inserting leading blanks.

Argument: `string` is a scalar of type `VARYING_STRING`. The argument is unchanged by the procedure.

Result Attributes: scalar of type `VARYING_STRING`.

Result Value: The result value is the same as `string` except that any trailing blanks have been deleted and the same number of leading blanks inserted.

2.4.9 The REPEAT procedure: The generic function reference of the form

```
REPEAT(string,ncopies)
```

shall be added.

Description: concatenates several copies of a string.

Arguments:

`string` - is a scalar of type `VARYING_STRING`,

`ncopies` - is a scalar of type default `INTEGER`.

The value of `ncopies` must not be negative. The arguments are unchanged by the procedure.

Result Attributes: scalar of type `VARYING_STRING`.

Result Value: The result value is the string produced by repeated concatenation of the argument `string`, producing a string containing `ncopies` copies of `string`. If `ncopies` is zero, the result is of zero length.

2.4.10 Comparison procedures: The set of generic function references of the form

```
LOP(string_a,string_b)
```

shall be added, where `OP` stands for one of:

```
LT   -   less than
LE   -   less than or equal to
GE   -   greater than or equal to
GT   -   greater than
```

Description: compares the lexical ordering of two strings based on the ISO 646 : 1991 collating sequence.

Arguments: `STRING_A` and `STRING_B` are scalars of one of the type combinations:

`VARYING_STRING` and `VARYING_STRING`,

`VARYING_STRING` and `CHARACTER`, or

`CHARACTER` and `VARYING_STRING`.

The arguments are unchanged by the procedure.

Result Attributes: scalar of type default `LOGICAL`.

Result Value: The result value is true if `STRING_A` stands in the indicated relationship to `STRING_B`, and is false otherwise. The collating sequence used to establish the ordering of characters for these procedures is that of the International Standard ISO 646 : 1991. If `STRING_A` and `STRING_B` are of different lengths, the comparison is done as if the shorter string were padded on the right with blanks. If either argument contains a character `C` not defined by the standard, the result value is processor dependent and based on the collating value for `IACHAR(C)`. Zero length strings are considered to be lexically equal.

2.4.11 The INDEX procedure: The generic function reference of the form

```
INDEX(string,substring,back)
```

shall be added.

Description: returns an integer that is the starting position of a substring within a string.

Arguments: `STRING` and `SUBSTRING` are scalars of one of the type combinations:

`VARYING_STRING` and `VARYING_STRING`,

`CHARACTER` and `VARYING_STRING`, or

`VARYING_STRING` and `CHARACTER`.

`BACK` - is a scalar of type default `LOGICAL` and is `OPTIONAL`.

The arguments are unchanged by the procedure.

Result Attributes: scalar of type default `INTEGER`.

Result value:

Case(i) : If `BACK` is absent or is present with the value false, the result is the minimum positive value of `I` such that,
`EXTRACT(string,I,I+LEN(substring)-1)==substring`,
or zero if there is no such value.
Zero is returned if `LEN(string)<LEN(substring)`, and one is returned if `LEN(substring)==0`.

Case(ii) : If `BACK` is present with the value true, the result is the maximum value of `I` less than or equal to `LEN(string)-LEN(substring)+1` such that
`EXTRACT(string,I,I+LEN(substring)-1)==substring`,
or zero if there is no such value.
Zero is returned if `LEN(string)<LEN(substring)`, and `LEN(string)+1` is returned if `LEN(substring)==0`.

2.4.12 The SCAN procedure: The generic function reference of the form

```
SCAN(string,set,back)
```

shall be added.

Description: scans a string for any one of the characters in a set of characters.

Arguments: **string** and **set** are scalars of one of the type combinations:
VARYING_STRING and **VARYING_STRING**,
VARYING_STRING and **CHARACTER**, or
CHARACTER and **VARYING_STRING**.
back - is a scalar of type default **LOGICAL** and is **OPTIONAL**.
The arguments are unchanged by the procedure.
Result Attributes: scalar of type default **INTEGER**.
Result Value:
Case(i) : If **back** is absent or is present with the value false and if **string** contains at least one character that is in **set**, the value of the result is the position of the left-most character of **string** that is in **set**.
Case(ii) : If **back** is present with the value true and if **string** contains at least one character that is in **set**, the value of the result is the position of the right-most character of **string** that is in **set**.
Case(iii) : The value of the result is zero if no character of **string** is in **set** or if the length of either **string** or **set** is zero.

