DIM({*AUTO|*CTDATA|*VAR:}numeric_constant)

The DIM keyword defines the number of elements in an array, table, a prototyped parameter, array data structure, or a return value on a prototype or procedure-interface definition.

[Image: delta.gif]The numeric constant is required unless the first parameter is *CTDATA. It must have zero (0) decimal positions. It can be a literal, a named constant or a built-in function.[Image: deltaend.gif]

The constant value does not need to be known at the time the keyword is processed, but the value must be known at compile-time.

When DIM is specified on a data structure definition, the data structure must be a qualified data structure, and subfields must be referenced as fully qualified names, i.e. "dsname(x).subf". Other array keywords, such as CTDATA, FROMFILE, TOFILE, and PERRCD are not allowed with an array data structure definition.

[Image: delta.gif]

DIM(*CTDATA)

When the first parameter for the DIM keyword is *CTDATA, the dimension of the array or table is determined by the number of records in the compile-time data for the array or table. The CTDATA keyword is not required.

In the following example, array ARR is defined with DIM(*CTDATA) 1 . There are three records in the compile-time data for the array 1 , so the dimension of the array is 3.

```
DCL-S arr CHAR(10) DIM(*CTDATA); //   1


**CTDATA arr    2
abc
def
ghi
```

Rules for arrays and tables defined with DIM(*CTDATA):

- DIM(*CTDATA) is only valid for standalone arrays and tables.
- The compile-time data must have one element per record. If the PERRCD keyword is specified, the parameter must be 1.
- If there is an alternate array or table, it must also be defined with DIM(*CTDATA).
- The compile-time data must be indicated by **CTDATA.
- The compile-time data must precede all other data.

[Image: deltaend.gif]

[Image: delta.gif]

Varying-dimension arrays

When the first parameter for the DIM keyword is *AUTO or *VAR, the dimension of the array is variable.

The second parameter for the DIM keyword represents the maximum number of elements for the array.

The number of elements for the array is initialized to zero. Any initialization values for the array are used when the dimension of the array is increased.

See Varying-dimension arrays for more information about varying-dimension arrays.

[Image: deltaend.gif]

- Varying-dimension arrays

Parent topic: Definition-Specification Keywords

[Image: delta.gif]

Varying-dimension arrays

A varying-dimension array is defined with DIM(*AUTO:maximum_elements) or DIM(*VAR:maximum_elements). The second parameter indicates the maximum number of elements in the array.
The number of elements of the array is initialized to zero. The number of elements in the array can be changed using the %ELEM built-in function. In the following example, the array has a maximum of 100 elements (1). The current number of elements in the array is set to 25 using the %ELEM built-in function (1). DCL-S

```
var_array1 CHAR(10) DIM(*VAR : 100); //  2


%ELEM(var_array1) = 25; //  2
```

When a varying-dimension array is defined with DIM(*AUTO):
- The number of elements can also be increased by assigning a value to an element that is greater than the current number of elements.
- You can specify index *NEXT when the array is the target of an assignment statement. See Example: Use *NEXT as an index for an array defined with DIM(*AUTO).

In the following example, the array is defined with a maximum of 1000 elements (1). When a value is assigned to element 100 of the array (2), the current number of elements increases to 100. The previous number of elements was zero, so elements 1 to 99 are initialized with the initialization value of the array.
When a value is assigned to element 50 of the array (3), the number of elements does not change, because 50 is less than the current number of elements. DCL-S

```
auto_array1 CHAR(10) DIM(*AUTO : 1000);  //  1


auto_array1(100) = 'abc'; //  2
auto_array1(50) = 'abc';  //  3
```

The current number of elements can also be reduced using %ELEM. In the following example, the current number of elements is changed to 100 due to the assignment to element 100 (1). It is reduced to 25 by the assignment to %ELEM (2). DCL-S

```
auto_array2 CHAR(10) DIM(*AUTO : 1000);


auto_array2(100) = 'abc'; //  1
%elem(auto_array2) = 25;  //  2
```

Restrictions for varying-dimension arrays

- A varying-dimension array can only be a top-level definition, either a standalone array or a data structure array. Tables, multiple-occurrence data structures, subfields, procedure return values, and procedure parameters are not allowed.
- A varying-dimension array cannot be used in fixed-form calculations.
- A varying-dimension array cannot be based on a pointer.