2.4.13 The VERIFY procedure: The generic function reference of the form
VERIFY(string,set,back)
shall be added.
Description: verifies that a string contains only characters from a given set by scanning for any character not in the set.

Arguments: **string** and **set** are scalars of one of the type combinations:
VARYING_STRING and **VARYING_STRING**,

VARYING_STRING and **CHARACTER**, or
CHARACTER and **VARYING_STRING**.

back - is a scalar of type default **LOGICAL** and is **OPTIONAL**.
The arguments are unchanged by the procedure.
Result Attributes: scalar of type default **INTEGER**.
Result Value:
Case(i) : If **back** is absent or is present with the value false and if **string** contains at least one character that is not in **set**, the value of the result is the position of the left-most character of **string** that is not in **set**.
Case(ii) : If **back** is present with the value true and if **string** contains at least one character that is not in **set**, the value of the result is the position of the right-most character of **string** that is not in **set**.
Case(iii) : The value of the result is zero if each character of **string** is in **set** or if the length of **string** is zero.

2.5 Additional Generic Procedure for Type Conversion

An additional generic procedure shall be added to convert scalar intrinsic fixed-length character values into scalar varying-length string values.

2.5.1 The VAR_STR procedure: The generic function reference of the form
VAR_STR(char)
shall be provided.
Description: converts an intrinsic fixed-length character value into the equivalent varying-length string value.

Argument: `char` is a scalar of type default `CHARACTER` and may be of any length. The argument is unchanged by the procedure.
Result Attributes: scalar of type `VARYING_STRING`.
Result Value: The result value is the same string of characters as the argument.

2.6 Additional Generic Procedures for Input/Output

The following additional generic procedures shall be provided to support input and output of varying-length string values with formatted sequential files.

`GET`	-	input part or all of a record into a string
`PUT`	-	append a string to an output record
`PUT_LINE`	-	append a string to an output record and end the record

2.6.1 The GET procedure: The generic subroutine references of the forms

```
CALL GET(string,maxlen,iostat)
CALL GET(unit,string,maxlen,iostat)
CALL GET(string,set,separator,maxlen,iostat)
CALL GET(unit,string,set,separator,maxlen,iostat)
```

shall be provided.
Description: reads characters from an external file into a string.
Arguments:

`string`	-	is of type `VARYING_STRING`,
`maxlen`	-	is of type default `INTEGER` and is `OPTIONAL`,
`unit`	-	is of type default `INTEGER`,
`set`	-	is either of type `VARYING_STRING` or of type `CHARACTER`,
`separator`	-	is of type `VARYING_STRING` and is `OPTIONAL`,
`iostat`	-	is of type default `INTEGER` and is `OPTIONAL`.

All arguments are scalar. The argument `unit` specifies the input unit to be used. It must be connected to a formatted file for sequential read access. If the argument `unit` is omitted, the default input unit is used. The arguments `maxlen`, `unit`, and `set` are unchanged by the procedure.
Action: The `GET` procedure causes characters from the connected file, starting with the next character in the current record if there is a current record or the first character of the next record if not, to be read and stored in the variable `string`. The end of record always terminates the input but input may be terminated before this. If `maxlen` is present, its value indicates the maximum number of characters that will be read. If `maxlen` is less than or equal to zero, no characters will be read and `string` will be set to zero length. If `maxlen` is absent, a maximum of `HUGE(1)` is used. If the argument `set` is provided, this specifies a set of characters the occurrence of any of which will terminate the input. This terminal character, although read from the input file, will not be included in the result string. The file position after the data transfer is complete, is after the last character that was read. If the argument `separator` is present, the actual character found which terminates the transfer is returned in `separator`. If the transfer is terminated other than by the occurrence of a character in `set`, a zero length string is returned in `separator`. If the transfer is terminated by the end of record being reached, the file is positioned after the record just read. If present, the argument `iostat` is used to return the status resulting from the data transfer. A zero value is returned if a valid read operation occurs and the end-of-record is not reached, a positive value if an error occurs, and a negative value if an end-of-file or end-of-record condition occurs. Note, the negative value returned for an

end-of-file condition must be different from that returned for an end-of-record condition. If **iostat** is absent and an error or end-of-file condition occurs, the program execution is terminated.