- A varying-dimension array cannot be imported or exported.
- When a varying-dimension array is passed as a parameter to a prototyped procedure or program, the parameter must be defined with OPTIONS(*VARSIZE).
- A varying-dimension array cannot have type Object.
- A varying-dimension array cannot be null-capable.
- A subfield of a varying-dimension data structure array cannot be null-capable unless the null-indicator is also a subfield in the same data structure.
- A varying-dimension array cannot appear on an Input specification unless an index is specified.
- A varying-dimension array cannot appear on an Output specification unless an index is specified.
- A varying-dimension array element cannot appear on a Lookahead Input specification.
- A varying-dimension array cannot be a compile-time array or a pre-run array. The CTDATA and FROMFILE keywords cannot be used.

Example: Use *NEXT as an index for an array defined with DIM(*AUTO)

In the following example, the array is defined with DIM(*AUTO). Assume that file CUSTFILE has a field NAME. During the assignment to CUSTNAMES(*NEXT) (1), the dimension is increased by one, and the value of NAME is assigned to the new element.

```
DCL-F custfile;
DCL-S custNames VARCHAR(10) DIM(*AUTO : 1000);


READ custfile;
DOW NOT %EOF;
  custNames(*next) = NAME;  //  1
  READ custfile;
ENDDO;
```

Considerations for varying-dimension arrays

- Warning: If you use %ADDR to obtain the address of the array or an array element, take care to obtain the address again if the number of elements changes.
- Warning: The debugger is currently unaware that the dimension of the array is variable. You must avoid working in the debugger with elements that are not available in the array.
  While debugging, you can evaluate _QRNU_VARDIM_ELEMS_arrayname to determine the current number of elements. Note that changing this value in the debugger will have no effect on the actual current number of elements in the array. For example, if the array is called myArray, your debug session might look like the following:

```
> EVAL _qrnu_vardim_elems_myarr
 _QRNU_VARDIM_ELEMS_MYARR = 0
> EVAL _qrnu_vardim_elems_myarr
 _QRNU_VARDIM_ELEMS_MYARR = 3
> EVAL myarr(1..3)
```

```
MYARR(1) = 'Jack  '
MYARR(2) = 'Sally '
MYARR(3) = 'Tom   '
```

Example: Increase the current number of elements without changing the value of the new elements

Normally, when the number of elements is increased, the new elements are initialized with the initialization value of the array. If you know that the data in the new elements is already correct, you can prevent any initialization of the new elements by specifying *KEEP as the second parameter of %ELEM. In the following example, when the current number of elements is set to 5 (1), the new elements are initialized to the initialization value '?'.
When the current number of elements is set to 3 (2), elements 4 and 5 have the values 'd' and 'e' due to the previous assignment statements.
When the current number of elements is set to 4, and *KEEP is specified as the second parameter of %ELEM (3), the value of the new element 4 is not changed, so it has the value 'd'.
When the current number of elements is set to 5, and *KEEP is not specified as the second parameter of %ELEM (4), the value of the new element 5 is set to the initialization value '?'. `DCL-S var_array2 CHAR(10) INZ('?') DIM(*VAR : 1000);`

```
%elem(var_array2) = 5;       //  1
var_array2(1) = 'a';
var_array2(2) = 'b';
var_array2(3) = 'c';
var_array2(4) = 'd';
var_array2(5) = 'e';
%elem(var_array2) = 3;       //  2
%elem(var_array2:*KEEP) = 4; //  3
%elem(var_array2) = 5;       //  4
```

Example: Ensure sufficient elements are allocated to the array

You can increase the number of elements that are allocated to the array without changing the current number of elements in the array by specifying *ALLOC as the second parameter of %ELEM.
In the following example, the array is passed by reference to a procedure that will set values in some of the array elements. Before calling the procedure, the number of elements allocated to the array is set to 100 (1). However, the current number of elements remains zero.
The second parameter passed to the procedure specifies the number of elements that the procedure can set. The procedure returns the number of elements that were set (2).
%ELEM is used with *KEEP to set the new current number of elements (3).

```
DCL-S var_array3 CHAR(10) INZ('?') DIM(*VAR : 1000);
DCL-S num_elems INT(10);
DCL-PR proc INT(10);
  array CHAR(10) DIM(1000) OPTIONS(*VARSIZE);
  max_elems INT(10) VALUE;
END-PR;
%elem(var_array3:*ALLOC) = 100;      //  1
num_elems = proc(var_array3 : 100);  //  2
%elem(var_array2:*KEEP) = num_elems; //  3
```

Parent topic:DIM({*AUTO|*CTDATA|*VAR:}numeric_constant)

[Image: deltaend.gif]