2.6.2 The PUT procedure: The generic subroutine references of the forms

```
CALL PUT(string,iostat)
CALL PUT(unit,string,iostat)
```

shall be provided.

Description: writes a string to an external file.

Arguments:

string - is either of type **VARYING_STRING** or type **CHARACTER**,

unit - is of type default **INTEGER**,

iostat - is of type default **INTEGER** and is **OPTIONAL**.

All arguments are scalar. The argument **unit** specifies the output unit to be used. It must be connected to a formatted file for sequential write access. If the argument **unit** is omitted, the default output unit is used. The arguments **unit** and **string** are unchanged by the procedure.

Action: The **PUT** procedure causes the characters of **string** to be appended to the current record, if there is a current record, or to the start of the next record if there is no current record. The last character transferred becomes the last character of the current record, which is the last record of the file. If present, the argument **iostat** is used to return the status resulting from the data transfer. A zero value is returned if a valid write operation occurs, and a positive value if an error occurs. If **iostat** is absent and anything other than a valid write operation occurs, the program execution is terminated.

2.6.3 The PUT_LINE procedure: The generic subroutine references of the forms

```
CALL PUT_LINE(string,iostat)
CALL PUT_LINE(unit,string,iostat)
```


shall be provided.

Description: writes a string to an external file and ends the record.

Arguments:

string - is either of type **VARYING_STRING** or type **CHARACTER**

unit - is of type default **INTEGER**

iostat - is of type default **INTEGER** and is **OPTIONAL**.

All arguments are scalar. The argument **unit** specifies the output unit to be used. It must be connected to a formatted file for sequential write access. If the argument **unit** is omitted, the default output unit is used. The arguments **unit** and **string** are unchanged by the procedure.

Action: The **PUT_LINE** procedure causes the characters of **string** to be appended to the current record, if there is a current record, or to the start of the next record if there is no current record. Following completion of the data transfer, the file is positioned after the record just written, which becomes the previous and last record of the file. If present, the argument **iostat** is used to return the status resulting from the data transfer. A zero value is returned if a valid write operation occurs, and a positive value if an error occurs. If **iostat** is absent and anything other than a valid write operation occurs, the program execution is terminated.

2.7 Additional Generic Procedures for Substring Manipulation

The following additional generic procedures shall be provided to support the manipulation of scalar substrings of scalar varying-length strings.

INSERT	-	insert a substring into a string
REPLACE	-	replace a substring in a string
REMOVE	-	remove a section of a string
EXTRACT	-	extract a section from a string
SPLIT	-	split a string into two at the occurrence of a separator

2.7.1 The INSERT procedure: The generic function reference of the form

```
INSERT(string,start,substring)
```

shall be provided.

Description: inserts a substring into a string at a specified position.

Arguments:

string	-	is either type **VARYING_STRING** or type default **CHARACTER**,
start	-	is type default **INTEGER**,
substring	-	is either type **VARYING_STRING** or type default **CHARACTER**.

All arguments are scalars. The arguments are unchanged by the procedure.

Result Attributes: scalar of type **VARYING_STRING**.

Result Value: The result value is a copy of the characters of the argument **string** with the characters of **substring** inserted into the copy of **string** before the character at the character position **start**. If **start** is greater than **LEN(string)**, the value **LEN(string)+1** is used for **start** and **substring** is appended to the copy of **string**. If **start** is less than one, the value one is used for **start** and **substring** is inserted before the first character of the copy of **string**.

2.7.2 The REPLACE procedure: The generic function references of the forms

```
REPLACE(string,start,substring)
REPLACE(string,start,finish,substring)
REPLACE(string,target,substring,every,back)
```

shall be provided.

Description: replaces a subset of the characters in a string by a given substring. The subset may be specified either by position or by content.

Arguments:

string	-	is either of type **VARYING_STRING** or type default **CHARACTER**,
start	-	is of type default **INTEGER**,
finish	-	is of type default **INTEGER**,
substring	-	is either of type **VARYING_STRING** or type default **CHARACTER**,
target	-	is either of type **VARYING_STRING** or type default **CHARACTER**,
every	-	is of type default **LOGICAL**, and is **OPTIONAL**,
back	-	is of type default **LOGICAL**, and is **OPTIONAL**.

All arguments are scalar. The argument **target** must not be of zero length. In all cases the arguments are unchanged by the procedure.

Result Attributes: scalar of type **VARYING_STRING**.